DESIGN AND SIMULATION OF

INTELLIGENT DEDUCTIVE DISTRIBUTED DATABASES

FOR INTEGRATED MEDICAL SYSTEMS

by

NARONGRIT WARAPORN

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2006

UMI Number: 3231938

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

This manuscript has been read and accepted for the
Graduate Faculty in Computer Science in satisfaction of the
dissertation requirement for the degree of Doctor of Philosophy.

Professor Syed V. Ahamed

Date — Chair of Examining Committee

Professor Theodore Brown

Date — Executive Officer

Professor Michael E. Kress

Professor Michael Anshel

Professor Amotz Bar-Noy

Professor Nancy Griffeth

Dr. Victor B. Lawrence
Supervision Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

DESIGN AND SIMULATION OF

INTELLIGENT DEDUCTIVE DISTRIBUTED DATABASES

FOR INTEGRATED MEDICAL SYSTEMS

by

NARONGRIT WARAPORN

Adviser: Professor Syed V. Ahamed

Internet technology has dominated in networking. Internet Protocol (IP) is popular with network scientists and engineers because of the popularity of the Internet. Dedicated networks have become less frequent. In the past, researchers concentrated on building a dedicated network for a specific area, such as a medical network. However the high cost of building dedicated network for one particular interest is not a good investment of resources.

In recent years, many researchers have studied and simulated many medical problems such as work flow of patient appointment, healthcare architecture, and smart-card for health services. The databases have been advanced to the coordinated databases, called distributed databases. Also the Internet has been enormously researched and used throughout the world. However, those works were done on a dedicated network or Internet that does not store medical information in the pattern of a knowledge-based deductive structure. If we can construct and gather medical information around the Internet together and process it as the collection of knowledge, then it would be possible

to bring the intelligence into the Internet and benefit the integrated medical system throughout the world. At the same time, many applications of Intelligent Internet have been proposed for sharing educational, financial and medical resources.

In this dissertation, we propose and build a new medical system called Medical Knowledge Technology (MKT). Deductive medical analysis among participating nodes sharing knowledge via Internet is deployed. In addition, simulated results tested on the MKT system at many geographical levels; local, regional, national, and global are presented.

Acknowledgments

First and foremost, I would like to thank Prof. Syed Ahamed, my mentor, who has guided me throughout the process of this dissertation with expertise and compassion. His insights and assessments have always been helpful and instructive. It has been a pleasure and an honor to work with him.

I also would like to thank all committee members of my dissertation committee: Prof. Michael Kress who has given me both scientific and general suggestions on my thesis (his suggestion of time table has been posted in front of my desk to remind me to work hard on my research), Dr. Victor Lawrence and Prof. Michael Anshel who provided support all these years, and Prof. Nancy Griffiths, and Prof. Amotz Bar-Noy; for coming to my dissertation defense.

Without supporting from the Computer Science department, I would not have been able to do my research. Special thanks to Prof. Stanley Habib for accepting me and finding the financial support, to Prof. Brown for encouraging and motivating me to pursue the work to completion, and to Joe Driscoll for all his help in every process of doctoral study.

I also thank Prof. Aaron Tenenbaum, Brooklyn College and the research foundation of CUNY for financial support which is not always easy to handle when many graduate students are looking for it. I sincerely appreciated this opportunity to teach and be a better teacher by learning from great teachers such as Prof. Keith Harrow, and Prof. Paula Whitlock.

My intellectual journey has also received help from our fellow graduate students at the Graduate Center; Lin Lueng, Jonathan Tang, Eryuan Huang, Jayson Rome, G. Kahanda and many more. Our friendship is inestimable.

Many parts of the results in this dissertation are generated by the help of many friends around the world: Lillian Stallone, Thamchai Techawaro (Chui), Tussaneekorn Srichaiwong (Idd), Suppamas Supawannapong (Mas), Wannida Suntreerat (Ying) and Boontita Suchukorn (Joy). Their participation in running my program really helped me.

One of the most gratitude is to Dr. Frank Stallone who reviewed my thesis from the first to the last page, from many days before to many days after Christmas, and from many months before to the month after his cancer operation. His tremendous amount of time is very generous and his insightful comments and discussions are invaluable.

Last but the most is the support from my family. My dad, Aron Waraporn, and my mom, Sopha Waraporn, who know the importance of education, provided us, all four children; To-Tid-Tom-Toye, the utmost support all their whole lives. This is for you, mom and dad. My love to all my brothers and sister: my big brother, Watcharapong Waraporn, who encourages me to study the Ph.D., my sister, Nattawan Waraporn, and my brother–in-law, Benjamin Stallone who shelter me from the day one in United States and my little brother, Pitsanu Waraporn, who supports me all along.

The last person of my gratitude is to my lovely wife, Chanakarn Waraporn (Ja). Without her patient waiting through the ups and downs of the long months, and the love she provided, this dissertation would not be done. Thank you very much and I love you.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

Since the 1980s, the growth with which the internet has been adopted by researchers, business, and individuals has been exponential. At this state, the internet has been advanced into Intelligent Internet. Applications of Intelligent Internet have been proposed by many scientists. In this chapter, we investigate the revolution of medical system, Intelligent Network, and distributed databases. At the end of the chapter, we conclude with our contribution of this dissertation into another milestone of Intelligent Internet on medical systems.

1.1. Medical Systems

One of the major aims of the mankind is to prolong life by treating curable diseases, preventing diseases before they start, and research to find treatments for incurable diseases. To remedy, physicians and nurses must be well trained and continuously educated for the new medical discoveries from researchers. To prevent the spread of infectious disease, a team of medical experts, emergency teams and local and federal governments must cooperate. While these professionals are working together, there are many innovations of computer technology to support their needs. The following are terminology of health care system industries.

- **Document Management Systems** A document management system manages individual documents for each patient within the physician's office. It dramatically cuts time for searching information. Imaging and voice recognition software can be a part of document management systems.

Figure 1- 1 Computerized Patient Records from www.phisiciansnews.com

- **Electronic and Computerized Patient Records, EPR and CPR** As computers start appearing in most physicians' clinic, more physicians become acquainted with information and Internet usage of computer-based records including all clinical and administrative information about patients. It offers methods of storing, manipulating, searching, and communicating all kinds of information including text, sound, image, video, and tactile senses of patients. Its computerized notes can facilitates the physician-patient relationship for which physicians and patients share and contribute. However, there must be trust between physician and patient. Patients must be informed of how their confidential records are stored, especially when their records can be retrieve via Internet, and, if appropriate, give their consent.

The term *electronic patient record* (EPR), an older term and still in use, is being replaced by the term *computerized patient record*.

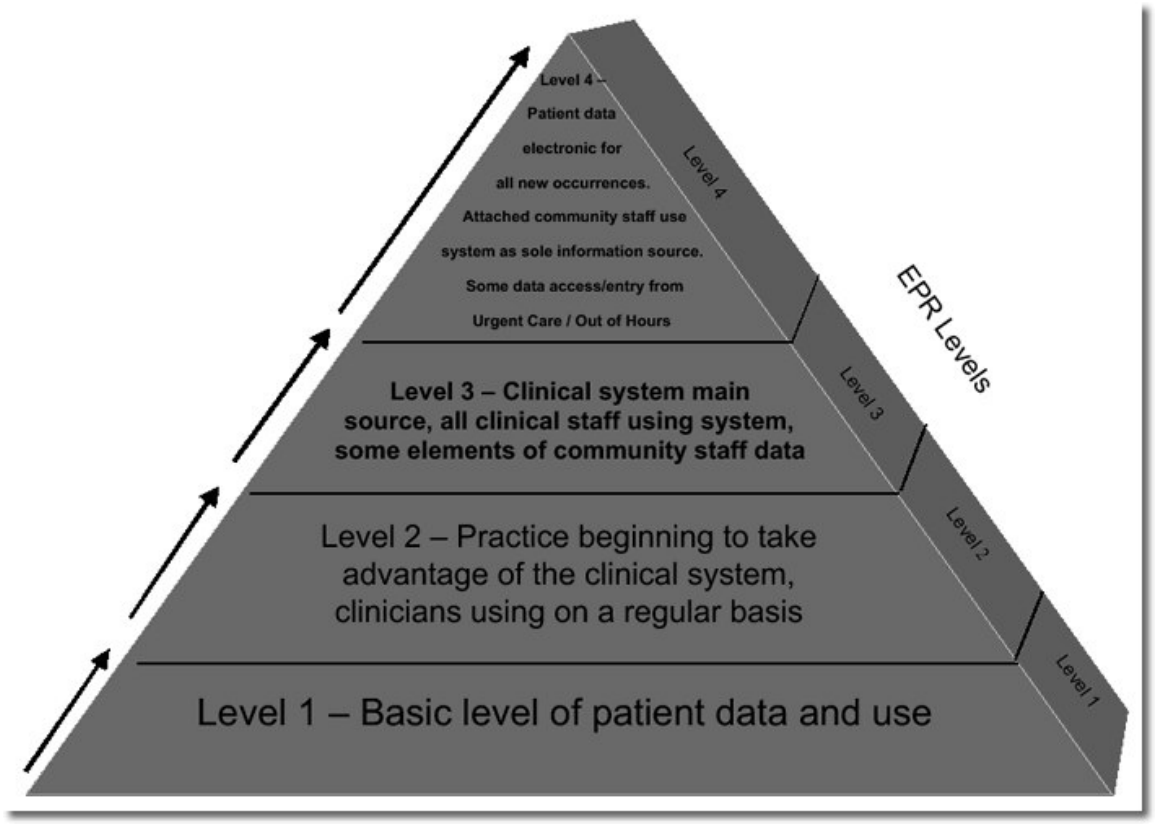


Figure 1- 2 Electronic Patient Record Level from www.eguidelines.co.uk

- **Automated and Computerized Medical Records, AMR and CMR** Physicians and staffs scan paper of patients into computer as a file. But the paper was kept for legal purposes. Its main purpose is for the administrative benefits such as patient/physician scheduling, clinical form maintaining, billing, etc. Both terms, automated medical records and computerized medical records, are infrequently used at the present because of emerging of electronic and computerized patient records that place more emphasis on an individual patient. They were commonly used in early 1990's. However, the term *automated medical records* has been resumed again by some health agencies that research disease occurrence for planning purposes, resource allocating and automated reporting systems. The illness's syndromes are of interest because it takes several days' advanced warning for a serious problem, such as anthrax.

- **Electronic Medical Records, EMR** The electronic medical record is a sophistication of the document management systems. Not only paperless system is improved, but also information import from a variety of sources such as laboratories, radiology facilities, and export to pharmacies. It cuts cost of storage, improves the accuracy, and efficiency and quality of documentation. Mobile/wireless healthcare applications run on Internet-based electronic medical record software for universal accessibility. The physician can view patient records via desktop, PDA handheld device, or tablet PC. Reporting system helps the clinical business analysis and future planning that allows user to formulate a report by demographics, insurances, ICD (International Classification of Diseases) codes or CPT (Current Procedural Terminology) codes, etc. As for patients, electronic medical record has ability to transmit information to a pharmacy regarding an individual prescription of a patient.

- **Continuity of Care Records, CCR** Continuity of Care Record is a standard specification being developed jointly by American Society for Testing and Materials International (ASTM International), the Massachusetts Medical Society (MMS), the Health Information Management and Systems Society (HIMSS), the American Academy of Family Physicians (AAFP), and the American Academy of Pediatrics. It is developed and enhanced in response to the need to organize and make transportable a set of basic information about patient's health care that is accessible to both physicians and patients. It is intended to foster and improve continuity of patient care to minimize medical errors and to ensure at least a minimum level of quality of transportability when a patient is referred or transferred to, or seen by another provider. The patient information includes current medical condition, diagnoses, allergies, medications, history illnesses, recent

procedures, recent care provided, as well as recommendations for future care and the reason for referral or transfer. The CCR may be used to accelerate the exchange of clinical information among providers, institutes, and other health care associations. It can be used by a patient to review as a brief summary of recent care. Since CCR is stored in the XML format, it is readable by both human and machine. The electronic medical record can simply download or upload and collect all relevant data from and to the CCR document. Various formats of data content, such as in web browser, PDF reader, or word processor can be displayed and printed.

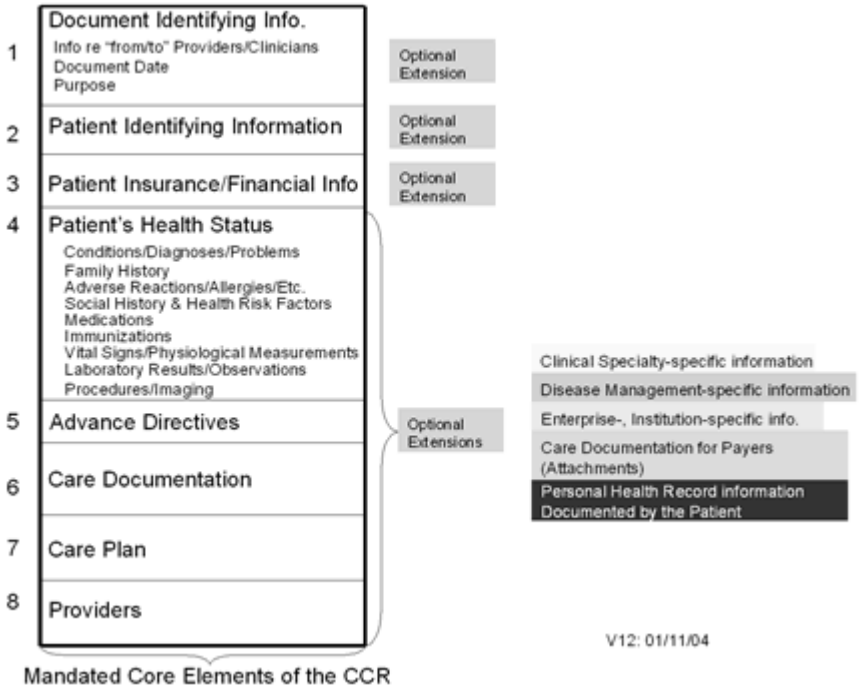


Figure 1- 3 Continuity of Care Records from www.medrecinst.com

- Personal Health Records, PHR Unlike the continuity of care record that mainly provides information to the physicians, the personal health record provides individual records primarily for the patient. Personal health information can be completely compiled into a single unit, such as a thumb disk, of a patient. Patient

information can be consistent on one source. It reduces or eliminates duplicate medical tests by different physicians and in case of an emergency. Patient plays a more active role on his/her medical records by having physicians add data to it. Some of personal health records allow the patients to update their information. Also, some of them include an Internet-accessible application that provides a form for the patient to simply print, add (but not medical records), or view medical records via secure connection to the Internet.

- Electronic Health Records, EHR The electronic health record is an interoperable, sophisticated and longitudinal record of patient health information including demography, progress notes, past medical history, vital signs, medications, immunizations, lab data and radiology reports. The major difference from other medical systems is that the electronic health record is not owned by any one physician, but rather combines the powerful database capabilities with the comfortable presentation layer or graphical user interface. The patient information can be added by primary care physicians, specialists, laboratories, radiology facilities, and insurance carriers. Because the data are moved to a central database, patient care activities and record access can occur simultaneously to multiple locations and is always up-to-date to all users.

Compared to electronic medical record, which places more emphasis on documentation of medical information, the electronic health record is an aggregate of all viewpoints related to the patient care. Other features that electronic health record supports are:

- Patient lookup and management of personalized patient lists
- Real-time information resource for both physicians and patients
- Problem list management

- Order entry for laboratory, radiology, medications

- Results notification and retrieval

- Patient Report retrieval

- Health maintenance and disease state management reminders

- Clinical encounter documentation

- Incorporating evidence-based clinical decision support

- Documentation of immunizations, patient education, health and reproductive factors, etc.

- Coding support (ICD and CPT)

- Consult/referral generation and tracking

- Billing

- Outcomes Reporting

- Public health disease surveillance and reporting

1.2. Distributed Databases

Distributed database is a database that is a part of large connection of several databases that have many locations within a network. Each database has its own CPU, memory, and storage device. Collection of data can be distributed to multiple locations. Each portion of data is called fragment or partition. The distributed databases synchronize all the data periodically, in case of concurrent access by multiple users, so that they are consistent on both updates and deletions.

The distributed database can use the architecture of client-server or collaborating servers. Using client-server architecture, a client sends a query to all database servers in

the distributed system. After each server returns the result, client caches and accumulates the response. For the collaborating architecture, a client sends a query to the nearest server, probably its server. Then the server executes the query locally and, at the same time, the server sends the query to other server if required. When all results from other servers send back to the server (the nearest server of the client who submitted the query), the server sends the response to the client.

1.2.1. Types of Distributed Databases

There are two types of distributed databases considering the software and hardware for the database system:

- **Homogenous Distributed Databases** Every site runs on the same Database Management System (DBMS). Because all nodes use the same database software, and likely hardware platform, the homogeneous system is easier to design and manage and is preferred by the database designer than the heterogeneous system.

- Heterogeneous Distributed Databases Different sites can run on different DBMS. Each site chooses their DBMS software and hardware on their own. Later the network connections are added, so the heterogeneity occurs. The designer must assemble different DBMS together on the distributed databases. (If data are not shared among the databases, it is not the distributed database.) The translations are needed in a heterogeneous system to allow the databases to communicate.

Users can access data in distributed database from either local application which do not require data from other sites, or global application which require data from other sites.

1.2.2. Transparency

Ideally the distribute databases must be transparent to users. Forms of transparency are as follows:

- **Data Distribution Transparency** The user should not know how data are fragmented into different databases. Data fragmentation in three approaches –vertical, horizontal, and mixed fragmentation- is explained in detail in chapter 4. If the database maintains duplicated copies, the replication transparency is desirable. So the user is unaware of the fact that multiple copies exist.

- **Location Transparency** Users should be unaware of the actual location of databases in the network. When they access a database, they must feel it as access into their local database.

- Database Management System Heterogeneity Transparency When users access the databases that are running on different DBMS, they should not be concerned about the differences between his/her local DBMS and the others that are connecting to the same network on the distributed database system. The system must provide the translation among different DBMS. For example, if a user logs into an Access DBMS and want to submit an Access query that will be sent to another DBMS, let us say Oracle, the system must provide transparency to the user. So the user does not have to translate to an Oracle SQL statement. If the query needs to access data from both DBMS, the system must support the distributed query. This is explained in chapter 4.

- **Transaction Transparency** Similar to a database system, a distributed database must support the transaction management. In this case, the system must guarantee concurrency transparency to ensure that two transactions employing databases in the

distributed databases will not interfere with one another. Not only “abort” and “commit” commands of the fundamental transaction statements in a database system, but also “prepare” and “end” of log statements must be handle by the coordinator and subordinates of the distributed databases. In case of a failure, the distributed database must also provide the recovery transparency. Two-phase commit protocol, consisting of a voting phase and a resolution phase, can be used to handle and failure.

1.2.3. Advantages and Disadvantages of Distributed Databases

Using distributed databases gives the DBMS many advantages.

- Capacity and Growth When the organization grows, new sites can be added with little or no effect on the DBMS. Systems can be modified from the distributed database without affecting other modules (systems). This differs from a centralized system, in which the growth may require a major change in hardware and software that affects the entire database.

- Reliability and Availability Failure of one database will affect only the site related to it, not entire system. The overall system remains available. With replicated data, the failure of one site still allows access to the replicated copy of the data from another site. The remaining sites continue to function. The greater accessibility improves the reliability of the system.

- Local Autonomy Data are fragmented according to the departments they relate to. A department can control the data about them.

- Sharing Users at site A can access data stored at other sites, B, C and etc, while the users at site A still preserve the control of their site A.

- Economics It costs less to create multiple small networks and DBMS than a single powerful large network and server.

In the opposite, the distributed databases also have some disadvantages.

- **Redundancy and Inconsistency** In case of poor synchronization, the redundancy of data storing in many databases can be inconsistency.

- **Security** Concerning of fragment security must be increased comparing to one centralized database. The infrastructure of multiple sites must be securely designed.

- **Complexity** Fragmentation of data to multiple sites increases extra works to the DB administrators to maintain the databases and the developer to perform operations across multiple systems.

However, with the proper design of Knowledge Database Management Unit (KDMU) of the Intelligent Network (IN)[1] that contains a series of front-end processors inside SMS (Service Management System) for entering customer data into actively duplicated databases in multiple synchronized locations inside SCP (Service Control Point), the disadvantages are well considered and mostly excluded from the using of the distributed databases.

1.2.4. Examples of Distributed Databases

There are many researches of distributed databases. The following shows two examples. One example, Mariposa, is the research on the distributed database. Another example, BioSimGrid project, is the research that uses distributed database as a tool for the biomolecular study in Chemistry.

---

[1] KMS and IN are explained in chapter 3.

Mariposa is an example of a distributed database management system that is an ongoing research at the University of California at Berkeley. Its approach reflects the emergence of the Internet and the problems of distributed databases in the Wide Area Network (WAN). This research of the distributed database management system emphasizes five principles:

- Scalability to a large number of cooperating sites. Their goal is to scale to 10,000 servers.

- Local autonomy. Each site controls over its resources including which objects to store and which queries to run.

-Data mobility. Data should remain available during movement.

- No global synchronization. Updated records and database schema changes will not force a site to synchronize with all other sites.

- Easily configurable for a local database administrator to change the behavior of a Mariposa[2] site under the purpose of low response time and high system throughput.

Another example of distributed database is the BioSimGrid[3] project using distributed database for biomolecular simulations. It is a collaboration of many researchers from several institutes, University of Oxford, University of Southampton, University of London, University of Bristol, University of Nottingham, and Biotechnology and Biological Sciences Research Council.

The purpose of the BioSimGrid project is to allow the results of large-scale computer simulations of biomolecules to be more accessible to the biological community. Because most simulation data reside in the home laboratory and are not accessible to

[2] For more information of Mariposa, see http://mariposa.cs.berkeley.edu/
[3] For more information of BioSimGrid project, seehttp://www.biosimgrid.org/

other research groups, BioSimGrid deposits all simulation results in a centralized database. They use the Grid to draw together distributed collections of simulation data in disparate formats, while maintaining a centralized accessible federated database.

1.3. Intelligent Network

An intelligent network is a telecommunication network that provides independent service such as free-phone 800 numbers, public emergency 911 numbers, virtual private networks, alternate billing service, and automatic calling cards. By replacing switches with computer nodes distributed throughout networks, the network operators can develop and control their service independently, and efficiently. Because the services are independent, a new service can be easily added, and customized to the need of customers.

1.3.1. Intelligent Network 1 (IN/1)

During the mid-1980's, the regional Bell operating companies (RBOC) proposed the architectures of an intelligent network (IN/1) that met three objectives, the rapid deployment of services in the network, the increasing opportunities for non–RBOCs to offer services, and vendor independence with standard interfaces. The first mark of IN/1 is to create and separate the service database, called service control points (SCPs), externally from the switching systems. Originally there were two services, 800 numbers, and calling card verification that required two separated service control points. To communicate with the service logic in the service control points, new software, that is a part of signal transfer point (STP) within the common channel signaling (CCS), was deployed to the switching system 7, SS7 network.

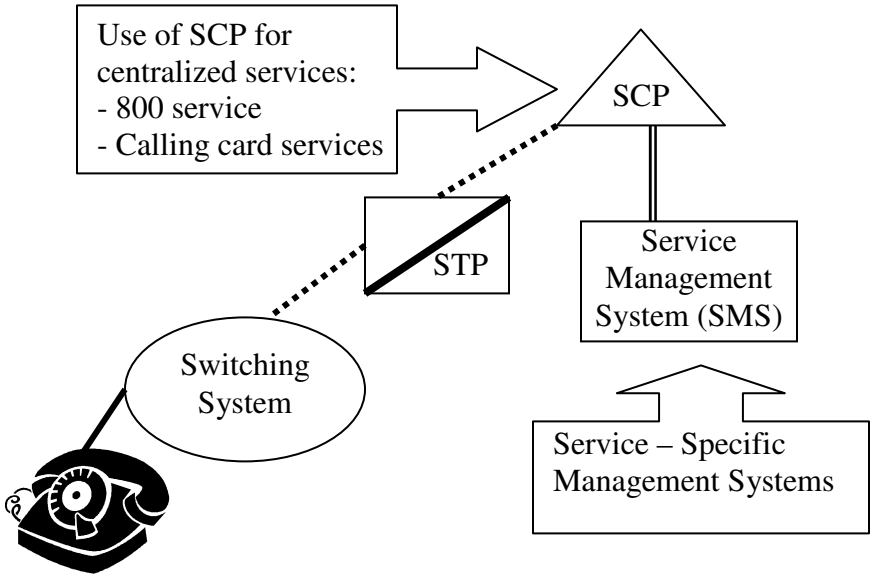


Figure 1- 4 Intelligent Network 1 (IN/1)

1.3.2. Advanced Intelligent Network (AIN)

Later, the Regional Bell Operating Companies (RBOCs) and the Bell Communications Research (Bellcore) proposed IN/1+ and IN/2. After that, Bell Communications Research developed Advanced Intelligent Network (AIN) that was recognized as an industry standard in North America and, later on, the International Telecommunications Union (see ITU-T), endorsing the concepts of AIN, developed an equivalent version of AIN called Capability Set 1 (CS-1). New elements, added to the IN/1 to become IN/1+, IN/2 and AIN are:

- **Intelligent Peripheral (IP)**, connecting to the SS7, for specialized telecommunication resources such as voice synthesis, and voice and data encryption,

- **Integrated Service Management System (ISMS)**, connecting to SCP via front-end protocol BX.25, for enhancing the generalized version with flexibility to a new network service without major software revisions.

- **Service Logic Interpreter (SLP)**, connecting to CCS7, for the conjunction with development of ISMS to provide the multiprocessor execution environment.

- **Network Information Database (NID) and Network Resource Manager (NRS)**, inside the SS7, to contain the customer-access and network-access information for the network mapping capability.

- **Service Circuit Node (SCN)**, connecting to the service switching point (SSP), to create new circuit-switched services addressing voice band information synthesis, interpretation, repetition, and translation.

- **Network Access Node (NAN)**, also connecting to the SSP, to detect trigger conditions whether to forward the request to another node that has the SS7 network access.

- **Operation Systems (OSs)**, distributed to the SCP, SSP, and IP, to supply the operation, administration, and maintenance functions.

The last three elements are the additions only to the AIN. Each of these elements is explained in detail in chapter 3. The following table shows the capabilities of AIN for each release.

Release	Capabilities
0	- Trigger checkpoints at off-hook, digit collection and analysis, and routing points of call - Code gapping to check for overload conditions at SCP - Seventy-five announcements at the switching system - Based on American National Standards Industry (ANSI) Transaction Capability Application Part (TCAP) issue 1
0.1	- Adds a formal call model that distinguishes the originating half of the call from the terminating half - Additional triggers - Two-hundred-fifty-four announcements at the switching system - Activate and deactivate subscribed triggers - Monitor resources - Based on ANSI TCAP issue 2
0.2	- Adds Phase 2 Personal Communication Service (PCS) support - Voice Activated Dialing (VAD) - ISDN-based SSP-IP interface - Busy and no-answer triggers - Next events list processing at SCP - Additional functions in all operations areas (e.g., network testing) - Default routing when calls encounter error conditions, they can be sent to a directory number, an announcement, etc.,
1	- Full set of above capabilities

Table 1- 1 Capabilities of each AIN release.

1.3.3. Wireless Intelligent Network (WIN)

The concept of the wireless intelligent network was developed by Telecommunications Industry Association (TIA) Standards Committee TR45.2. The idea is to apply the intelligent network (IN) capabilities into the wireless network without making the standard communication network obsolete. Wireless intelligent network brings in the successful strategy of IN into the wireless network for the expected service such as caller ID, voice/data messaging, personal communications services (PCS), Internet surfing, and location-sensitive billing services.

The elementary customer requirements of wireless services are

- **Roaming** Different networks can talk to another when customers are outside the provider's area. Wireless applications require additional SS7 messages of intelligent networking for various validations and billing reciprocation of wireless calls.

- **Carrier Select** Because there are many carriers and business partnership in different areas, both provider and subscriber can select the network to complete the call based on codes and automatic handset selection. The carriers can select only their business partners to control cost, if they know the rate of different networks. The carrier can select services that require the IN messaging.

- **Data-Service Capabilities** Short Message Service (SMS), like a pager but an additional service of telephone calling, requires many SS7 messages to set up the signaling, and get the data through the wireless network. It requires complex routing and authentication of IN to find the database, pull up the message, encapsulate it with the right header information, route it to the correct user, and then send it out like a phone call.

- **Voice Recognition Service** Instead of dialing a phone number, hands-free wireless services offers voice-activated dialing to convert voice to data. Applying the IN to this service, the network routes the call to intelligent peripheral (IP) devices that provide voice recognition, and voice-controlled services, translate the data, and dials. After the call is on, the IP device of IN application is removed from the telephone circuit whereas the IP device of the non-IN application is connected to the call the entire time.

- Fee Structure Billing of calls from different networks that have different negotiated fee structures can be easier by using IN flags to accumulate records of calling among networks even though the call was made to either wireless or wireline company.

1.3.4. Architecture of Wireless Intelligent Network

The architecture of WIN based on the IS-41 wireless network is the mirror of the architecture of the IN. Mobile Switching Center provides switching function as Service Switching Point (SSP) of IN. Major elements of IN such as service control point (SCP), intelligent peripheral (IP), and signal transfer point (STP) still exist in the WIN.

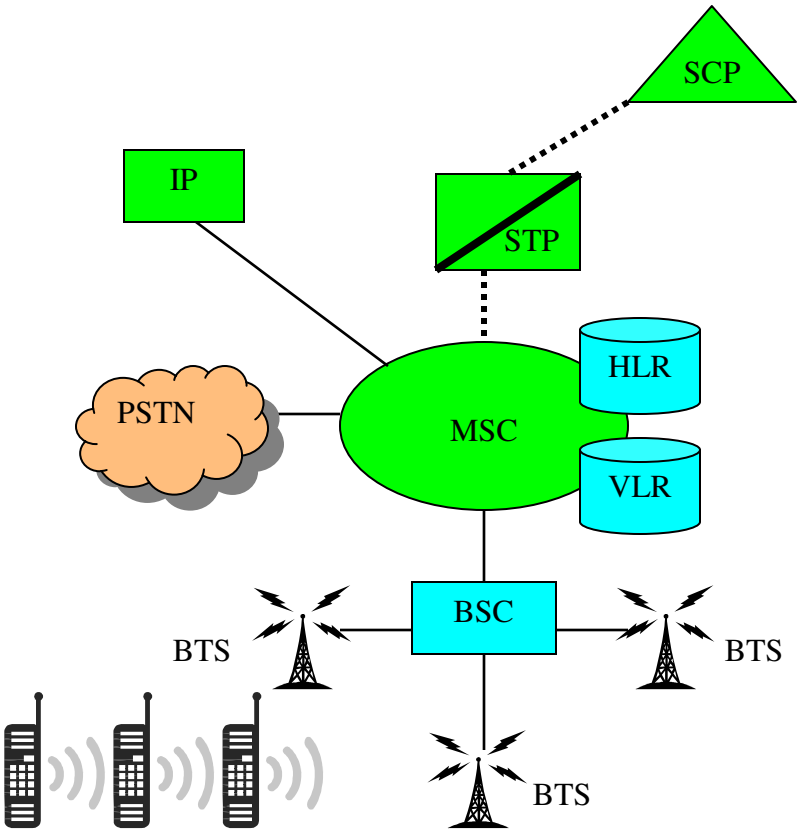


Figure 1- 5 Components of Wireless Intelligent Network

- **Public Switched Telephone Network (PSTN)** It is a global collection of interconnections to support traditional voice phone service that originally carries analog voice data but now almost entirely digital. Parts of the PSTN are also utilized for DSL, VoIP and other modern computer networking technologies.

- Location Registers They support Mobile Service Center, MSC, to maintain information about subscribers for record purposes. Because MSC cannot have a database of all potential users, location register is split into two elements:

- **Home Location Register** (HLR) It provides service control and mobility management for subscribers and mobile stations. The HLR is usually a network element such as an SCP.

- Visitor Location Register It is used by a MSC to retrieve information for handling calls from/to visiting mobile station user and to provide mobility management for subscribers in visited system.

Both HLR and VLR may serve more than one MSC. Because the subscribers and technology is growing fast, the HLR and VLR can be a stand-alone element off the MSC.

- **Base Transceiver Stations, BTS** The interface equipment is used to terminate radio path at the user side and to provide access services from the network to the user.

- Base Station Controller It is an intermediate element between BTS and the MSC. It is used as a way to segment the network and control congestion to ensure quality communications via traffic management, so it eliminates the case that every base station talks directly to the MSC.

1.4.Deductive Distributed System for Medical Systems

1.4.1. Deductive Distributed System

Computer systems have been used to manage medical information for decades. In 1981, the American Association for Medical Systems and Informatics (AAMSI) was incorporated. This organization was the result of the efforts of two predecessor organizations, the Society for Computer Medicine (SCM), incorporated in 1972, and the Society for Advanced Medical Systems (SAMS), incorporated in 1975, to merge into one. The main purpose is to support patient care, teaching, research, and health administration

through the development and implementation of computer systems. The health care systems have been deployed, developed, and improved from one generation to the next, from Document Management Systems to Electronic and Computerized Patient Records, EPR and CPR, to Automated and Computerized Medical Records, AMR and CMR, to Electronic Medical Records, EMR, to Continuity of Care Records, CCR, Personal Health Records, PHR, and Electronic Health Records, HER. Each of them has been applied with the new computer technology when they were introduced from the early computer system to the world wide connected computer on the Internet. Details are in section 1.1. Also the concept of Intelligent Network (IN) has been introduced to the medical system as an intelligent medical system by using the concepts of the major nodes of Intelligent Network such as SCP, STP, and IP to the health care utilities. Details are in chapter 3.

At the same time that medical societies have applied computer system to them, architecture of the computer system has been dramatically developed from giant-sized computer system to personal computer, wireline computer networking, laptop, and wireless digital-device networking including computer, cell phone, PDA, etc. Distributed systems were also introduced in the 1980's. Because of the popularization of the World Wide Web in 1993 into the Internet, the distributed system has been fully developed.

For example, the Sprite distributed operating system has been developed at University of California at Berkeley, and the Amoeba distributed operating system has been developed by Vrije University at Amsterdam. Sprite connects large numbers of personal workstations in a high-speed LAN. Each workstation has its own processor, whereas Amoeba uses processor pool which is a collection of processor, each having its own memory and Ethernet connection and is shared among users. The idle workstations

allow Sprite to use process migration to balance the workload of the system via the Remote Procedure Call, RPC, for the communication by using implicit acknowledgements to ensure the massage deliveries while Amoeba uses explicit acknowledgement to which a client has to an additional packet when the client receives the response from the server. The Sprite file system, which is the same form as a UNIX file system, caches files on both the server and client sides. The Amoeba file system has a large primary memory to store files that cannot be updated and cache only on the server side. To update the file, a new file is created from the old file, modified, and saved and the original file is deleted from the server. Amoeba server stores a file on contiguous block, but this is not necessary for the Sprite server. Therefore, the Amoeba server can transfer files faster than the Sprite server.

Java language has been used widely for the development of distributed system. Java servlets extend the functionality of web server by using special syntax of JavaServer Pages to allow programmers to create a web page to encapsulate Java functionality and write scriptlets. Servlets can be used for database applications that require minimal client support, called thin-client. The code of dynamic content generation can be written once and remain at the server side for client access.

Jini is an extended Java language framework for fault-tolerant distributed systems and services expansion from industry-based networks into home-based networks. The Jini uses lookup service to maintain information about the Jini services and enables clients to discover and use them. Client communicates with the service provider via Remote Method Invocation (RMI) which is a Java protocol to invoke a method of an

object on a remote computer to transmit objects between remote processes without explicitly program sockets.

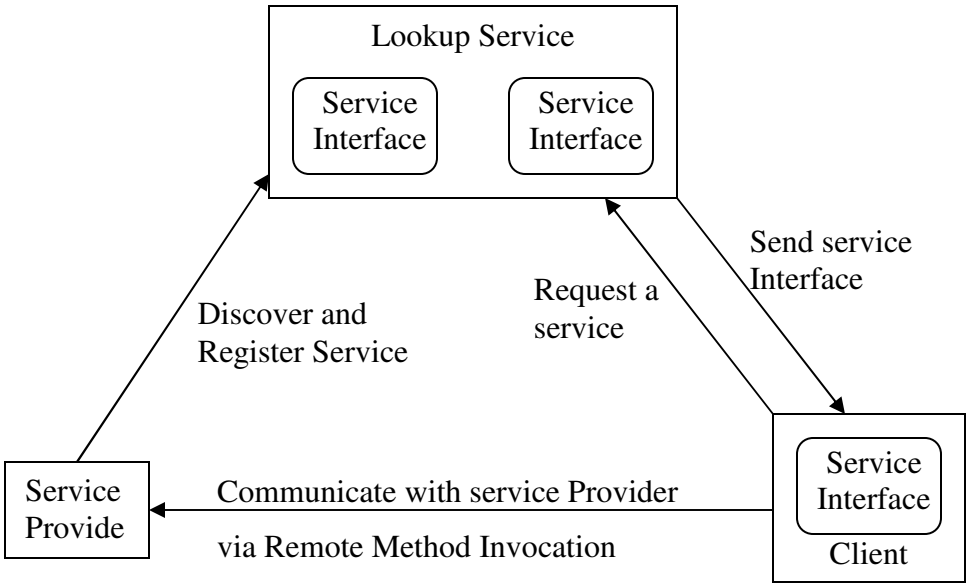


Figure 1- 6 Jini Architecture

The details of the distributed systems, and distributed databases are discussed in chapter 4 that includes designing, data fragmentation, data allocation and replication, and query processing.

1.4.2. Appling to Medical System

With the current technology of distributed systems being developed on the Internet, multiple medical agencies can be vigorously applied into them. Each medical agency has its own knowledge and information. This knowledge includes diseases, medication, health care, and on-going medical research. While information includes patient records of health, their medicines, test's results as in text, sound, or image, insurance, etc., and physicians' information such as their appointments, clinic hours,

specializations, hospital's information, etc. The information has been gathered, processed, and exchanged as business among the health service's provider of medical systems as mentioned earlier in section 1.1. But the concentration of knowledge is far less expanding comparing with the business issue of the patient/physician information. To collaborate the knowledge of several medical agencies, the following are required.

- Knowledge gathering. In multiple agencies, the knowledge system could be homogenous which is much easier to gather. However, it cannot be expected that all agencies will have the same system. Heterogeneous systems are more likely, but they are harder to implement. The concept of the distributed system must be applied to it. Not only is the difference of systems considered but also the distribution of knowledge in databases of different agencies. The data fragmentation in distributed databases can be reasserted. Details of distributed databases are located in chapter 4.

- Collecting Evaluation. After knowledge has been gathered, it must be processed. Since knowledge processing has been well adopted by researchers in the last few decades, our research concentrates on how knowledge is evaluated to fit into the need of users, for our case, for medical users, patients, physicians, medical researchers and computer scientists. Confidence level of the knowledge collected is one of the evaluation processes, much we can trust the result of knowledge processing and deduction. The details of confidence level of knowledge deduction are located in section 5.3.

1.4.3. Our Contribution

In our dissertation, we propose the concept of Medical Knowledge Technology (MKT). The Figure 1- 7 demonstrates the architecture of our Medical Knowledge Technology.

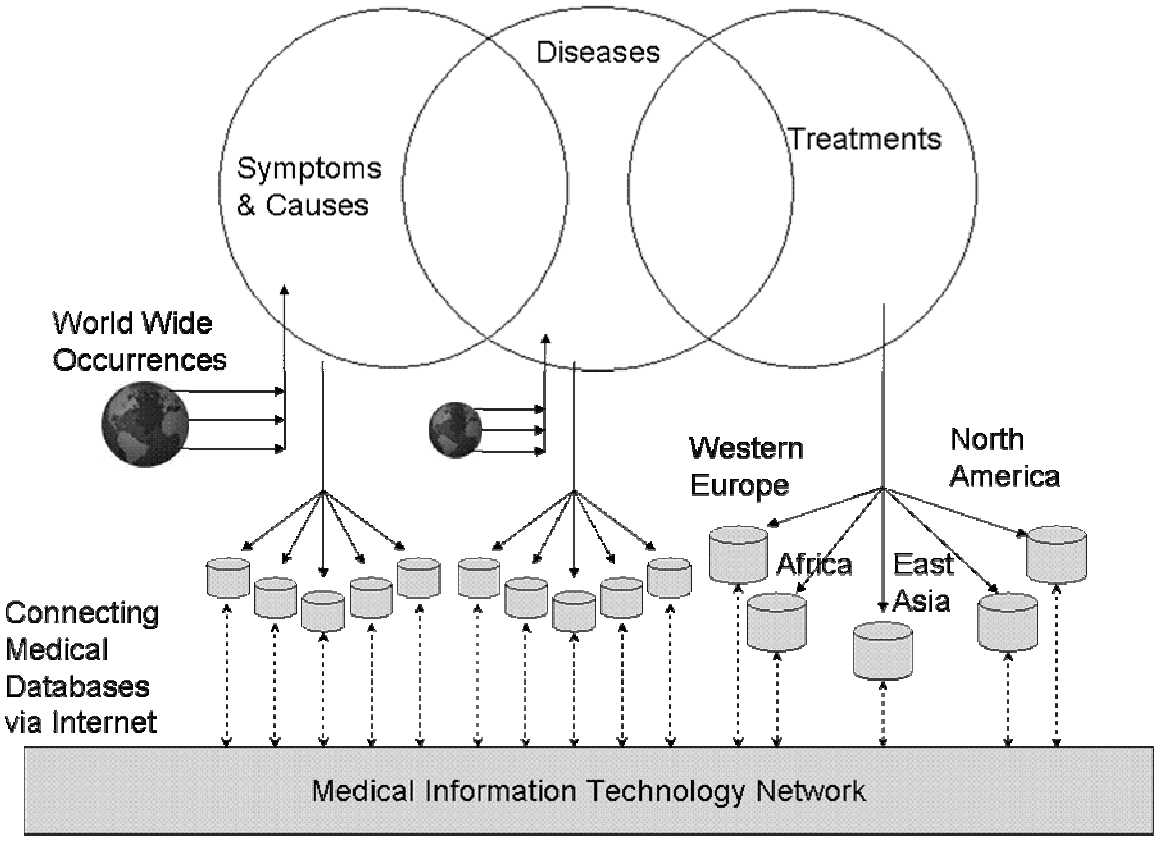


Figure 1- 7 Medical Knowledge Technology Architecture

As the world wide web becomes more and more available almost everywhere on earth, and as medical researchers make more discoveries, the web can be used to facilitate spreading news of medical advances quickly worldwide, wherever it is needed. Researchers discover the knowledge about illness, such as their symptoms and treatment, and record this knowledge in their medical databases. Once the knowledge accepted by respected medical organizations or by regulatory agencies such as FDA, the knowledge

can be published in hard-copy and sent to related physicians for medical practice. However, the questions are: "Who receive the publication?", "How many of them?", "Is it sent world wide?", "How much does it cost to send it?", "How can it be used?", etc. These questions can be answered by using the Medical Knowledge Technology (MKT) Network.

The Medical Knowledge Technology Network uses an existing internet to connect medical knowledge of researchers world-wide. The occurrences of symptoms and diseases in different locations may not be the same. Their discovery and understanding by each group of researchers are approved and stored in their databases and can be shared among medical researchers who connect to the MKT network via internet.

This shared knowledge can be retrieved to review, and enhance the knowledge of other medical researchers and physicians. However, that has already existed in webpages such as WebMD, and other medical internet services. We can provide not only a review of knowledge shared on the Internet, but also the level of confidence of results of the automated medical analysis.

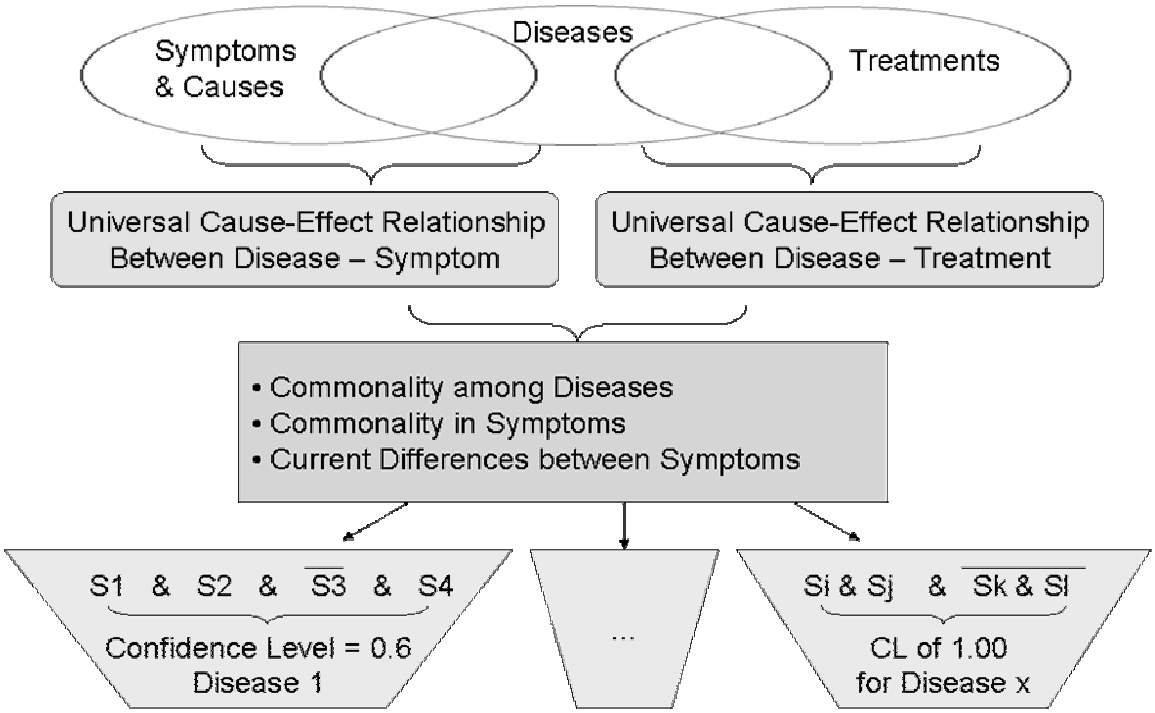


Figure 1- 8 Symptom Analysis in Medical Knowledge Technology

In Figure 1- 8, we briefly demonstrate the concept of symptom analysis. The medical knowledge supplied by researchers who connecting to MKT network is used to analyze symptoms of a patient; either by a doctor during visiting of the patient or by the patient him/herself. The knowledge of relationship between disease and its symptoms, and disease and its treatments are generalized to find the commonality of diseases, symptoms, and the differences among symptoms of a disease. Using this knowledge and symptoms Si, Sj, … and Sl of the patient, the symptom analysis provide a list of likely diseases that may occur in the patient. For each disease x, its confidence level is shown to the user of the MKT.

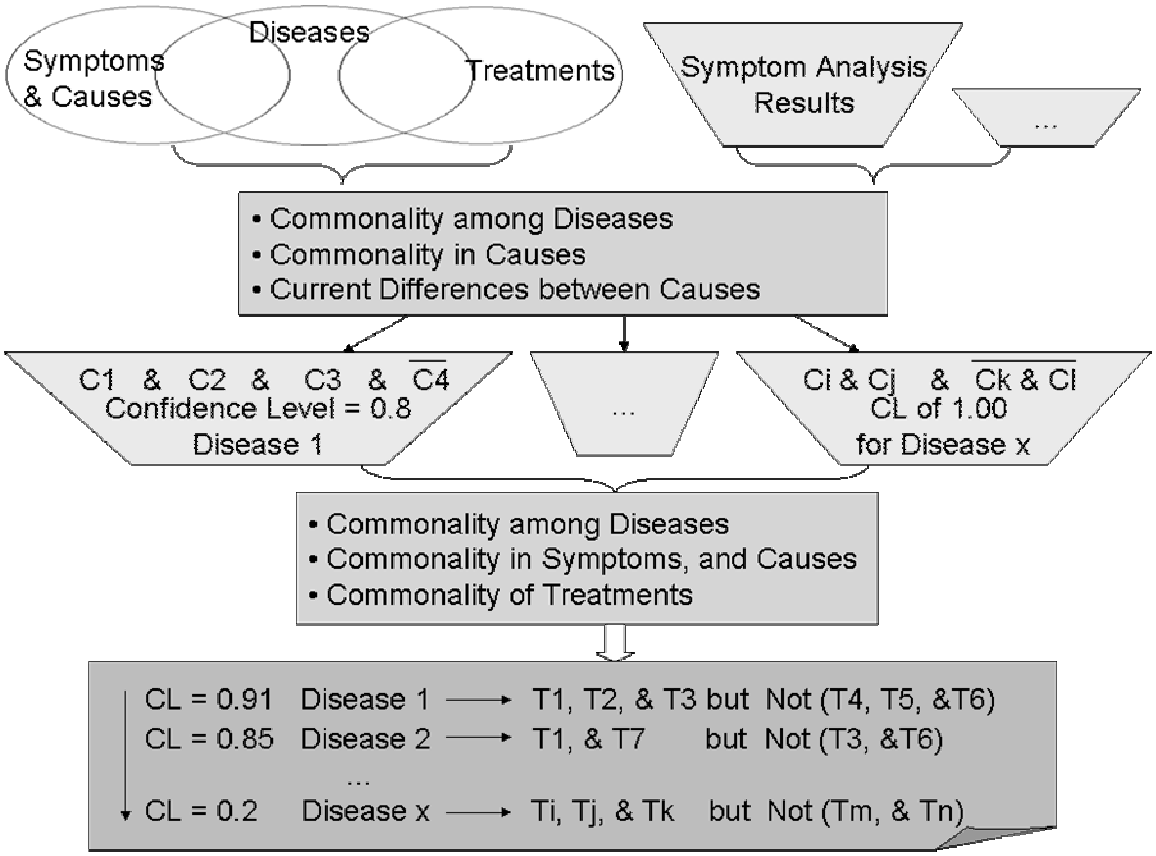


Figure 1- 9 Cause Analysis and Treatment Conclusion in Medical Knowledge Technology

Figure 1- 9 briefly demonstrates the cause analysis, and treatment conclusion given to the user of MKT. Similar to the symptom analysis, the cause analysis uses the commonality of diseases, and causes of illness, and the differences among causes to alter the confidence level of each disease that is likely to occur to the patient. The confidence level of a disease x may increase or decrease depending on the knowledge in the medical databases and the information given by the patient such as causes Ci and Cj occurring to the patient but not causes Ck and Cl.

Using the symptoms and causes provided by the patient and the knowledge in the medical databases, the medical analysis shows the list of likely diseases ordered by confidence level. For each disease, a list of treatments (methods of cure) is provided to

the user. According to the symptoms and causes of occurrence of a disease of the patient, not all treatments are suitable to the patient. For example, for disease x, the patient should receive the treatments Ti, Tj, and Tk but not Tm and Tn.

The detail of confidence levels are further discussed in chapter 5 whereas the medical analysis are explained in detail in chapter 6.

We ran tests on the Medical Knowledge Technology network for the medical analysis and their results are shown in the chapter 6.

To implement a system using distributed databases, the security issue is also considered along with the designing of the overall picture of the system. The secured network prevents an intruder to harm the system while the access codes prevent the patients' records which must be protected for their privacies. However the security concern is a major essential study of its own right. We will not consider it in our research.

Because the security and privacy are beyond our research, this study will not include patients' records which are also necessary and can improve the results of the confidence level study for future research.

Chapter 2 Relational Database Design

The amount of information has increased tremendously in recent years, and the number of data files has also increased. This has posed a challenge for developing systems with which data can be stored, searched and retrieved efficiently and at a reasonable cost. The relational databases have been introduced to handle such problems.

The relational database theories were studied early by mathematician. The term relation was taken from the definition in set theory.

$$R \subseteq A \times B$$

Relation R between A and B is a subset of $A \times B$.

$$A \times B = \{ (a, b) / a \in A \wedge b \in B \}$$

$A \times B$, a Cartesian product between two sets, A and B, consists of all ordered paired (a, b) having first component from a member of set A, and second component from a member of set B.

Why were databases implemented from set theory? A set is a collection objects as the database is a collection of data. All operations of set theory can apply to databases such as union, intersection, set difference, etc.

Later, more complicated operations are required to retrieve data from database. Mathematician introduced Relational Algebra that is the theory of SQL, modern database language. Other languages had also been researched such as Domain Relational Calculus, implemented in commercial language as QBE (Query-by-example) by IBM, and Tuple Relational Calculus, implemented as QUEL by Berkley University. However, the SQL was adopted by most of commercial databases.

2.1 Entity Relationship Model (ER)

Entity –Relationship Model (ER) is used for designing database by using a diagram to represent relation among objects in the system. It defines different objects by drawing different pictures.

2.1.1 Attributes

An object has its properties. Each property is called an attribute. For example, a patient has name, social security number, address, telephone number and etc. Each of these properties is called an attribute of a patient. Combining all these properties together for a patient is called an entity. For this case, a patient is an entity in a database.
We use ellipse to represent an attribute in the ER diagram.

Figure 2- 1 Ellipses Represent Name and Social Security Number Attributes of Patients

Some considerations about attributes are:

- Simple/Composite attributes

- Single/Multi valued attributes

- Derived Attributes

If an attribute can be categorized into two or more sub-values, we call this attribute a composite attribute. Otherwise, it is simple. Address can be categorized by street-address, street, city, state, and zip code. Address can be considered as a composite attribute.

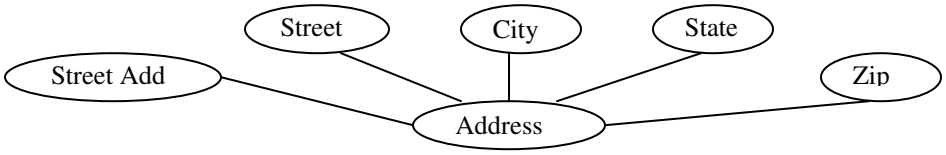


Figure 2- 2 Composite Attributes

Some entity may have an attribute that contains more than one value. If so, the attribute is called multi-valued attribute. A patient may have more than one telephone number such as home phone, cell phone, and work phone. Then we should consider the telephone number attribute is a multi-value attribute for an entity patient. However, a patient must not have more one social security number. The social security number of an entity patient must be a single, also simple, value. We use the double ellipse to represent a multi-valued attribute.

Figure 2- 3 Double Ellipse Represents Multi-valued Attribute

Considering an attribute to be multi-valued depends on how many different values it would be. If there is a limited number of a value, we should consider it as separated attributes. Telephone numbers of a patient can be many numbers but we know what kinds and the most numbers of phone numbers of a patient can have. For example, a patient may have two cell-phones, three home-phones, one fax, and two work-phones at most for each. If this is the case, we should model telephone numbers as eight different attributes.

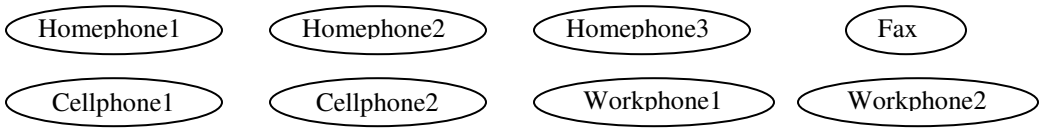


Figure 2- 4 Limited Number of Values of Multi-valued Attribute is Represented by Multiple Attributes

Creating table/columns of a multi-valued attribute leads to a lower performance of query execution. However, there is also a disadvantage of having many attributes instead of having multi-valued attributes. First, each database application has a limit number of columns for a table. Spurious numbers of columns may not reach the limitation. Later on in some case, the database administrator cannot expand the table if necessary. Second, having many columns may not apply to most tuples, a lot of columns will hold a NULL value that is wasting of space and poor performance. For example, most patients may not have, or submit to the system, all these eight values of phone numbers. Most of them may show only one home-phone and one work-phone. Then the rest six attributes will hold NULL values. If it is the case, we may combine all theses eight attributes to be one multi-valued attribute. Third, there is no distinction of usage among these different attributes. The home-phone and fax have a distinction of usage. We should separate them. Considering children of a patient, one patient may have none, one, two, …, or even fifteen children. There is no distinction of usage in the system if the system needs to know just names of them. Then the dependent (or children) must be model as a multi-valued attribute.

For simplicity, we shall use the telephone number as a multi-valued attribute throughout the dissertation.

There is a case that an attribute could be both composite and multi-valued attribute. For example, if the database designer wants to store all addresses of an entity patient and an address composes of many sub-values as mention earlier, then the address for this case is both multi-valued and composite attribute,

Some property of an entity may be able to acquire from other attributes (or relations.) If this is a case, then this attribute is called a derived attribute. For example, a patient entity can have an attribute called the number of phone numbers. The number of phone numbers can be counted (derived) from the attribute telephone number of each entity patient. We used dashed ellipse to represent a derived attribute.

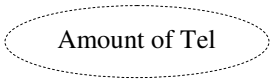


Figure 2- 5 Derived Attribute Represented by Dashed Ellipse

2.1.2 Entity Set

Each object in a database is an entity. But a group of entities, that share the same properties (attributes), can be listed together and is called an entity set. For example, patients named Mike and Susan both have their own social security number, address, and telephone number. Mike, Susan and other patients share the same attributes. So we put them together as an entity set named Patient. We use a rectangle to represent an entity set connecting to all of its attributes.

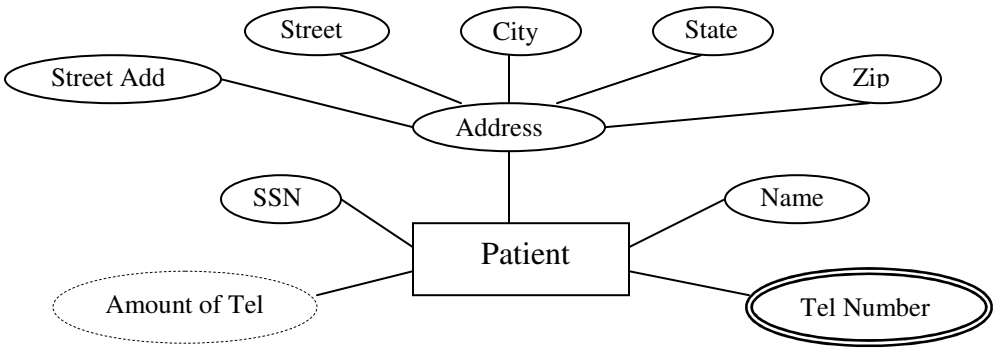


Figure 2- 6 Patient Entity Set

2.1.3 Relationship

Participation among entity sets is called a relationship. If a relationship is participated by two entity sets, it is called a binary relationship. Mostly, a relationship may be participated by two entity sets. However, there are cases of three or more entity sets to participate the relationship.

The mapping of a relationship between two entity sets can be one of three following cases:

One-to-one relationship

One-to-many relationship

Many-to-many relationship

- One-to-One Relationship.

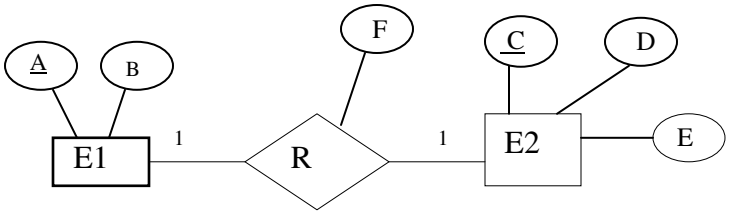


Figure 2- 7 One-to-One Relationship

This means that for each entity in the entity set E1 can participate in the relationship R to only one entity of the entity set E2 and for each entity of the entity set E2 can also participate in the relationship R to only one entity of the entity set E1.

A patient can have, at most, one insurance number. Each insurance number must belong to one patient. Insurance has its own attributes, coverage, name of company, and etc. This is a one-to-one relationship between patient and insurance.

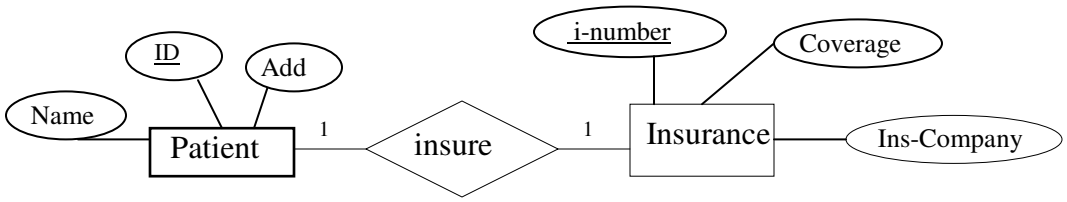


Figure 2- 8 One-to-One Relationship between Patient and Insurance

When a case of one-to-one relationship occurs, a database designer should reconsider whether it is really one-to-one relationship or two entity sets, participating in a one-to-one relationship, can be combined as one entity set, not two separated entity sets. For example, a patient entity set having attributes, id, name, and address, participates a one-to-one relationship with a blood entity set having attributes, patient id, blood type, and blood bank location.

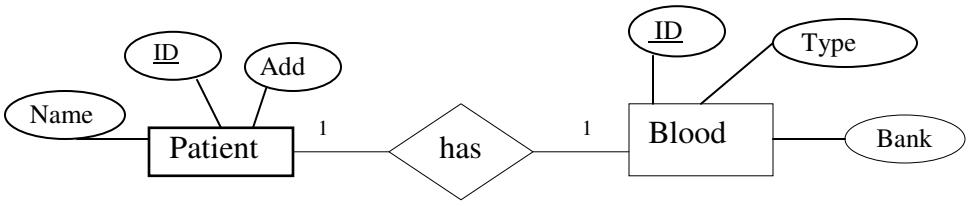


Figure 2- 9 One-to-One Relationship between Patient and Blood Type

In this case, if the blood entity set does not participate in any other entity sets, we can combine the two entity sets as one. So we can reduce a spurious table when transform

ER to tables. Too many unnecessary tables reduce overall performance of the database engine. The new patient entity set would be:

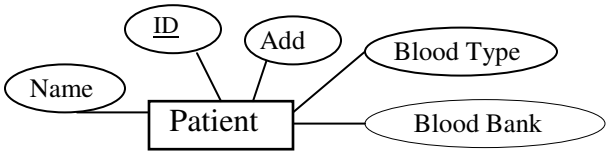


Figure 2- 10 Combining One-to-One Relationship into an Entity Set

But then as in the earlier example, should the patient entity set combine with the insurance entity set? Suppose there is an insurance that is for a group, not for individual. The insure relationship would be separated as two different relationships because each does not participate in the same pair of entity sets.

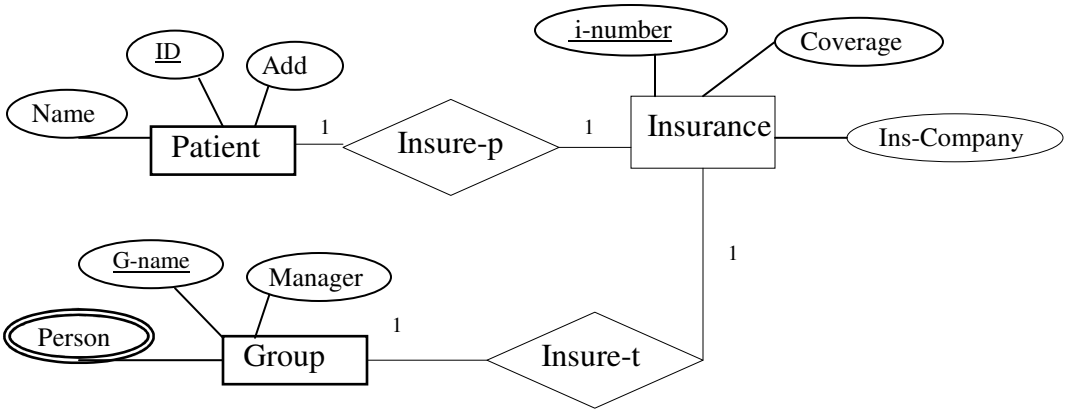


Figure 2- 11 Uncombined One-to-One Relationship

Therefore, we should not combine patient and insurance together as one entity set because some insurance may not insure a patient, and there may by a situation that one insurance number may use for a patient and a group that has the patient.

- **One-to-Many Relationship**.

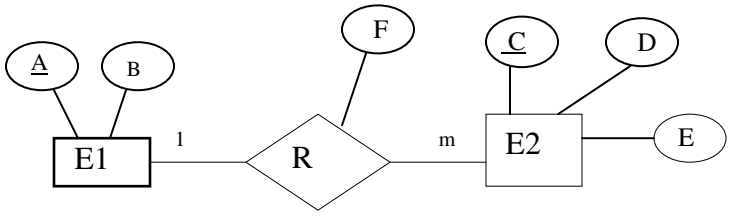


Figure 2- 12 One-to-Many Relationship

This means that each entity in the entity set E1 can participate in the relationship R to many entities of the entity set E2 but for each entity of the entity set E2 can participate in the relationship R to only one entity of the entity set E1.

For example, a patient has only one primary doctor. However, a doctor can be a primary doctor for many patients. (This is a primary doctor, not specialized doctor. A patient may consult many specialized doctors but only one primary doctor of the patient can sign a referral for the patient to meet a specialized doctor.

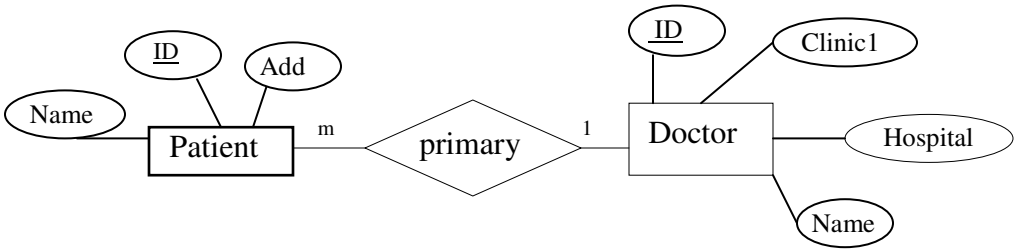


Figure 2- 13 One-to-Many Relationship between Patient and Doctor

In the entity-relationship design, there is no difference between one-to-many and many-to-one relationships. It depends on which way we look at the relationship. As in the picture above, we would say that it is many-to-one (m on the left and 1 on the right of the relationship primary). If we flip the diagram, we would have:

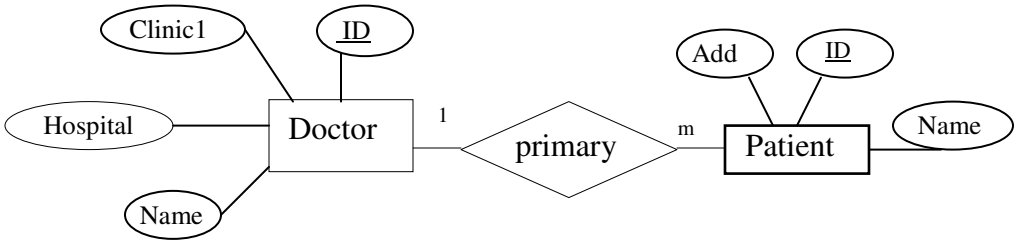


Figure 2- 14 Flipping Relationship of One-to-Many

This looks like a one-to-many relationship (1 on the left and many on the right.) However, the meaning of the design remains the same. That is why a database designer comments only one of them one-to-many.

- Many-to-Many Relationship.

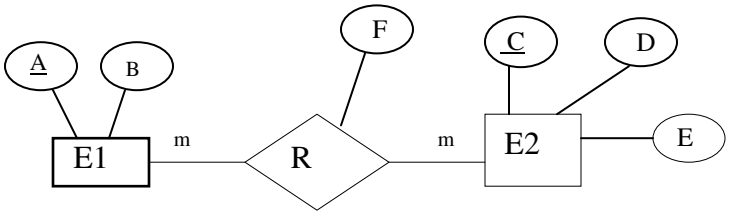


Figure 2- 15 Many-to-Many Relationship

This means that for each entity in the entity set E1 can participate in the relationship R to many entities of the entity set E2 and also for each entity of the entity set E2 can participate in the relationship R to many entities of the entity set E1.

Patient entity set has a one-to-many relationship with the doctor entity set because the system allows a patient to have only one doctor. However, a patient may consult many other specialized doctors. This shows us another relationship between the patient and doctor entity sets. Vice versa, a specialized doctor may see and advise many patients. Therefore, a new relationship, consult, is many-to-many between patient and doctor entity sets.

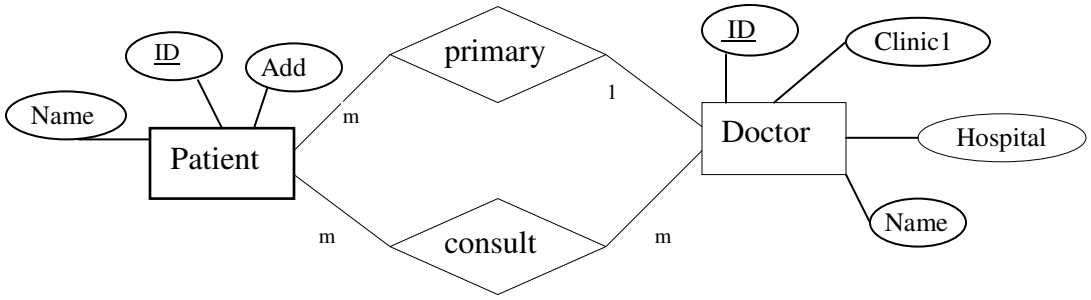


Figure 2- 16 Many-to-Many Relationship when Patients Consult Doctors but One-to-Many Relationship when Patients Visit Primary Doctor

Another example: there are many brand medicines, made by different drug companies. For example, headache of a fever can be cured by taking an aspirin. There are many brands of aspirin. An aspirin can also be used to reduce a pain, high temperature, etc. Therefore the many-to-many relationship is applied here.

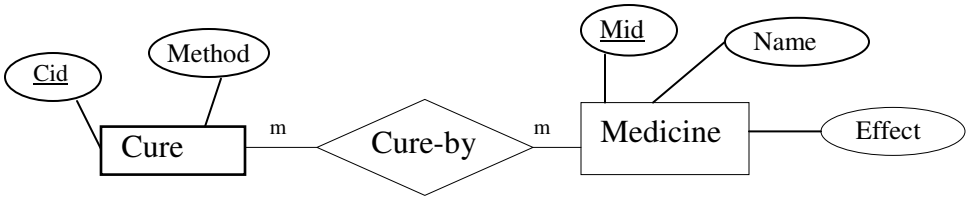


Figure 2- 17 Many-to-Many Relationship between Method of Cures and Medicines

There is another consideration of a relationship when an entity set participates in it. An entity set either may or must participate in a relationship. If it must participate, it is called total participation. Otherwise, it is partial participation.

For example, if a system requires that a patient must have a primary doctor, then the primary relationship from a patient entity set is total. We use a double line for a total participation. However, not all doctors must be a primary doctor of a patient. We should leave the participation between the doctor entity set and primary relationship as partial, a single line.

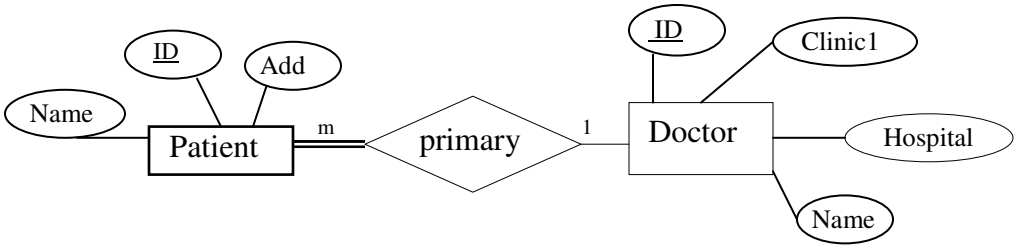


Figure 2- 18 Total Participation. A Patient must have a primary doctor.

This can be considered when we combine two entity sets that participate one-to-one relationship together. Usually we would see that at least one of them is a total participation. As our example, patient is total to has-blood relationship because every patient must have a blood type. The opposite is also true that each id of patient on the blood entity set must belong to a patient.

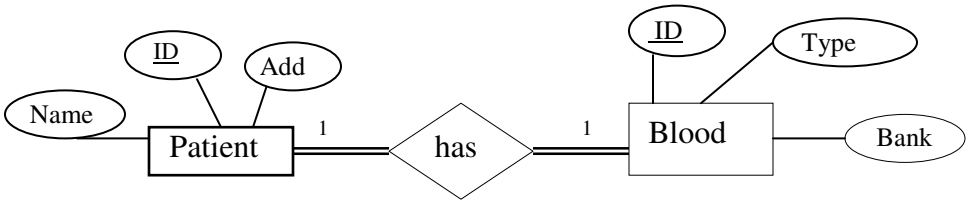


Figure 2- 19 Total Participation on Both Sides of Relationship

This would reassure to us that these two entity sets should be combined together as one entity set.

2.1.4 Weak Entity Set

There is a case that an entity set cannot form a primary key because its unique value depends on another entity set. For such a case, the entity set that cannot form the primary key is called a weak entity set. However, there must be a way to uniquely identify an entity from others. The weak entity set must have an attribute (or a set of

attributes) that can combine with a primary key of a strong entity set to which the weak entity set is depending. This attribute is called a partial key.

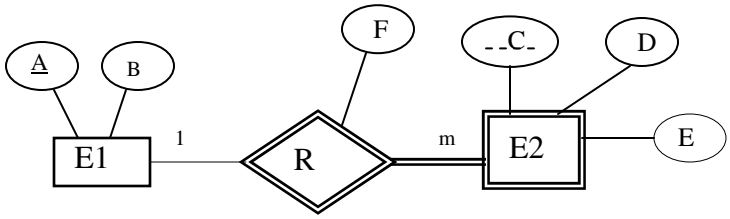


Figure 2- 20 Weak Entity Set

For the above example, the entity set E2 is a weak entity set that exists depending on the entity set E1. It means that every entity in the entity set E2 cannot exist without having an entity in the entity set E1 to depend on. The attribute C is a partial key that can uniquely identify each entity in the entity set E2 by combining it with the prime attribute A of the entity set E1. The relation R is showing that the entity set E2 is depending on the entity set E1. There is a case that a weak entity set participates in a relationship with other entity sets on which it does not depend. The following is the case.

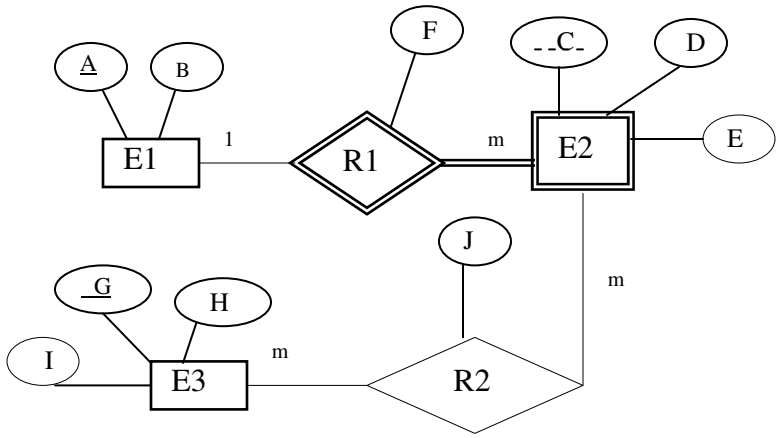


Figure 2- 21 Weak Entity Set E2 Existentially Depending on Entity Set E1, but not Entity Set E3

The relationship R1 is a relationship showing that the entity set E2 is depending on the entity set E1, not the entity set E3. The relationship R2 is just a regular

relationship to which the entity sets E2 and E3 participate together, but has nothing to do with existing dependency between them.

It is also possible that a weak entity set is depending on two or more entity sets.

An example of such a weak entity set in a medical system is when there is an analysis of a patient with his/her symptoms that occurs on each day. The analysis can be an entity set containing information such as the number of occurrences of a symptom. The date alone cannot be a primary key of the entity set analysis because, in one single day, many patients analyze their symptoms. Analyzing a single patient may discover many symptoms in a single day. This mean analysis is not only depending on the entity set patient but also the symptom entity set.

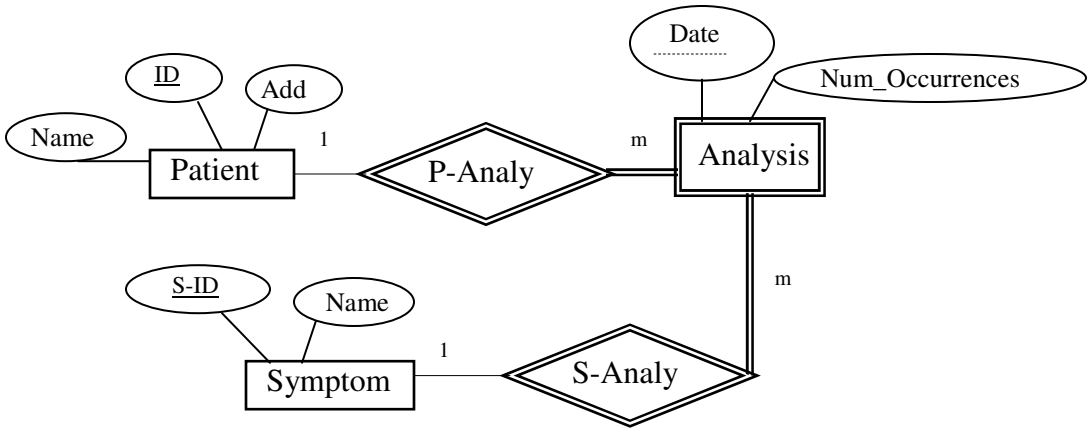


Figure 2- 22 Analysis Entity Set Existentially Depending on Both Patient and Symptom Entity Sets

The combination of patient, symptom, and date can uniquely identify the differences among entities in the entity set analysis.

2.2 ER to Tables

From the ER diagram, there are eight cases from the ER model to physical tables in a database. We shall consider not only what tables and columns will be created but also primary key, foreign key and other constraints.

2.2.1 Strong Entity Set (Regular Entity Set)

Each strong entity is a table. All attributes is a column except some special cases, explained later. Its primary key is defined as in the diagram.

For the following strong entity set patient,

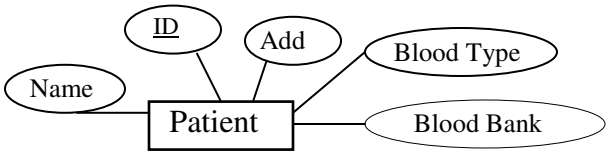


Figure 2- 23 Simple Patient Entity set

The entity set patient is a table with five attributes, ID, name, address, blood_type, and blood_bank. ID is a primary key to uniquely identify the difference among patients in the table. The data type of ID should be number. Data types of other columns are string for this patient entity set. Blood type may have a constraint to check only one of valid blood types, such as A+, A- B+, B-, AB+, etc.

For a composite attribute, each sub-value is a separated column if we will use them distinctively.

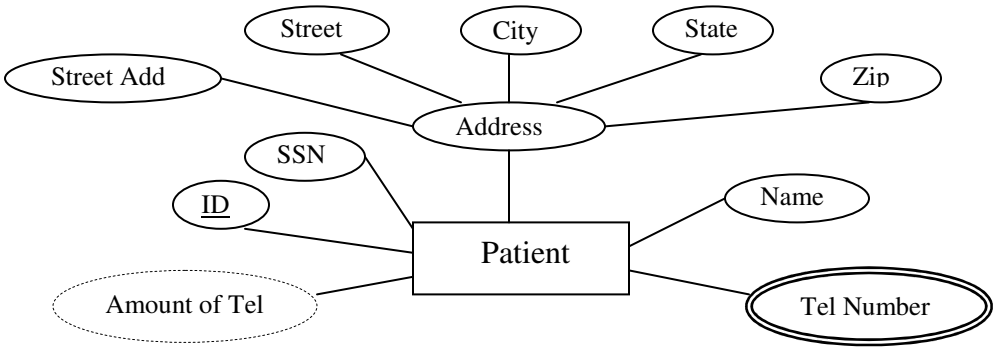


Figure 2- 24 Patient Entity Set with Composite Attribute Address, Multi-Valued Attribute Telephone Number and Derived Attribute Amount of Telephone Numbers

For the above entity set patient, address is a composite attribute. Instead of having one big column, address, of many sub-values that may bring down the performance when querying, the patient table contains these sub-values as columns, street_add, street, City, State, zip.

In case of derived attribute such as the attribute Amount-of-Telephones, this attribute is not a column of the table in the physical storage. Instead, we make a view for it because the value of a derived is varying depending on values of other columns or tables. The derived attribute, itself, does not have its own value. For the above example, the number of telephone number is a derived attribute. Its value is depending on the number of values of the attribute telephone-number. If a patient has more (or fewer) phone numbers, the value of this derived attribute will be increased (or decreased) by one. Therefore, we should make a query to calculate this derived value by counting the number of values in the telephone-number attribute. This query is a view to show values of a derived attribute.

Therefore, so far, the patient table composes of the following columns, ID, ssn, name, street_add, street, City, State, and zip but do not include Amount-of-Telephones

and telephone-number. Telephone number is a multi-valued attribute. We do not include it as a column in the table. We shall consider after this.

2.2.2 Multi-valued Attribute

Because each column in a row of a table can hold only one value, an extra table is needed to handle the multi-valued attribute. To join this new table back to the original table, the primary of the entity set must be included

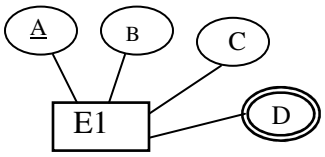


Figure 2- 25 Entity Set E1 with Multi-valued Attribute D

If the attribute D is a multi-valued attribute, the table E1 will has three columns, A, B, and C but not D. An extra table, let say E12, has columns A and D. Both columns A and D are combined as a primary key. A in an extra table has a foreign key referencing to the table E1 to make sure that the value of A in the new table cannot exist without having it in the table E.

Consider the strong entity set patient having a multi-valued attribute telephone number.

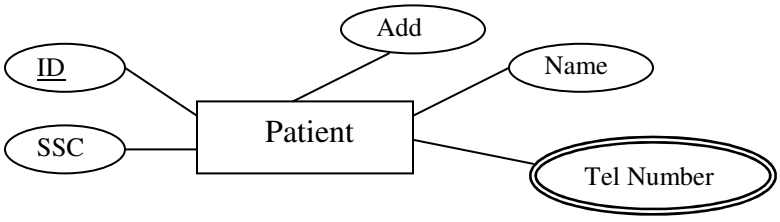


Figure 2- 26 Patient Entity Set with Multi-valued Attributed Telephone Number

The table patient does not include in the column telephone. The telephone attribute must be separated as a table. The table, let called patient-telephone, composes of two columns, ID from the entity set patient, and the telephone-number column. The primary key of this new table is a combination of ID and telephone-number columns together. One patient can have as many as phone numbers because each phone number of a patient will be in separated rows on a separate table, patient-telephone.

We have to make sure that patient in the new table must already exist in the original patient table. We can do so by adding a foreign key to the ID column of the new table to refer to ID in the patient table.

For the above diagram, we shall have the following

Patient (ID, name, address)

ID (of patient) is a primary key.

Patient-Telephone (ID, telephone-number)

The combination of ID and telephone-number together is a primary key.

ID has a foreign key referencing to ID in the patient table.

However some commercial databases try to support a multi-valued column. But there still be some limitations.

2.2.3 Weak Entity Set

A weak entity set is also a table with all of its attributes. However, because a weak entity set cannot form a primary key, a table for a weak entity set must includes the primary key of a strong entity set to which it is depending on. So the table can form a

primary key by combining its partial key with the primary key of its dependent strong entity set. It may be, possibly, depending on more than one table. The primary key must include all primary key of its dependent strong entity sets.

For example, E2 in the following picture is a weak entity set, existing depending on E1.

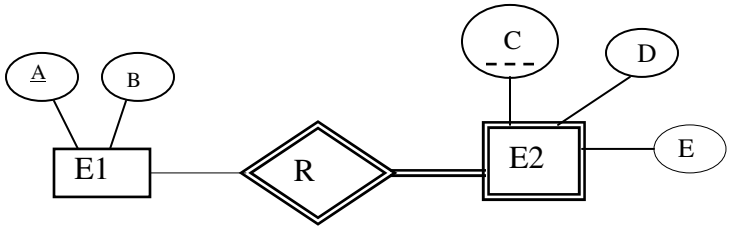


Figure 2- 27 Weak entity set E2 where C is its partial key.

A table for the weak entity set E2 will include all of its columns, C, D, E, and A, the primary key of E1 to which E2 exists depending on. The primary key of table E2 is the combination of A and C.

E2 (A, C, D, E)

Also the added column A in the table E2 must have a foreign key referencing to the column A in the table E1.

Consider our weak entity set analysis. For simplicity, we shall partially assume that the entity set analysis exists dependently only on the entity set patient. However, it is not completely true. The complete existing will be shown later.

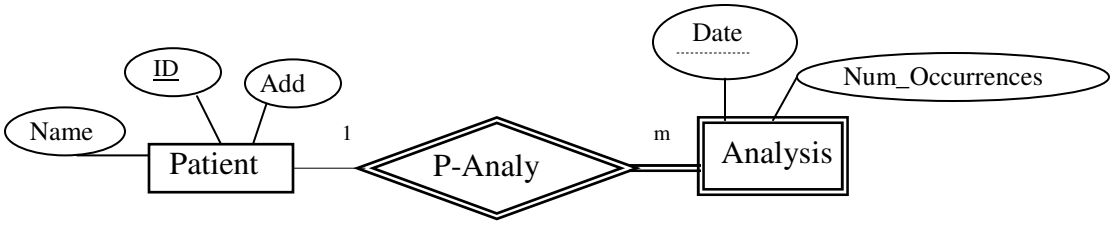


Figure 2- 28 Weak Entity Set Analysis Existentially Depending on Strong Entity Set Patient where Date is Its Partial Key.

For this case, the weak entity set analysis exists dependently on the entity set patient. The entity set analysis is a table composed of date and num_occurrences columns. Also the analysis table must include the primary key ID of the entity set patient on which the analysis is depending. Its primary key is a combination of both ID and date columns together. ID in the table analysis has a foreign key referencing to the column ID in the patient table. The foreign key assures that the analysis can occur only if the system has already had the information about the patient.

However the entity set analysis does not only exist dependently on the entity set patient but also the entity set symptom.

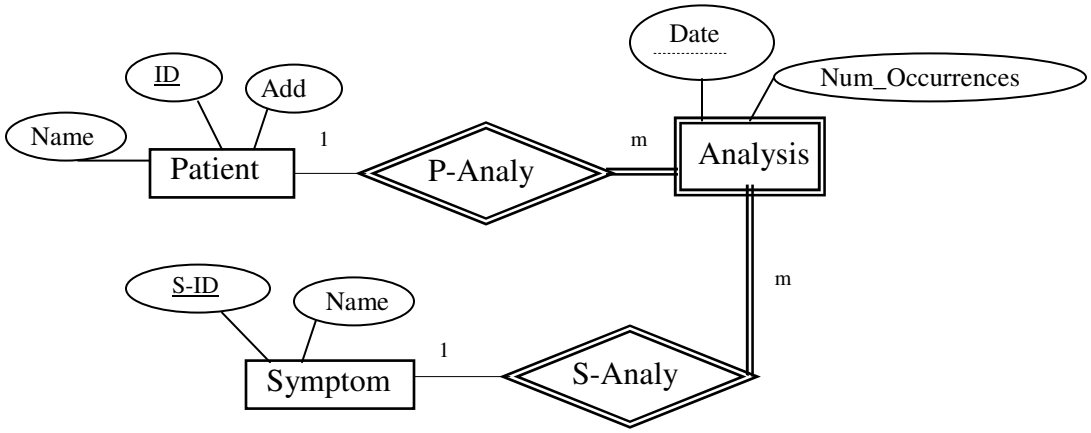


Figure 2- 29 Partial Key Date will be Combined with Patient ID and Symptom ID to Form Primary Key for Analysis Table.

Since the entity set analysis is existing dependently on the symptom, the primary key of the entity set symptom, S-ID, must also be added to the table analysis. Therefore, the table analysis composes of ID, date, S-ID and num-occurrences columns. Its primary key is a combination of three columns, ID, S-ID, and date together. We have two foreign keys in the table analysis. ID references to table patient and S-ID references to table symptom.

Semantically, each tuple in a table analysis must present ID of a patient, his/her symptom and the date of the occurrence of the symptom of this patient. This ensures that the same symptom may occur to the same patient many times, but on different dates. For the combination of this primary key, the analysis table will not accept a tuple of which the same symptom occurs twice to a patient on the same date. However, the num-occurrences can store a value as two or more if the same symptom occurs to a patient twice or more on the same date. For this case, the analysis of an illness considers only the number of symptom occurring for each day but not the time of an occurrence in each date.

Therefore the table for the above diagram are:

- Patient (ID, Name, Address)

  ID is a primary key.

- Symptom (S-ID, Name)

  S-ID is a primary key.

- Analysis (PID, S-ID, date, number-occurrences)

  Combination of PID (patient ID) and S-ID (symptom ID) together is a primary key.

  PID has a foreign key referencing to ID column in the patient table.

  S-ID has a foreign key referencing to S-ID column in the symptom table.

2.2.4 Many-to-Many Relationship

When a relationship involves n entities of an entity set with other m entities of another entity set, an extra table is necessary for joining these two entity sets together. Attributes of the table include primary key of the entity sets that participate to this m-to-

m relationship. If the relationship, itself, has one or more descriptive attributes, they will be included as attributes in the table. The primary key of the table is the combination of all primary keys of entity sets that participate in this relationship. Since the new table is a relationship from other entity sets, their primary keys of an entity in the new table cannot exist without having the entities in their original entity sets. Another word, the foreign key is need for both primary keys of the new table to reference back to their original table.

For example, the R relationship in following diagram is a m-to-m relationship

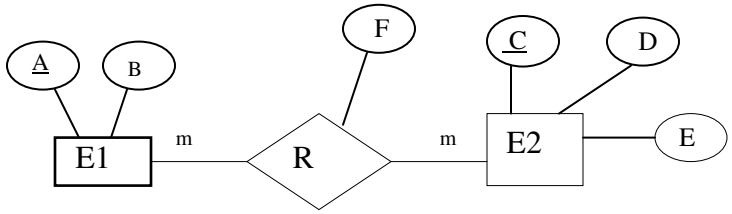


Figure 2- 30 Relationship R Needs Primary Keys A and C to Form Primary Key of Table R

Attributes of the table for the m-to-m relationship R are A, primary key of E1, C, primary key of E2, and F, its descriptive attribute. The primary key of the table is the combination of A and C together.

R (A, C, F)

The columns A and C in the table R have foreign keys referencing to the columns A and C in the table E1 and E2, respectively.

An example is the many-to-many relationship cure-by.

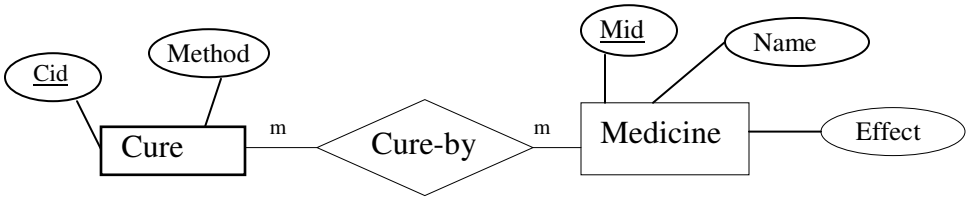


Figure 2- 31 Relationship Cure-by Needs Primary Keys Cure-ID and Medicine-ID to Form Primary Key for Table Cure-By

The relationship cure-by must be created as a table. The table is composed of all primary keys of all entity sets participating to the relationship cure-by. The cure-by composes of cid, primary key of the entity set cure, and mid, primary key of the entity set medicine. The combination of both columns cid and mid is the primary key of the table. Because of the primary key, one kind of cures can store, in the table, different kinds of medicines and one kind of medicines can store, in the table, different kinds of cures.

Therefore the table for the above diagram are:

- Cure (Cid, Method)

  Cid is a primary key.

- Medicine (Mid, Name, Effect)

  Mid is a primary key.

- Cure-by (Cid, Mid)

  Combination of Cid (cure ID) and Mid (medicine ID) is a primary key.

  Cid has a foreign key referencing to Cid column in the cure table.

  Mid has a foreign key referencing to Mid column in the medicine table.

If the cure-by relationship has a descriptive attribute, the attribute must be included in the cure-by table.

2.2.5 One-to-Many Relationship

An extra table is not needed as a one-to-m relationship. Consider the following diagram

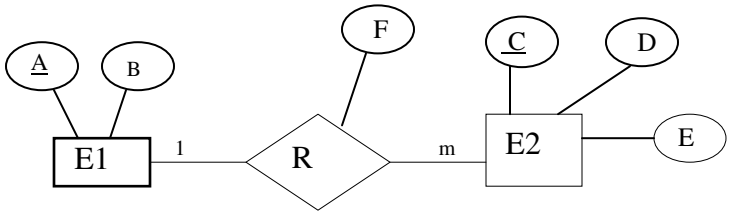


Figure 2- 32 One-to-many relationship R is not a table.

Because each entity of entity set E2 does not participate in more than one entity of entity set E1, there is only one record needed. Instead of creating an extra table for the relationship R, the table of entity set E2 will be added the primary key A of E1 as a column to participate the relationship R. The column A in the table E2 has a foreign key to reference to the column A in the table E1.

In conclusion, the primary key of entity set on one-side will be included as a column in the table of the entity set on many-side.

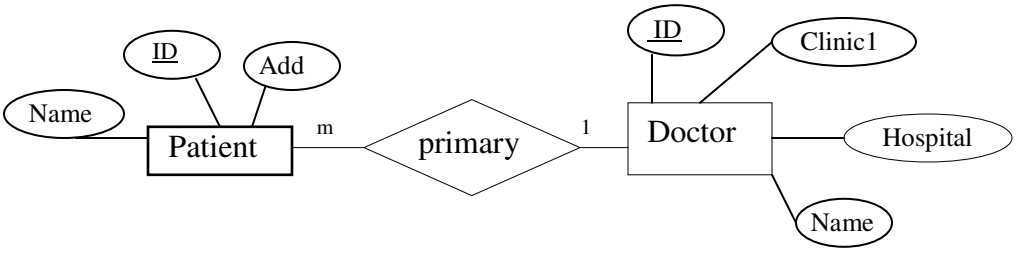


Figure 2- 33 One-to-many relationship Primary is not a table.

One doctor is a primary doctor of many patients but one patient can have only on primary doctor, one-to-many relationship. As described earlier, instead of making an extra table for the relationship primary, we add one more column, the primary key of doctor, ID, to the patient table. Since we already have column named ID in the patient

table, we cannot have another column with the same name, ID. Let us call this new column in the patient table primary-doctor-ID. Therefore, the patient table is composed of four columns, ID, name, add, and primary-doctor-ID. The primary-doctor-ID column has a foreign key referencing to the ID column in the doctor table.

If we consider the following part of diagram together, there are three, not four, tables to create.

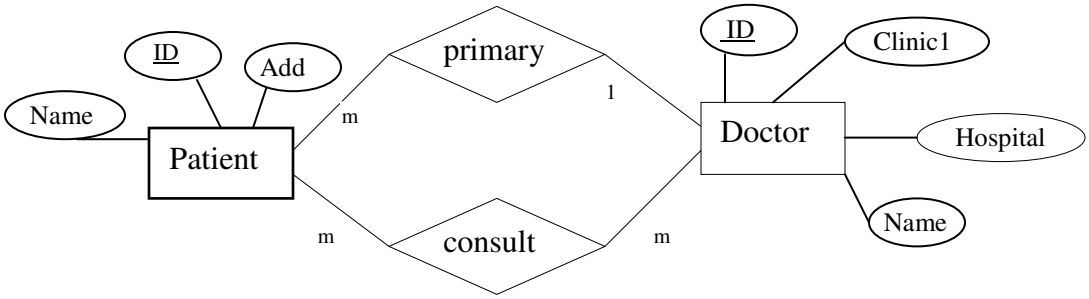


Figure 2- 34 Many-to-many relationship Consult forms a table, but not one-to-many relationship Primary

- Doctor (ID, Name, Hospital, Clinic1)

  ID (of doctor) is a primary key.

- Patient (ID, Name, Address, primary-doctor-ID)

  ID (of patient) is a primary key.

  Primary-doctor-ID has a foreign key referencing to ID column in the doctor table.

- Consult (PID, DID)

  Combination of PID (patient ID) and DID (doctor ID) together is a primary key.

  PID has a foreign key referencing to ID column in the patient table.

  DID has a foreign key referencing to ID column in the doctor table.

2.2.6 One-to-One Relationship

The idea is similar to one-to-many relationship. Since both entity sets participate relationship only one to another, we can add a primary key of either entity set to another. However, we should consider which entity set participates by more amounts of entities.

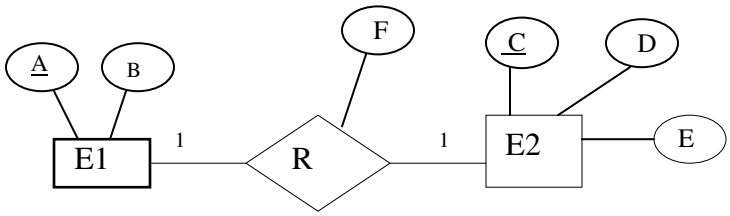


Figure 2- 35 Either primary key A or C must be added to the table on another side.

If the number of entities in E1 participates in relationship R more than the number of entities in E2, the primary key A of entity set E1 should be add as a column in table E2. So the table E2 would have less null value in column A comparing if we the primary key C of entity set E2 to the table E1. Following the addition, the column A in the table E2 must have a foreign key referencing back to the column A in the table E1.

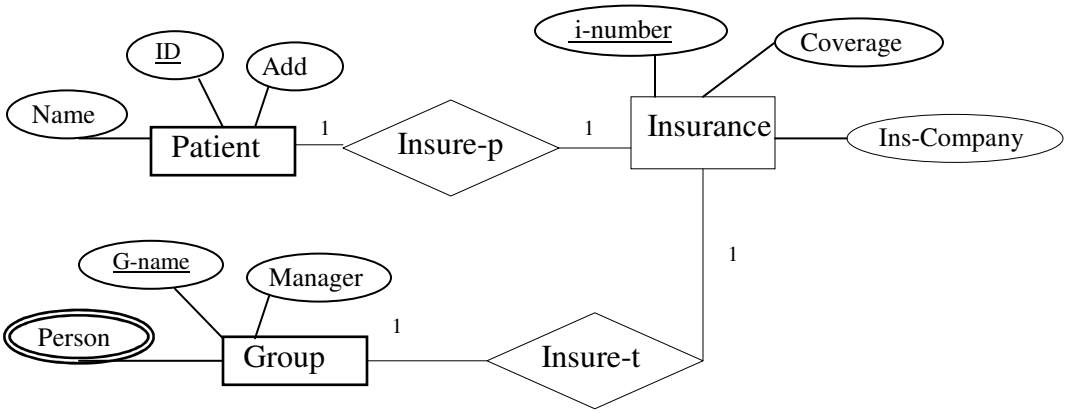


Figure 2- 36 Insurance number attribute is added to both tables Group and Patient.

For the above case, an insurance number must belong to a patient or a group. We can add two columns on insurance table. One is patient-ID and another is G-name. However, it could be either one. We have to scarify one NULL column for every row.

We should do the opposite. Add the insurance number to both tables, patient and group. Even though not all patients have an insurance, most do. We would have less NULL in patient and group tables comparing to in the insurance table. The columns insurance number in both tables, patient and group must reference to the column insurance number in table insurance.

Therefore, the tables for the above diagram are:

- Insurance (i-number, Coverage, Ins-company)

  I-number is a primary key.

- Patient (ID, Name, Address, i-number)

  ID (of patient) is a primary key.

  i-number has a foreign key referencing to i-number column in the insurance table.

- Group (G-name, Manager, i-number)

  G-name is a primary key

  i-number has a foreign key referencing to i-number column in the insurance table.

- Group-person (G-name, person-name)

  Combination of G-name and person-name is a primary key together.

There are other features in designing databases using Entity-Relationship diagram such as specialization, and generalization. I will omit them because they are not used in this dissertation.

2.3 Normalization

Another way of designing a database, instead of using entity-relationship diagram is the normalization using functional dependency. It is a formal method that a database designer must follow steps of normal forms.

2.3.1 Functional Dependency

Functional dependency is a statement of determining attributes by attributes.

Definition: A functional dependency $A \rightarrow B$, reads A determines B, specifies that

If $t1[A] = t2[A]$ then $t1[B] = t2[B]$

It reads that if there are two tuples t1, and t2 in a relation r having the same value in column A, then those tuples t1 and t2 must have the same value in the column B. It implies that there is no possibility that one value in column A will match to more than one value in column B. Another word, if we know the value of column A, then we shall know the value of column B.

2.3.2 Functional Dependency Axioms

The following rules are three well known axioms in the study of normalization. A, B and C are sets of attributes.

1) Reflexive: If $A \subseteq B$ then $B \rightarrow A$.

Suppose $t1[B] = t2[B]$. Then $t1[A] = t2[A]$ because A is a part of B. Therefore $B \rightarrow A$

2) Augmentation: If $A \rightarrow B$ then $AC \rightarrow BC$.

Proof by contradiction can be used. Assuming $A \rightarrow B$ but it is not true that $AC \rightarrow BC$.

That is $t1[A] = t2[A]$ 2.1

	t1[B] = t2[B]	2.2
	t1[AC] = t2[AC]	2.3
but	t1[BC] ≠ t2[BC]	2.4

Since t1[B] = t2[B] but t1[BC] ≠ t2[BC] then t1[C] ≠ t2[C].

Since t1[A] = t2[A] and t1[AC] = t2[AC] then t1[C] = t2[C]. That is the contradiction.

3) Transitivity: If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$.

$A \rightarrow B$ means if t1[A] = t2[A] then t1[B] = t2[B]

$B \rightarrow C$ means if t1[B] = t2[B] then t1[C] = t2[C]

Therefore by the chain rule, t1[A] = t2[A] then t1[C] = t2[C]. That is $A \rightarrow C$.

There are three other well known rules of the functional dependency theory.

4) Decomposition: If $A \rightarrow BC$ then $A \rightarrow B$ and $A \rightarrow C$

5) Union: If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow BC$

6) Pseudo-transitivity: If $A \rightarrow B$ and $BC \rightarrow D$ then $AC \rightarrow D$

We can proof the last three rules by using the above axioms.

Proof Decomposition

Given	$A \rightarrow BC$	4.1
Reflexive: since $C \subseteq BC$,	$BC \rightarrow C$	4.2
Transitivity 4.1, 4.2	$A \rightarrow C$	4.3

Proof Union

Given	$A \rightarrow B$	5.1
Given	$A \rightarrow C$	5.2
Augmentation 5.1 by A	$A \rightarrow AB$	5.3

AA is A		
Augmentation 5.2 by B	AB → BC	5.4
Transitivity 5.3, 5.4	A → BC	5.5

Proof Pseudo-transitivity

Given	A → B	6.1
Given	BC → D	6.2
Augmentation 6.1 by C	AC → BC	6.3
Transitivity 6.3, 6.2	AC → D	6.4

2.3.3 Normal Forms

There are five normal forms (technically six normal forms but Boyce-Codd is the strictly third normal form) that a database designer must follow. However, in practical industry, only up to 3rd, or Boyce-Codd normal forms are using because it is almost impossible to find all constraints by the time of designing of a new system. However 4th and 5th normal forms should be described here briefly.

- **First Normal Form** is to dealing with a multi-valued attribute. It states that there is only atomic attribute allowed on a relation that is in first normal form.

If a relation is not in the first normal form, we have to decompose it by separate the multi-valued attribute into a new relation.

For example, if we have relation R (A, B C, D) with functional dependency A → BCD and D is a multi-valued attribute. Decomposed relations will be R1(A, B, C) and R2(A, D). The primary key of R2 relation is attributes A and D together.

Second Normal Form is to assure that all non-prime attributes of a relation in the second normal form must be fully dependent on the primary key of the relation. Another word the second normal form does not allowed an attribute to be determined by a partial of the primary key. So the primary key of a relation can determine any attribute of its relation.

If a relation is not in the second normal form, we must decompose the attributes that against the second normal form into a new relation considering with their functional dependency.

For example, suppose there is a relation R(A, B, C, D, E, F), whose primary key is attributes A and B together and two functional dependencies A → CD and B → EF. This relation is not in the second normal form because CD and EF are not fully dependent on the primary key AB. Therefore, the relation R must be decomposed following their functional dependency. Two new relations are R1(A, C, D) and R2(B, E, F). Now both R1, R2 are in the second normal form. Another relation R3(A, B) must also be created, for this case, to prevent the lostless-join. The primary key of relations R1, and R2 are A and B respectively. The primary key of R3 is the combination of attributes A and B together.

Third Normal Form can reduce the transitivity. It states that all non-prime attributes must not be transitively determined by the primary key of its relation. That is, no transitivity rule can be applied from the primary key to a nonprime attribute.

When a relation is not in the third normal form, we need to decompose the relation to withhold the transitivity inside the relation.

For example, if we have a relation R(A, B, C, D), whose primary key is the attribute A, with two functional dependency $A \rightarrow BCD$ and $C \rightarrow D$. This relation R is already in the second normal form because all nonprime attribute B, C, and D are determined by the primary key A. However, it is not in the third normal form. Using the decomposition rule, it infers $A \rightarrow C$. By transitivity, we would have $A \rightarrow D$. Therefore the relation R must be decomposed into two relations R1(A, B, C) and R2(C, D). The primary key of R1 is the same as of the relation R. The primary key of R2 is C.

Boyce-Codd Normal Form, as mentioned, is a strictly third normal form. It is not the only non-prime attribute that does not allow the transitivity but also the attribute(s) of the primary key itself. There are cases that a primary key composes of more than one attribute and a non-prime attribute, that originally determine by the primary key, determines a partial of primary key.

To decompose a relation that is not in the Boyce-Codd normal form, is similar to the decomposition of a relation that is not in the third normal form. Some functional dependency may not been seen as in one relation, but still hold if we join back the relations.

For example, suppose there is a relation R(A, B, C, D), whose primary key is the attributes A and B together, with functional dependencies AB → CD and C → B. This relation R is in third normal form because no nonprime attribute is transitively determined by the primary key. However, a prime attribute B is transitively determined by the primary key, using decomposition rule, AB → C and transitivity between AB → C and C → B, AB → B. Decomposition is the same way as decomposing a relation into the third normal form. The relation R must be decomposed by the functional dependency C → B. Two new relations are R1(A, C, D) and R2(B, C). The primary key of R2 is B obviously.

We have AB → CD but B is not in the new table. What would be a primary key of R1 since there is no functional dependency among A, C, and D? However, we use the functional dependency rule to show the primary key of the R1.

Given	AB → CD	(1)
Given	C → B	(2)
Decompose (1)	AB → D	(3)
Augmentation (2) by A	AC → AB	(4)
Transitivity (4), (3)	AC → D	(5)

Therefore the primary key of the new relation R1(A, C, D) is the combination of attributes A and C together.

Forth Normal Form is assure that there is no multi-valued dependency X →→ Y when X is the super-key of a relation R who is the forth normal form.

Definition: When X, Y are sets of attributes on R and Z is R – (X ∪ Y), the multi-valued dependency X $\rightarrow\rightarrow$ Y on a relation R is specifies that

If t1[X] = t2[X] then 1) t1[X] = t2[X] = t3[X] = t4[X]

2) t1[Y] = t3[Y] and t2[Y] = t4[Y] and

3) t1[Z] = t4[Z] and t2[Z] = t3[Z]

For example, suppose we have a relation R(A, B, C) with multi-valued dependency A $\rightarrow\rightarrow$ B and the primary key is all attributes A, B, and C together. Then we have to decompose this relation R to be relations R1(A, B) and R2(A, C). The primary keys of both relations R1 and R2 are the combination of all of their attributes. Attributes A and B together are for the relation R1. Attributes A and C together are for the relation R2.

Fifth Normal Form is a project-join normal form. It is to consider with functional, multi-valued, and join dependency.

For example, if a relation R(A, B, C) is not in the fifth normal form. The relation R must be decomposed into three relations considering all combinations: R1(A, B), R2(A, C) and R3(B, C). The primary key is the combination of both attributes of the relation.

2.4 Query Processing

One of the main performance of a database management system is the query processing. It is the main reason of choosing a database package that suits for the requirement of a company. Most database packages accept a query in SQL. But only some database packages allow a complex query such as nested sub-query. SQL-Server, Oracle, and some others allow a SELECT statement nested inside a main SELECT statement. However MySQL, that is a freeware for academic, cannot process a nested sub-query. Its query engine was not built for a complicated query because most queries have alternatives of writing them in SQL to solve the same query. Also a programmer, at some point, must build an application to submit a query, let the database server process, receive the result back to application, re-process the result of query, and re-submit a new query for a finer result. That is, more work has to be done from the programmer point of view. Therefore, query engine should be able to process, optimize and execute a complex query.

The execution of a query includes the scanning, parsing, validating, building query tree, optimizing, and then retrieving from files.

Scanning The database engine identifies all words in the query. These words include SQL keywords, names of relation, and name of attributes. There may be some other words in a query such as variables, not known in advance.

Parsing It is the same as parser of a language. It checks the syntax of the given query before executing it. The query must be composed according to the grammar of the language (mainly in SQL syntax except some applications that has an easy-to-use query builder for users.)

Validating After the syntax is checked, each scanned word that is not a keyword of SQL language, will be checked with the database schema for the name of attributes or name of relations. Considering the syntax by parser, each word must be meaningful for the query.

Building query tree The representation of a query usually is done as a tree, or graph data structure.

Optimizing There are many different strategies to execute a query. The database engine must choose the most suitable strategy to retrieve data from database files. Some strategies take more time and resources than others. The main job of the database is to store and retrieve data for the user. So the query optimizer must be intelligently implemented to maintain good performance for user requests.

Retrieving After the best strategy is chosen by the query optimizer, the database engine will read/write data from/to the database files. Index files may be used. Then the result of the query will be passed back to user.

There are two main techniques to implement the query optimizer. The heuristic rules and the system estimation. Practically, both techniques are used together in a query optimizer. Before we explore these two techniques, we should understand how a query is processed from the start. The database engine starts processing a query from translating SQL query into relational algebra, and then searching method for selection, join, and projection operation.

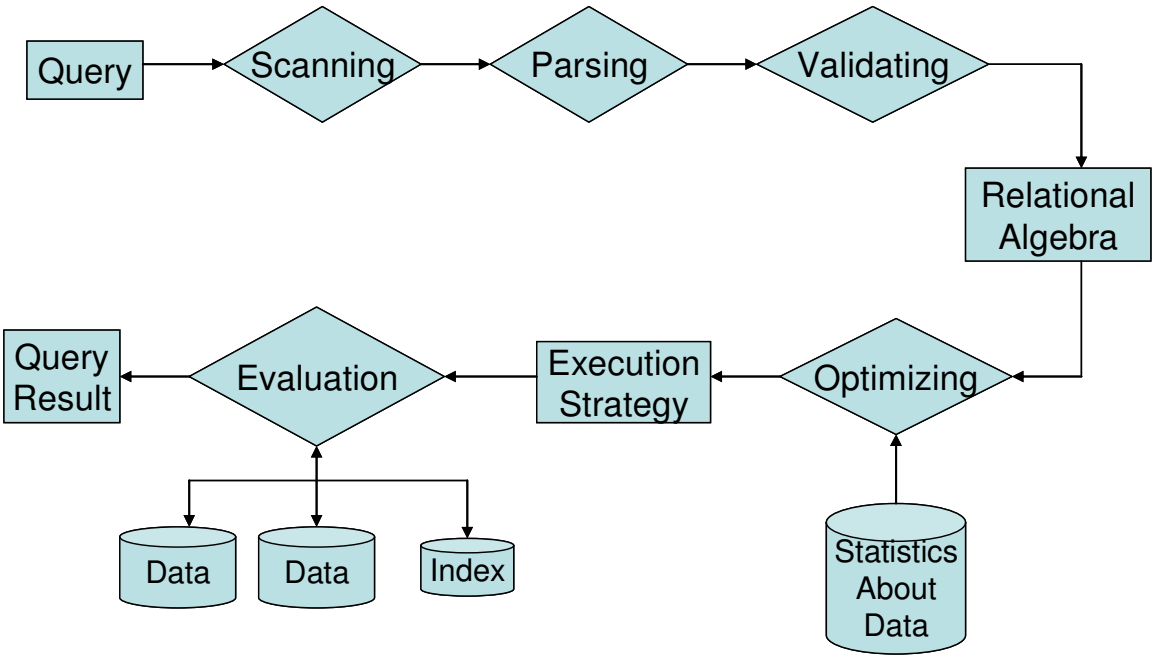


Figure 2- 37 Query Processing Steps

2.4.1 Translating SQL Queries into Relational Algebra

Each SQL expression can be translated into an equivalent relational algebra expression in many ways. We can represent them in query trees that will be optimized later. Each query, typically, is a query block which is constructed from SELECT-FROM-WHERE clauses. It can also include GROUP BY and HAVING clauses.

For illustration, let consider the following query:

Select patient-name, disease, occurrence

From patient-analysis

Where occurrence > 5

The query can be translated into one of two equivalent relational algebra expressions.

$$\pi_{\text{patient-name, disease, occurrence}} (\sigma_{\text{occurrence} > 5} (\text{patient-analysis}))$$

or

$$\sigma_{\text{occurrence} > 5} (\pi_{\text{patient-name, disease, occurrence}} (\text{patient-analysis}))$$

For the top relational algebra expression, the following query tree is built:

$\pi_{\text{patient-name, disease, occurrence}}$

|

$\sigma_{\text{occurrence} > 5}$ use index file1.

|

Patient-analysis

Note that if there is an index on an attribute in the specific query, query optimizer can add it to the tree for more efficient execution.

For some complicated query, the nested queries that have a query inside the outer query, and set operations such as union composed from more than one SELECT clause, need to separate into two or more query blocks depending on the number of SELECT clauses. The separated blocks are also applied to the aggregate functions, MAX, MIN, AVG, SUM, and COUNT.

Consider the following nested query:

```
Select   patient-name
From    patient
Where   primary-doctor   in   Select  doctor-name
                              From    Doctor
                              Where hospital = 'Mount Sinai'
```

The query must be decomposed into two separated query block. The inner query block is for the inner query:

Select doctor-name

From Doctor

Where hospital = 'Mount Sinai'

And the outer query block is for the outer query:

Select patient-name

From patient

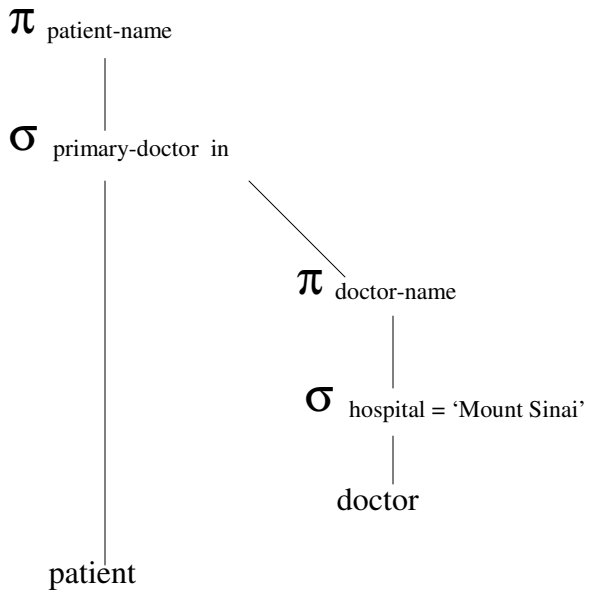
Where primary-doctor in L

L is the result of the inner block. Relational Algebra expressions (showing only π and then σ) for both query blocks are

$\pi_{\text{doctor-name}} (\sigma_{\text{hospital = 'Mount Sinai'}} (\text{doctor}))$

and $\pi_{\text{patient-name}} (\sigma_{\text{primary-doctor in L}} (\text{patient}))$

The tree of the both query block together is:



Since the inner query has no tuple variable related to the outer query, the inner is built and evaluate by only one block. For the complex query that has tuple variable passed from the outer query to the inner query, the query optimizer must be more sophisticate to build more blocks. It is an exception in some database applications that do not accept a nested sub-query.

2.4.2 Methods of Selection Operations

Searching and retrieving records from database files with conditions in the SELECT operation can be handled in many searching algorithms. They are also known as file scans. If an attribute involved in the condition has an index, it will be used to speed up the searching. When an index is involved in the searching, it is called index scan. The following methods are algorithms used in SELECT operation. They start from simple methods to complicate methods depending on what is available in the system.

1 Linear search: The database engine retrieves all records and test attributes, involving in the condition, of records that satisfies the selection condition. If the testing attribute is a prime attribute, the search will stop after it finds the first record that satisfies the selection condition. It is simplest algorithm because it does not use any index. Sometime, it is called brute force method. If any following algorithm cannot be used, the linear search will be chosen. However, because this method may search for all data blocks, it is the most time consuming. If the database file composes of b data blocks, the average cost of searching time is b/2. The linear search is last choices for the query optimizer.

2 Binary search: If the database is ordered on an attribute and the searching involves an equality comparison only on that attribute, the binary search can be used. The database engine performs a binary search on the data blocks of the database file. It is more efficient than linear search because that attribute is already ordered. The cost of searching is only $\lceil \log_2 b \rceil$ comparing with b/2 of the linear search.

The first two methods do not involve any index. The following methods use index(es) if it exists on the attribute involving in the comparison condition. Index provides an access path to locate and access the data file. Since the index is ordered, it is fast and direct. However, overhead cost will rise due to an extra access to the index file itself. For the small size database, using index will perform worse than liner search. The query optimizer will estimate the cost among choices before execution.

3 Searching using a primary index: If the selection condition involves equality on the key attribute, patient-id = x, the primary index is used to search the record. Because of the uniqueness of the primary key, only one single record is retrieved.

If the selection condition involves non-equality on the key attribute, the primary key is used to retrieve multiple records. If the condition is patient-id > 200, the primary ordered index is used to find 200, and then retrieve all records after that record. If the condition is patient-id < 200, starting from the first record, all record will be retrieved until the primary ordered index reaches 200. The record of 200-value is omitted, except if the condition is $\leq$.

The cost of using primary index is less than $\lceil \log_2 b \rceil$ when b is the number of data blocks. Because the data is ordered on the primary key, the records on the primary index are only the first records of each data blocks. If one data block contains 10 records, the

number of records on the index file is b/10. Suppose one index contains 50 index records. (It is more the number of records per block on the index file than data file because each record of primary index contains only the key attribute and block address. It is only two fields comparing with data that usually has many fields, more than two, for each relation.) The number of blocks on the primary index is b / 500. The cost of using primary index for the equality condition on the key attribute is $\lceil \log_2 (b/500) \rceil + 1$. $\lceil \log_2 (b/500) \rceil$ is the height of the tree of the primary index and plus 1 is to read data after knowing the address of data block from the index block. For non-equality condition, the cost is depending on how many preceding or following records.

4 Searching using clustering index: The multiple records can be retrieved by using clustering index if the selection condition involves the attribute that has the clustering index that is for non-key (not unique) but ordered attribute. The cost of using clustering index on equality condition is also the height of the tree of the clustering index plus one and the number of records because it is a non-key attribute.

5 Searching using secondary index: The secondary index can be used if the selection condition involves the non-key unordered attribute that has a secondary index. Since the index is on the unordered attribute, the height of index tree is probably more than primary index. The index record must contain all records of the data file. However, since the size of index record is much less than the size of the data record, more records per block can be stored on the index files. Therefore, the number of blocks of the index files is less than of the data file. It is also ordered index. Searching on secondary index is still faster. Using the same values as above example, the cost of using secondary index is $\lceil \log_2 (b * 10 /50) \rceil = \lceil \log_2 (b/5) \rceil < b/ 2$.

6 Searching using one index on conjunctive selection: If any single simple condition of the conjunctive condition involves attribute that has an index (either primary, clustering, or secondary index), the index can be used following one of the above methods and then applies the remaining conditions of the conjunctive condition to the retrieving records.

The cost of having conjunctive selection with one index is the same as of which index we choose plus the cost of checking remaining conditions on the retrieving records.

7 Searching using a composite index on conjunctive selection: If both attributes A and B of the conjunctive condition has a composite index that is one index composing on two columns A and B together, then the composite index can be searched directly. One of the methods 3, 4 or 5 is used depending on which type of index is created for the composite index.

Also the composite index can be used on the simple condition, not only on the conjunctive condition. Let θ be a comparison operator. If both attributes A and B involve a condition A θ B and there is a composite index on attributes A and B, the composite index can be used. The cost of using composite index would be less comparing with using two indexes separately, if there are two one-column indexes (index on A and index on B).

8 Searching by intersection of record pointer on conjunctive selection: If there is a secondary index that holds record pointers, instead of block pointers, that index can be used. We apply intersection on all conditions among record pointers for each retrieval record.

9 Searching by union on disjunctive selection: The search is harder on the disjunctive condition because when one single simple condition is false, it does not imply false to the composite condition as in the conjunctive condition. If an access path is available to any attribute, it can be used. However, all results of each searching must be applied union operation together. If at least one attribute of the simple conditions on the disjunctive condition does not have an access path, then the linear search must be used. Note that the disjunctive selection seems likely to cost more than the conjunctive selection.

2.4.3 Methods of Join Operations

The term join in this section is referred to Natural join (or Equi-join). Join operation is an expensive operation because it consumes lot of time and resources when joining between two files. For example: $R \times_{A=B} S$

1. Nested loop join: For each tuple r in relation R, we search a tuple s in relation S for which attribute A of R is equal to the attribute B of S. If it is found then the tuple r concatenated by the found tuple s will be stored in the result. Then it will continue to search all tuples s in relation S for the inner loop and to all tuples r in relation R for the outer loop. It likes linear search for the selection operation. Nested loop join is a simple algorithm but cost the most. There is no index used. Sometime it is called a brute force join.

Algorithm:

for each tuple r_i in R

{ for each tuple s_i in S

{ if $r_i[A] = s_i[B]$ then

r_i . s_i is the result.

}

}

For the worst case, the number of block accesses are $blocks_r$ to access blocks on the relation R and $rows_r$ * $blocks_s$ to access the blocks on the relation S (access S for each tuple r) when $blocks_r$ and $blocks_s$ is the number of blocks of data for relation R and S respectively and $rows_r$ is the number of rows (tuples) in relation R.

Worst case: $blocks_r + (rows_r * blocks_s)$

The worst case occurs if the system has a buffer that can hold only one block for each relation. If the buffer is large enough to hold a whole relation, the block accesses for the algorithm is $blocks_r + blocks_s$. That is the best case because each block is read only once.

2. Nested loop join on block: In case of small memory comparing with the size of each relation, it might not be enough to fit a whole relation in the memory. The first algorithm, nested loop join, is implied that the memory can fit both relations R and S. If there is not enough memory, we can load one block of data at a time into memory.

Algorithm:

```
for each block B_k of relation R
{   for each block B_l of relation S
    {   for each tuple r_i in R
        {   for each tuple s_i in S
            {   if r_i[A] = s_i[B] then
                    r_i . s_i is the result.
            }
        }
    }
}
```

However, if one whole relation can be fit in a memory, then only the second relation is loaded one block at a time into memory. So one of the block loops is not needed.

3) Single loop join on index: If there is an index on one of two attributes of the condition R[A] = S[B], then using a single loop on the attribute, that does not have index, to retrieve one record at a time, let say $r_i[A]$ and use the index on S[B] to find the match of $s[B] = r_i[A]$.

Algorithm:

for each tuple r_i in R

{ use index on S[B] to search r_i[A] on S[B]

do

{ r_i . s_i is the result.

J++

} while r_i[A] = s_i[B]

}

It has a nested loop in the algorithm. However because index is ordered, the inner loop will retrieve only consecutive records that is equalt to r_i[A] and then stop. It is depending on the amount of repeating value s_j[B]. Normally comparing with the whole relation S, the number of repeating record on S[B] is much smaller than the number of records of the whole relation S.

4. Sort merge join: If both attributes A and B of relations R and S, respectively, are physically ordered, then we join consecutive rows of R[A] to consecutive rows of R[B], that satisfy R[A] = S[B], because both A and B attributes are sorted. Example of using sorted merge join is when attributes A and B are primary key of tables R and S. That is the case of natural join. If attributes A and B are not primary keys, they may be sorted first and then merge them later.

Algorithm:

Sort tuples in R by A if they are not sorted

Sort tuples in S by B if they are not sorted

i=1; j=1;

while (I <= |R| and j <= |S|)

{ if $r_i[A] < s_j[B]$ then i++

else if $s_j[B] < r_i[A]$ then j++

else //They are match.

{ r_i . s_j is the result.

K = j+1;

while (I <= |R| and $r_i[A] = s_k[B]$)

{ r_i . s_k is the result.

K++;

}

k=i+1;

while (j <= |S| and $r_k[A] = s_j[B]$)

{ r_k . s_i is the result.

K++;

}

i++, j++;

}

}

The number of block access is equal to $blocks_r + blocks_s$ because the pointers I and j continuously move downward. Each record is passed once by pointers i/j on relations R/S. The sort merge join is an efficient method of join.

5. Hash join: The hash function is used to partition tuples in both relations R and S. The partition groups the tuples of a relation that has same hash value on the join attribute. After the partitioning, the searching is used to match two partitions of relations R and S that has the same value on join attributes, R[A] = S[B]. The searching of matched partition is also called probe.

The efficiency of hash join is that the comparison is needed only to tuples that has the same hash value. There is no need to compare to other partitions that have different hash value. There are two other hash joins called partition hash join, and hybrid hash join. These two techniques were proposed so that the hash join can be used when the memory is too small to fit a whole relation in.

2.4.4 Methods of Projection Operation

The method of projection is easy to implement. In case of projection that includes the key attribute, all tuples will be chosen. The method is more interesting when the projection does not project the key attribute. Duplicate records must be eliminated. The sort merge can be used. Since the record will be sorted first, the first record of each duplicating set will be retrieved. That is the merge method.

Suppose A is set of attributes to be projected; π_A (R)

Algorithm:

```
if key attribute ∈ A then
    All tuple of R on attribute set A are the result
else
{   Sort tuples in R by A if they are not sorted
    i=1;
    while ( I <= |R| )
    {   ri[A] is the result.
        K = I;
        while (  I <= |R| and ri[A] = rk[A] )
        {   i++;  /* skip the duplicate */
        }
    }
}
```

2.4.5 Methods of Set Operations

There are three basic set operations in the database system, Union, Intersection, and set difference. All set operation can be used only when the relations R and S are compatible:

1) They must be of the same arity. That is, they must have same number of attributes.

2) Both attribute I on R and S (R[i] and S[i]) must have the same domain.

We can use sort merge to set operations. For union R ∪ S, during scanning through the sorted file, we merge all tuples of both relations. If we found the duplicate, the merge is skipped to the next tuple. For intersection R ∩ S, during scanning, we keep the result only when it exists in both relations R and S. For set different, R – S, we skip when the tuple of relation R exist in relation S. The algorithms of three set operations are below.

Algorithm for union

```
Sort tuples in R and S by unique attributes
i=1;   j=1;
while ( I <= |R| and j <= |S| )
{   if  r_i[A] < s_j[B] then
    {     r_i is the result.
          I++;
    }
    else if  s_j[B] < r_i[A] then
    {     s_j is the result.
          J++;
    else   //They are match. Skip
    {    i++; j++;
     }
}
```

Algorithm for intersection

```
Sort tuples in R and S by unique attributes
i=1;   j=1;
while ( I <= |R| and j <= |S| )
{   if  r_i[A] < s_j[B] then  i++;
    else if  s_j[B] < r_i[A] then  j++;
    else   //They are match.
    {     r_i is the result.
          I++; j++;
     }
}
```

Algorithm for set difference

```
Sort tuples in R and S by unique attributes
i=1;   j=1;
while ( I <= |R| and j <= |S| )
{   if  r_i[A] < s_j[B] then
    {     r_i is the result.
          I++;
     }
    else if  s_j[B] < r_i[A] then  j++;
    else   //They are match. Skip
    {     i++; j++;      }
}
```

Chapter 3 Intelligent Networks (Ins) and Integrated Medical Systems

3.1 Intelligent Networks (Ins)

The functioning of any generic IN depends upon the four essential purposes. The *interfaces* provide for the flow of information in and out of the network. The *monitors* supervise and control flow of the information via the various channels. The *switching systems* switch the channels and complete the physical and logical channels between the user interfaces. The *associated transmission facilities* carry the electrical, optical, or microwave signal in the appropriate medium to convey and communicate the information.

In the public domain, there are five basic building blocks of intelligent networks, Service Switching Points, Service Control Points, Signal Transfer Points, Service Management System (SMS), and Intelligent Peripherals (IP).

3.1.1 Service Switching Points (SSP)

Service Switching Points (SSPs) are physical switch that performs the switching function, and the logical software modules that reside within the switch forcing the execution of the SSP functions. SSPs are generally located at or in close proximity to the switching systems that contain call-processing software. The switching system, which hosts the SSP, may be an end office or a tandem switching facility. The access tandems use Common Channel Signaling (CCS) to facilitate interoffice signaling and also allow the facilities to use interexchange carriers. The service switching points react to the specific triggers from the customers. In response, the service switching points transmit queries to a centralized service control point (SCP) within the network. When the SCP responds to the query from the SSP, the SSP instructs the switching system to complete

the call attempted by the user. The call-processing software of the switching system is then invoked to complete the call. The SSP/2 in the Intelligent Network/2 may be a circuit-switched, channel-switched, or even a packet-switched center.

3.1.2 Service Control Points (SCP)

Service Control Points (SCPs) contain the active database of customer records. Three distinct types of database in SCP are 800 numbers, alternate billing system (ABS), and private virtue networks (PVN). This database actively queries from the Service Switching Points within the network to seek and obtain service-completion information. Because of high volume usage of SCP, the design of the SCP is considered a highly efficient parallel processor and data storage retrieval computer system. There are four major hardware components of SCP.

- An elaborate and highly dependable mass storage system is the database storage. It is typically composed of large capacity disk drives and disk controllers.

- A bank of parallel processors with their own dedicated memory blocks serves to access the bulk storage within SCP. It also communicates with input/output devices in the database.

- A series of front-end processors preprocess the queries received via the Service Switching network such as SS7 or similar compatible network, with well-specified protocol.

- A series of BX.25 ring front-end processors receives service management information from the SMS and provides the interface for maintenance, security, and operations of the SCP.

The suitable computer hardware architecture should be parallel processor, multi-bus systems with optimally distributed bus structure to perform the SCP functions. With high dependability expected from the SCP, duplication of hardware and buses increases the confidence level in the functioning of these special-purpose computer systems.

For the software aspects, the architectural configuration of SCP functional software consists of five major modules.

- The service network interface receives the signaling information from the Common Channel Signaling 7 (CCS7) and other signaling systems querying the SCP and passes the information for response from the SCP. This major and perhaps the most optimally coded group of software routines serves to interface the SSP queries.

- The support system interface provides a software interface for the service management functions to be handled via the SMS. Network management control and customer record entry are updated and handled via this interface.

- The observation window for the SCP operating personnel, which functions and maintains the SCP, is provided by the operations subsystems. This interface responds to a system console that monitors the flow of data and information among the SCP components.

- The node manager subsystem fulfills the start-up and shutdown procedures of the SCP. It also serves the service continuity, early fault-detection and some possible rerouting. The multitasking and load sharing of numerous central processing units within the SCP are also managed by the node manager. In general, it serves as an operating system in computer system.

- The node administration subsystem permits efficient and optimal database functions and provides for the necessary backup of databases. This function is considered as fault-tolerance and service dependability.

3.1.3 Signal Transfer Points (STP)

Signal Transfer Points (STP) are distributed within the Common Channel Signaling (CCS) network. The STP nodes are high-capacity, extremely reliable packet switches for the transportation of signaling information between SSP, SCP, and other STPs. The four main functions of the STP are:

- error-free message routing
- protocol processing
- address translation
- message routing database look up.

The STPs contain the translation information necessary to forward the database queries from the service switching points to the appropriate service control points. Traditionally, the CCS network works in the packet-switched environments and the STPs constitute highly reliable packet switches, operating as two physically separated (and duplicated) pairs within the CCS network to ensure network integrity. The STP also needs some fundamental database support for performing optimal signal transfer and selecting the appropriate service control point. With appropriate software support, the STPs in the CCS can relay the control information in the CCISS7 signaling format throughout the network and respond to SSP queries and relay responses.

3.1.4 Service Management System (SMS)

Service Management System (SMS) is intricately tied with the SCP. The SMS is an off-line support facility used to enter customer data into the SCP's databases within the Intelligent Network. The SMS communicates with the SCP through interface cards that process the BX.25. The service management systems also need a series of front-end processors to link with the dual buses, which provide data paths between the other front-end processors of SCPs and the CPU-banks of the SCP. Since each provider can have an individual SCP for the particular type of service, the SCPs of each of the service providers have to respond in a consistent manner to the SMSs. The SMS also must be functionally transparent.

3.1.5 Intelligent Peripherals (IP)

Intelligent Peripherals (IP) is a stand-alone network-compatible element, which can connect to a switching system. The signaling to and from the IP is communicated via the Signaling System 7 (SS7) network to signal transfer points (STPs) in the IN. Intelligent peripherals also transfer voice and data to and from switching systems. In view of routing efficiency, the IP is generally connected to only one switching system, even though a switching system may have numerous Ips. The IP performs specialized functions in the network that need specialized and evolving telecommunication resources. Some of these resources include customer-specific announcements and/or service-specific announcements, voice synthesis, digit collection, and voice and data encryption. The IP is accessed by two groups of users: the caller when IN service is demanded by the customer, and the operating-company personnel as new services are being configured. A

typical example of the first case is when customers may want to recover an announcement stored for them. Upon customer verification (identification code or voice analysis) by the IP resource, the stored announcement may be made available to the customer by the database from which the announcement was retrieved. A typical example of the second case is when the operating-company personnel may be attempting to provide a new service by using the functional components (FCs) of the network resource management facility and the associated IP.

In the Intelligent Network 2 (IN/2), service switching systems, service control points, and intelligent peripherals integrate three rudimentary logical modules, service logic interpreter, network information database, and node resource manager.

3.1.6 Service Logic Interpreter (SLI)

Service Logic Interpreter (SLI) functions as an application of the execution of the service logic programs (SLPs). These SLPs are a collection of functional components (FCs) which use the network resources to accomplish specific submodular functions, for example, collect digit arriving from the customer, and forward dialed number. The SLO provides a multiprocessor execution environment. Several SLPs are executed as separate processes. Multiple invocation of the same SLP is necessary for different applications. Chain linking of SLPs permits them to initiate other SLPs. SLP priorities are necessary to permit efficient operation of the service logic interpreter and service management. The SLI also provides real-time response to network queries requesting call-handling instructions or IN/2 service provisioning. In the hard-wired logic, the assembler instruction performs the lowest level subfunction. On the other hand, at the highest level,

a specific application program, or software package in the computer environment, corresponds to the service logic program, which can be encoded in a service creation environment (SCE). The SLI may also respond to queries from other SLIs in the network. The execution of service logic program provides the answers to queries. These service logic programs may be written by network personnel or customers. Therefore the SLPs also access the databases that are generated and updated by the ISMS supporting that particular node.

In the execution mode, the SLI accomplishes four system services.

- Message handling between SLP and supervisory processes is carried out be SLI.

- Various processes invoked the different SLPs are managed. The data access and data structures are created and managed by SLI.

- The management of time and resources to achieve the multiprocessor environment is accomplished

- Error conditions and exceptions during the execution of individual SLPs are also handled by the SLI.

3.1.7 Integrated Service Management System (ISMS)

Integrated Service Management System facilitates the control by both the customer and the operating companies over the numerous network services foreseen in the IN environment. The IN/2 environment is enhanced to a generalized version with flexibility in offering and operating the newer network services without major software revisions.

Service providers (in different geographic areas, for instance) will have a standardized version of an ISMS, which can exchange control information. This control information may relate to the functions of SCP, SSP, or Ips in the network of the service providers. However, the ISMS of both service providers will be updated by a centralized administration and coordination (CA&O) facility.

The functions of the ISMS can be classified into five major categories.

- The ISMS receives (and transmits) information from (and to) the CA&O pertaining to the independent networks of the participating service providers.

- The ISMS exchanges updates of pertinent SCP, SSP and IP control information between service providers such that the IN functions are transparent to the users.

- The ISMS responds to the control information of its local service administration (SA) facility for updates and changes in the service environment.

- The ISMS dispatches the SCP and updates it with information received from other ISMSs, CA&O facilities, and its local SA.

- The ISMS dispatches the SSP and IP updates to local service switching point and intelligent peripherals.

3.1.8 Network Information Database (NID)

Network Information Database contains the customer-access, network-connection information. It facilitates which type of calls can be routed and which services can be provided to each individual customer. This data is stored in the form of a networks capabilities map. The database may be cross-queried by other modules dispersed in IN/2,

by way of appropriate data lines on the X.25 packet networks linking different IN/2 elements, or by ISDN lines between the network elements.

3.1.9 Node Resource Manager (NRM)

Node Resource Manager of IN/2 locates and identifies the address of network resources required to continue call processing, or provides the next increment of network service. The NRM also provides the SLI with information required to establish a connection to an IN/2 module.

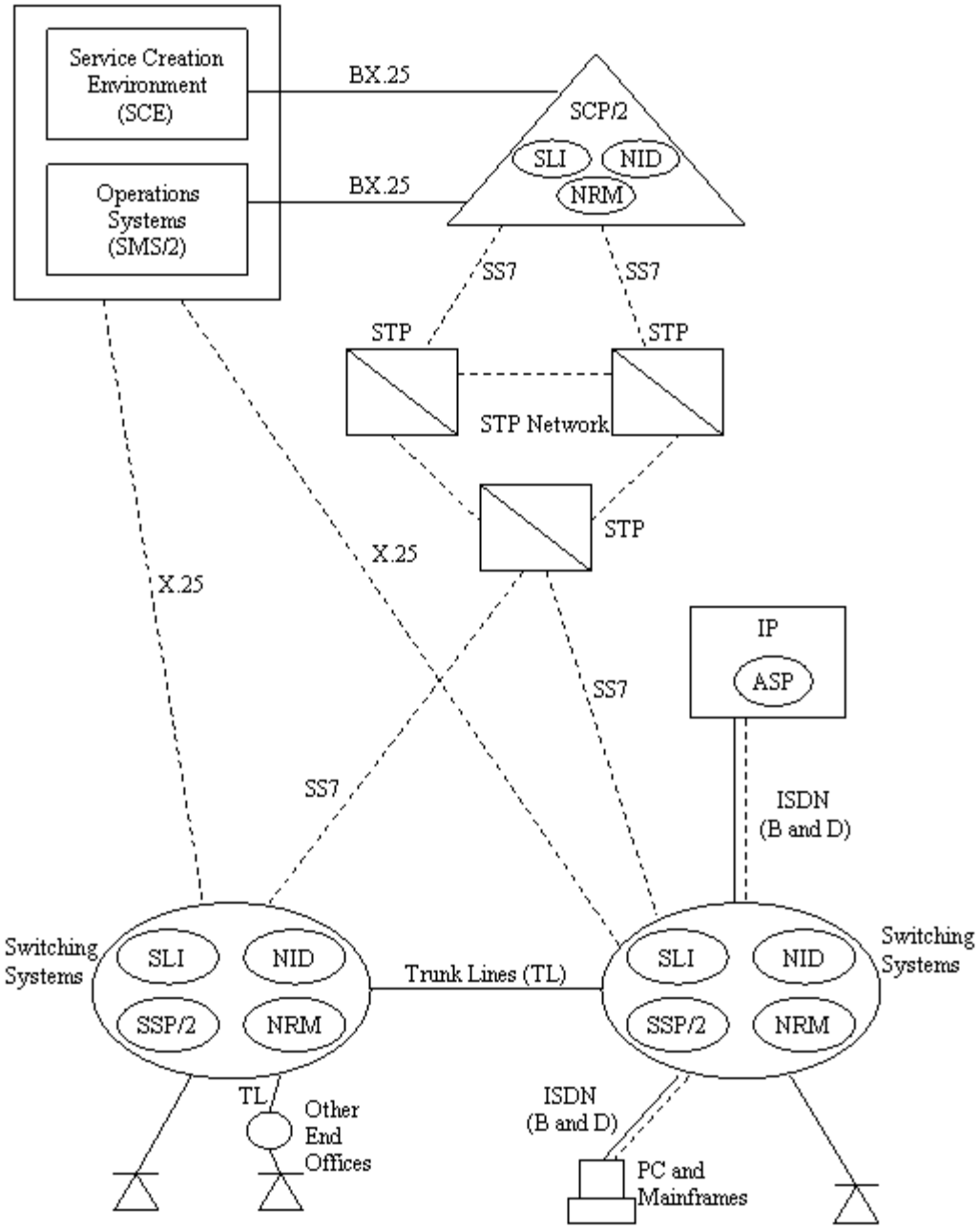


Figure 3- 1 The Architecture of Intelligent Network. (IN2)

3.2 Integrated Medical System

In past several years, the routine medical care and functions have been implemented on the network through the Internet. They have enhanced the productivity and efficiency of medical environments by confining the procedural steps, book-keeping functions and resource allocation. The human functions of pattern matching, judgment, and decision-making are also facilitated by a backbone of computers and network that is organized to perform knowledge-based AI functions. It is similar to the corporation to perform on the Management Information System using the Decision Support System that is quickly and routinely performed the decision-making and analysis on the transactions and other stock market environments.

3.3 Medical System Architecture

3.3.1 The Medical Processor Unit (MPU)

The medical applications need to focus on specific knowledge and experience. Traditional CPU architectures involved in typical data processing are not suited for the medical applications. The medical processor unit (MPU) generates a sequence of subprocedures for medical procedures such as patient history, capability of physician's expert system and capability of the medical facilities for a particular environment.

The operator/operand function of the MPU is based on knowledge of the patient that is in the patient database, experience of the medical team to categorize with the records of the patient. So the expert and knowledge-system based programs are used consistently with the capability of the medical facilities.

The microprograms in the MPU will initialize, accumulate, and function the knowledge from three different ways:

1. from the approved current concepts and breakthroughs to the expert system database.

2. from the practice of the particular physician and the medical facility to be consistent with knowledge level of the physician and the capabilities of the medical facilities.

3. from the experience of this physician and earlier physicians with the particular patient.

The MPU will use the opinion of the current expert, the approval of a team of physician (if necessary), and the verification processing to produce the output. In some case, checking and verifications are made before the administering any procedure to a patient and the patient history.

The output of the MPU is a series of instructions for subprocedures that will be dispatched to the actual medical facilities after appropriate allocation of existing medical resources to the patient's subprocedures. Scheduling, resource sharing, and priority allocation will be done automatically depending on the type or code of service being performed (emergency, standard, diagnostic routine, etc.)

The subprocedure instructions are categorized as knowledge-based inferential instructions, search instructions, and administrative instructions.

The knowledge-based inferential instructions query the cause-effect relation for the logical and predictable effect. The precondition, that is a strong point but not a certainty, will be investigated. The effect of each step produces a confidence level and

makes a chain of subprocedures that will make up the product of each individual confidence level.

The search instructions are separated into two groups, the search in the knowledge domain and the search in the data domain.

The search in the knowledge domain, such as professional, medical opinion, possible cures, diagnostics, etc., deals with the incomplete input of which branches of knowledge tree are logically associated. Two mechanisms based on the proof-theoretic interpretation of rules are used. The first mechanism is the bottom-up inference mechanism (forward chaining) where the inference starts from the given facts and generates additional facts that are matched to the goal of a query. Another mechanism is the top-down inference mechanisms (backward chaining) where the inference starts from the goal of a query and tries to find constant values that make it true. The latter mechanism has been used by Prolog. The knowledge search instructions can be invoked by the knowledge-based inferential instructions and vice versa.

The search in the data domain, such as files, patient information, database, insurance company, service providers, drug vendors, etc., deals with the complete input to process and offer the complete and definite result. For example, given the ID (mostly social security number) of patient, IMS searches the illness of patient in last two years.

The administrative instructions are for the local medical facilities that involve the personnel, support staff, and other services providing to patients. These instructions include scheduling of physicians or operation rooms, allocation and consistency of the physicians, accounting and billing updating the information of patient databases.

The medical processor unit, MPU can be separated into two types by their speed and capacities.

Remote Medical Processor Unit handles the batch type of job processing. It deals with any patient ID to a desired procedure and a primary medical data of the patient (such as heart rate, blood pressure, blood density, and temperature) as the paramedical team to enter these data. These jobs do not require the real time response. Mail or email to the doctor can be waited for the response. Therefore the capacity of this MPU can be compromised on the memory size and database capacity to act in conjunction with the AI-based programs. However, since it is the remote request and handles more than hundred input location, the MPUs need to be on the Intelligent Network Based.

Hospital-Based Medical Unit handles the on-line processing. The real-time response needs for the information from the hospitals to continuing process from a mainprocedure to other subprocedures. The search on the numerous knowledge-based to achieve a high confidence level is a time-consuming task. To achieve a hundred percent confidence level, it insists on insufficient input data and the exhaustive knowledge bases search or with sufficient input data and the inadequate knowledge bases.

3.3.2 Two Architectural Approaches

We explain two approaches of medical systems:

- Processor-Based and Local-Knowledge-Bank System
- Distributed-Knowledge-Bank System

- Processor-Based and Local-Knowledge-Bank System

This approach, the Processor-Based and Local-Knowledge-Bank System, has the knowledge medical databases tied together with the processor. The processor can acquire the knowledge quickly. The addressing is done by the bus-selector whose information is stored in a knowledge bank that can reduce the seek time of the massive information via switching module. This knowledge bank keeps the addressing of the knowledge bus, patient bus, procedure/lab bus, physician bus, input bus, and output bus. Physician can access the knowledge and patient information, and check the results of the observation, procedure and analysis. There are two considerable designs of bursting information from the memory via bus to processor. One is to burst entire relevant information from the knowledge bank through the bus to the main memory of the MPU where the instruction is executed. Another design is to send the complete instruction, dispatched by the MPU, to the knowledge bank and the knowledge bank uses its own processor to execute the instruction or part of instruction. The result or partial result is dispatched back to the MPU.

Therefore, every subprocedure is executed. The net result, generated from subprecedures, procedures, and the entire usage of the IMS, will be conveyed back to the user.

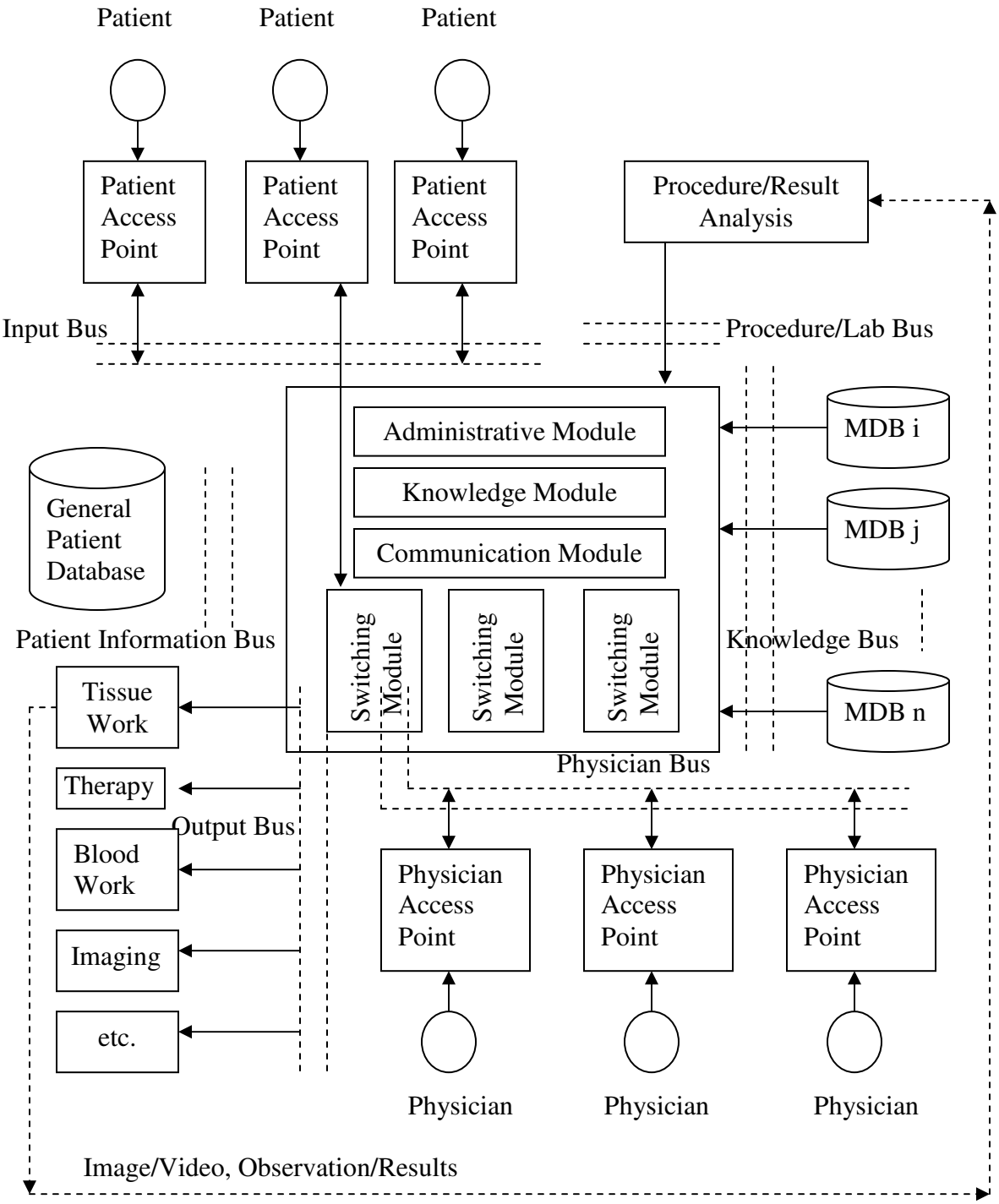


Figure 3- 2 Architecture Approach of a Hospital-Based Medical Processor Unit.

- Distributed-Knowledge-Bank System

Many MPUs and knowledge banks of this architecture, Distributed-Knowledge-Bank System, are linked together from distance via the high speed network. The addressing of the distant knowledge banks and other nodes are stored as identifier. This identifier of the knowledge bank is consistent with the information stored in a particular knowledge bank to reduce the seeking time. The isolated packets of information arrive at the knowledge bank from many IMSs. The packet switching network is used to burst the instructions to the knowledge bank. Every subprocedure is executed via an individual packet command. The net result conveyed to the user is a series of packet transactions. The transactions, processed by a single IMS and, possibly, multiple knowledge banks, are accumulated from procedures and subprocedures. The entire usage of the network-based IMS processes on the distributed computer environments.

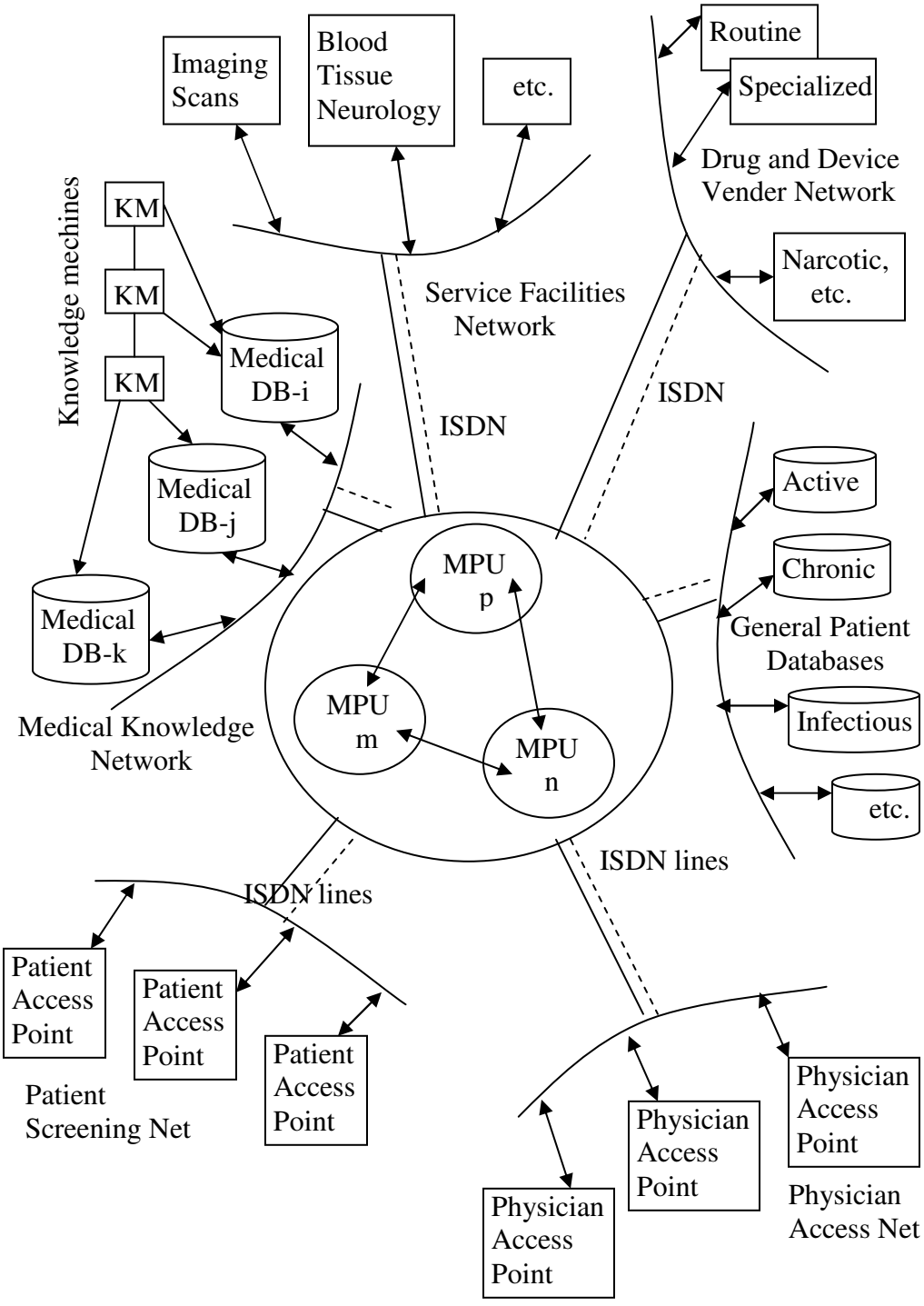


Figure 3- 3 Architecture Approach of a Network-Based Integrated Medical System

3.4 Knowledge Processing for Medical Database Network

A medical knowledge system processes upon the available current knowledge of both medicine and patients. The awareness of current events that are discovered into knowledge base is finite probability. The system must have capacity to store, retrieve, and update knowledge as information, ability to process them intelligently, and applicability to acquire the level of confidence from them for medical problems. Although the current computer system can serve those requirements, the system must also be able to acquire, or learn new knowledge and modify it into the knowledge machine. In order to do so, the knowledge processing machine must process information in two dimensions, the knowledge domain that generates incremental and integrated conclusions and the numerical domain that generates the level of confidence. Therefore, the hardware for the knowledge processing system has three basic distinct building blocks, the database management unit, the knowledge processing unit, and the numerical processing unit.

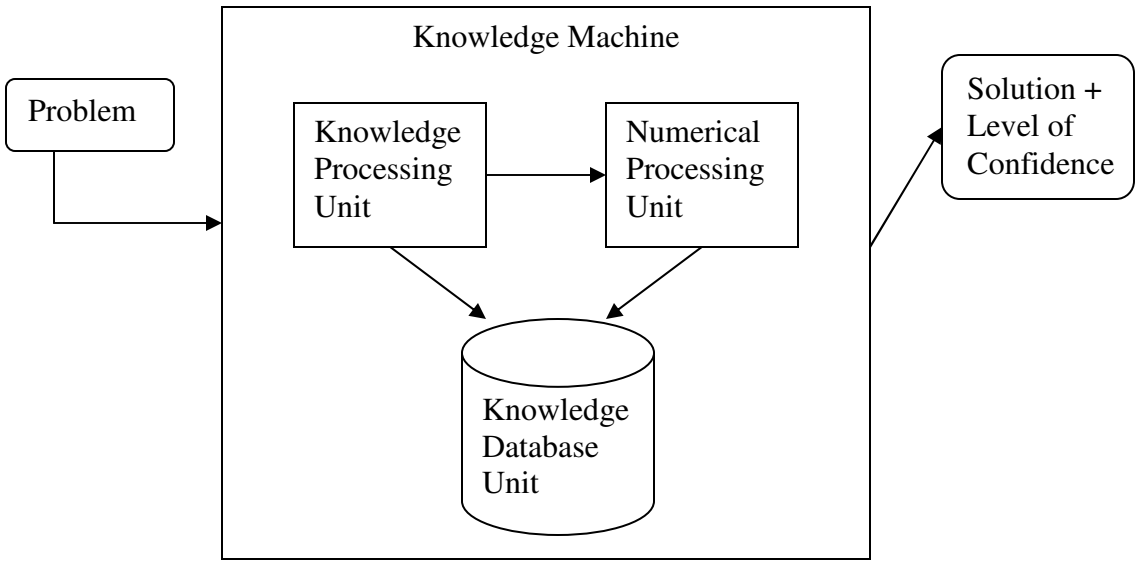


Figure 3- 4 Knowledge Machine Concept

3.4.1 Knowledge Database Management Unit

Database systems have been evolved from the basic file management system on a stand alone machine to database server that shares among client, and to the sharing information on the internet. The evolution has expanded on various topics such as data management, information distribution, and information security. In communication environment, the database design has been modified on both hardware and software designs to suit specialized need. The architecture has a front-end interface to permit a protocol conversion and decoding the arrival of the SS7 (Signaling System 7) packets or other nonstandard protocol structure into the NID (Network Information Database). A bank of central processing units (CPUs) processes the packets on the memory through the data bus using DMA (direct memory access) from large disks. If the databases are logically or physically separated, then the accessing and processing on them can be done on parallel basic. Therefore the interrelated blocks of data must be designed to function quickly as memory module that directly processed by the CPU bank.

Since much information must be stored, retrieved, and updated, the databases should be able to manage on its own. The major functions on the databases requires a set of dedicated CPU's from the CPU bank to reduce the share time processing from the CPU bank into the database unit. This is similar to the system that has dedicated processors on I/O devices to perform input/output functions. The disk buffer must also increase to dedicate to the database unit as the database-dedicated CPU's performs database functions on their caches.

As of storing information, the database unit should be built to handle the integrated environments of all user information, such as integers, complex and double

precision numbers, text processing, graphics, audio visuals, and so on. It also requires the responsibility from the user to manage (store, retrieve, and update) this integrated information in the appropriate format. Objects participating in an event could be input to the database unit of the knowledge machine. They would be far more complex than numerical data that routinely process in early program. Hyper-spatial complexity that is routine cycle of human transaction can be categorized into two parts: object as of human and event as of its transaction. The database unit of the knowledge processing unit is designed to manage objects and their events. The representational format of objects and their events for the generalization would be highly complex. Programs to handle the objects, events and their procedure become too varied and too many details for a simple operating system to manage such as page placement/replacement among virtual memory, main memory and cache for the memory management, disk scheduling and etc. Therefore, the separated database unit including both software and hardware architectures is necessary for the knowledge processing unit.

Besides handling the access and management of disk and memory, other functions of the database units are address tracking and translation, also called mapping, of objects and events; program procedure to associate/disassociate, enhance, and truncate objects, events, and their relationship; and their utility routine in the library.

3.4.2 Knowledge Processing Unit

Functions of a knowledge processing unit are more specific than a regular central processing unit CPU that processes numerical calculation on arithmetic logical unit ALU. Knowledge processing unit executes assembly–level knowledge instructions that support

binary knowledge instructions. CPU handles the central role of a traditional computer system while knowledge processing unit handles the important role on the knowledge microinstructions. Microinstructions in the knowledge processing unit have two essential parts: the knowledge-base operation codes and the objects to which the knowledge-base operation codes are performed. These objects act as operands on an operation.

The knowledge operation code can be a real knowledge domain function or a pseudo knowledge operation code that can be used to reserve for an operation code for new knowledge. A real knowledge operation code performs knowledge domain functions on the object operands. Knowledge domain functions manipulate objects and interpret a given set of events from the definition or the context of the problem. As the knowledge stores as information in the database unit, the knowledge processing unit categorizes, matches, infers, extends, analyzes, and recognizes the events base upon medical situation for which the medical knowledge machine is solving.

The execution cycle of a conventional CPU runs in four steps: fetch a new instruction from memory, decode the instruction into operation code, execute the operation code by arithmetic logic unit, and store the result back into memory as a binary number for use by later instructions or passing via data bus to a disk. Applying it to the knowledge processing system, the knowledge processing unit has an execution cycle running in two modes: solving a problem and learning new knowledge. In both modes, the knowledge processing unit follows the sequence cycle as of CPU, fetch, decode, execute, and store the new knowledge for later use. In the solving mode, the knowledge operation codes are distinct from the knowledge operation code in the learning mode. The knowledge operation code of the solving mode obtains new knowledge from the result of

execution while the knowledge operation code of the learning mode extracts information from the operand(s) of knowledge instruction. Generally, the result of execution of the knowledge operation code in the solving mode is new knowledge with a number about the confidence level of the validity of the instruction. Procedural learning in the next step can be obtained from a sequence of knowledge instructions learning from the previous, similar or a pattern of earlier instructions.

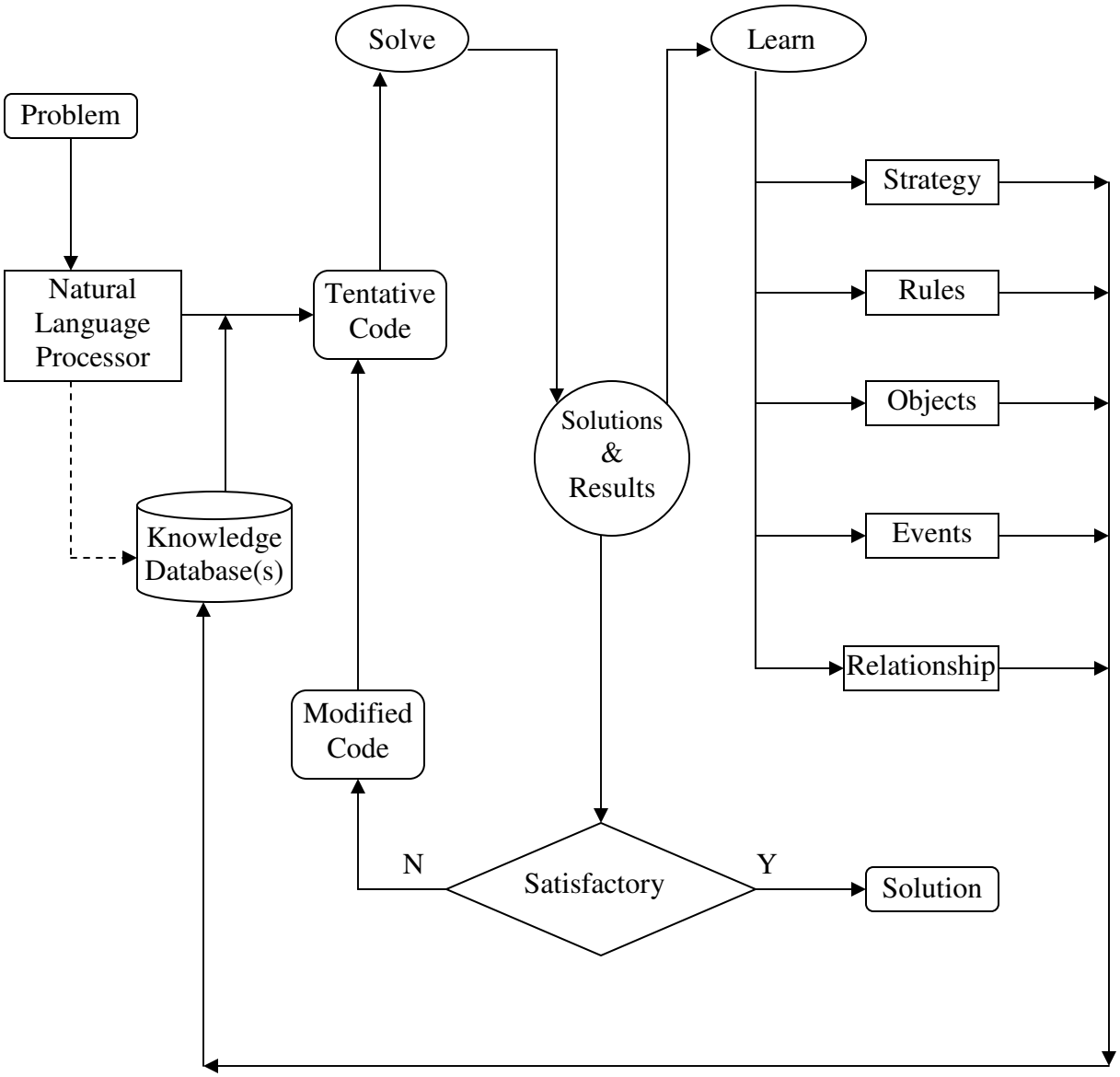


Figure 3- 5 Knowledge Processing System Functions

Figure 3- 5 presents an overview picture of the knowledge processing functions of the knowledge processing unit. In the solving mode, the system is solving the knowledge problem from the effect of events upon the group of objects related to the problem. First the natural language processor identifies the objects and their attributes participating in a set of events. A goal is set. A solution tree may be built if there are sub-goals. Then the tentative code is generated to the solving unit. The tentative code will be recycled through the execution loop into modified code until a solution is satisfied according to a group of criteria. Thus the machine terminates the program. During the time when the intermediate results are being generated, the learning mode retrieves programs and objects from the "solutions and results" unit. It learns strategies and rules from its program. It also learns objects, events, and relationships among them to record into the knowledge database for recycle processing in the future at the solving mode with different problems but similar objects, attributes or events. However, if the problem and its objects are totally new, the system will not be able to solve them. The introspection mode, as shown in picture, will learn the problem and their solutions as ripple effects and implications, implied relations, solution similarities, and preferred strategies.

The first step of execution cycle is the fetch. The knowledge processing unit fetches knowledge instructions from main. The second level fetch may occur if an object has a specific indirect address that would be either main memory or cache memory. The database unit will be activated if the indirect address of the object referring to the address in the database unit. Objects do exist not only in cache memory, main memory, and database unit, but also in the network environment attaching to the system.

The construction of object cache memories and object main memory are similar. The object cache memory can be constructed either by the unique functional specification of the hardware design using Very Large System Integration, VLSI, or, similar design by most operating systems, a data structure of address-mapping table. The object cache memory includes columns of object identifier number, the parent (including grandparent and higher), and child (including sibling, grandchildren, and lower) objects of the current object, it attributes and other related objects that may link in the past with the level of confidence when the object has relationship with it. The object main memory is constructed by containing all of those records as in the object cache memory and the object environment of all objects participating the knowledge processing of the knowledge program. The database unit provides the data to the object main memory. The network environment also provides the object environment from other distributed database or other logical join-unit of the knowledge processing environments.

The second step of the execution cycle is the decoding of instructions of the knowledge processing unit. A knowledge operation code is decoded depending on its mode of executions, either solving mode or learning mode. If a knowledge operation code is decoded for the solving mode, its solutions and results will be sent to the test of satisfactory. If it is not a satisfactory answer, the solutions and results will be recursively applied to produce a new modified code for solving the same problem. If a knowledge operation code is decoded for the learning mode, its solutions and results will be learned, for later analysis, in strategy, rules, objects, events, and their relationship. Therefore, a set of specified knowledge operation code must be generalized and defined for each sub-

function to execute in two modes of a function of knowledge processing unit. It is must also evolvable for further development.

After the decoding step, the solving or learning mode was identified for the execution step. The type of knowledge operation code is identified. The attributes within the object environment are processed. The object environment of that object is updated according to the knowledge micro-function. Therefore, every knowledge operation modifies the operation environment of the operand of the object(s). The learning mode at this step of the knowledge processing unit enhances the object environment. The basic knowledge of objects in the object environment is usually in a tabular form as in a database. Classification of object attributes can be learned at this step. The extent of object environment details about related objects can be built. For the knowledge machine in the medical distributed network for each local of a region, the operation environment of objects such as doctors, nurses, patients, patient's information, and etc. can be built. The knowledge processing unit functions to achieve its micro-functions in order to execute its knowledge program. The knowledge machine must give an optimal procedure depending on the availability of the current object environment such as doctors, nurses, patients, patient's information, and etc. to the user. The time, day of other addition constraints may also be considered.

The last step of the execution cycle is the storing step. It is necessary to define objects on knowledge machine as a variable defined in the binary machine. The knowledge operation code of the knowledge processing unit and the declaration of object type define the storage procedures for the result. The programmer at the machine-level language, such as assembly, and the compiler are responsible to optimize the class of

object in the operation environment. The syntax of the knowledge programming must be identified. So the knowledge program can be implemented as an application program for the ordinary computer. The programming may be specific or detailed. So its result is a single and well defined certainty. Simple deduction or major conclusion can be made depending on the program. However, if the knowledge programming is not specific, the various solution and results may be drawn. Each of them contains the level of certainty (or uncertainty depending upon the program). So the knowledge processing unit must reevaluate the solutions and results again through all the steps by using its knowledge database storing early solutions and results as objects, strategy, rules, events and relationship to solve the problem. Validation of hypothesis may be programmed to identify the probability of the hypothesis based upon the available information of objects in the knowledge database.

3.4.3 Numerical Processing Unit

The numerical processing unit provides basic numerical data processing as the arithmetic logic unit of a typical computer and also the numerical level of solutions in terms of levels of confidence, and level of irrelevance and ambiguity. A problem feeding into the knowledge machine could be indistinguishable and wide from a scope of definition. The processing of such problem would be uncertain, reluctant, or even error. The numerical processing unit, supported by knowledge processing unit and knowledge database unit, must provide the estimation of the probability of the solution of which we call level of confidence in percentage between 0 and 100. If the problem is clearly defined and its answer in the knowledge database is perfectly matched with the problem,

the numerical processing unit will provide the answer with 100 percent of the confident level to the user. Otherwise the knowledge machine has to solve a problem with a level of confidence, providing to user, in percentage that is less than 100.

Even though the level of confidence is less than perfect, less than 100 percent, the numerical processing unit still provides at least acceptable value of level of confidence when the knowledge processing unit has solved many problems and saved the new knowledge into the knowledge database unit. More problems to solves, better, likely, level of confidence to a new problem. The knowledge machine provides a series of all possible solutions that are higher than a preset level. For each step of problem solving, the numerical processing unit assigns a level of confidence to each solution. Along the recursively solving, some level of confidences of some solution may be increasingly or decreasingly updated according to the knowledge in the knowledge database providing to the knowledge processing unit to solve the problem. When the knowledge processing unit reaches the final stage of results of the problem, the level of confidence indicates how confident those final solutions are.

For example, in our research of medical problems, the knowledge processing unit processes a problem through the medical steps in the way a doctor gathers information from a patient. The knowledge processing unit starts from collecting symptoms from user. User provides symptom(s) that he/she recently notices. The knowledge processing unit tests these symptoms with the knowledge database to find other common symptoms that probably occur to the patient. During this process, the numerical processing unit generates the level of confidence to each symptom and disease related to them. The common symptoms are shown to the user to interact whether he/she has had the

experience with these shown symptoms. After the user provides the answer to the knowledge machine, the knowledge processing unit solves the problem for diseases that are likely to occur in the patient according to the symptoms given by user. The level of confidences of each disease is updated by the numerical processing unit. At the same time, the knowledge processing unit retrieves causes of these diseases from the knowledge database.[4] The knowledge machine shows likely diseases to the user with their level of confidences. If the user realizes the experiences of some of these causes, he/she can provide to the knowledge machine. The knowledge processing unit combines these user-experience causes with the likely diseases to find diseases that are more likely to occur. During this time, the numerical processing unit applies these user-experience causes and the current diseases and their levels of confidence to update to the final result.

The final result may not generate 100 percent of level of confidence. Nevertheless, before giving a final solution to the user, the acceptable level of confidence of solution has to generally satisfy with the expect criteria, such as expect risk, cost, and so on depending on the type of problem. If the level of confidence cannot reach the expect standard such as risk, the final result must be omitted or consult with the expert before showing to the user. With the numerous results given out, the knowledge machine offers composite traits and consequences rather than single most possible solution.

[4] Each cause may interact with level of occurrences if known. So the user should be shown from the most to the least of the common occurrences. These levels of occurrences can provide better level of confidence to the knowledge machine. For less complexity of this research, we leave this level of occurrences for the future improvement of the research that is ongoing.

3.4.4 Mode of Executions

As mention earlier, the modes of execution in the knowledge processing unit are the solving and learning modes. When the system is in the learning mode, it must learn all about the objects and their attributes and events. It studies the previous actions of humans who interact with the machine to extract rational information such as strategies, rules and relationship among them. The learning process occurs during and after the solving mode who intelligently applying new information with current knowledge into the solution of the problem.

Solving Mode

The solving mode occurs during the execution of knowledge operation code by the knowledge processing unit. The knowledge processing unit processes the knowledge domain function upon objects. The operation code defines a knowledge function by finding the relationship of two or more objects by their attributes and events. It also defines a knowledge function by generating attributes to objects or by finding a precondition that leading to objects or conditions. Therefore, the operands for operation codes are objects or conditions. Each knowledge operation code is passed into knowledge control unit where it is decoded. Knowledge operation codes and pseudo knowledge operation codes are categories by their function. The knowledge operation codes functions on definite micro-function on the objects whereas the pseudo knowledge operation code operates functions based on the result(s) of knowledge operation code. For example, after a knowledge operation code finishes a function to prepare a machine to ready for the next knowledge operation code, then a pseudo knowledge operation code is defined.

Examples of basic knowledge operation codes for the knowledge processing unit are:

- single object knowledge operation codes,
- multiple objects knowledge operation codes,
- single attribute knowledge operation codes,
- multiple attributes knowledge operation codes,
- single parent object knowledge operation codes,
- single child object knowledge operation codes,
- multiple antecedent object knowledge operation codes,
- multiple descendent object knowledge operation codes,
- memory oriented search knowledge operation codes,
- database oriented search knowledge operation codes,
- and network oriented search knowledge operation codes

The knowledge control unit of the knowledge processing unit dispatches control signals to perform the hard-wired function or interprets the micro knowledge code in the programmable knowledge processing unit after the knowledge domain function is decoded by a decomposition of knowledge operation code into its microscopic hardwired functions. Knowledge operation codes may depend on other knowledge operation codes. After the set of knowledge operation codes are identified, the knowledge machine can fetch to the next execution cycle.

Learning Mode

Another mode that is very important for execution of the knowledge processing unit in the knowledge machine is the learning mode. There are many algorithms of learning process for knowledge processing unit. When knowledge processing unit executes the knowledge operation codes, the knowledge machine learns which objects form groups, attributes and their relationship among them. The knowledge operation codes and objects executing on it could be learned together. Also processing information that produces new information can be included in the knowledge database unit. Therefore, the expectation of outcome of knowledge operation code upon objects is the next learning step. Four major levels of learning process are described.

The first level of learning process is to learn from the occurrences of object or group of objects and their events, also called attributes. The knowledge processing unit identifies the object or group of objects and classifies these objects in terms of their occurrences. The identification of objects can be done by using the rules of pattern recognition. The coordination of properties among objects is mapped into the knowledge database for future use. For example, in the medical network, when there is an occurrence of a new disease, the knowledge processing unit scans the properties of disease such as the symptoms, recent or just-found cause from a patient record who is recently sick by this new disease.

The studying of the past information in the knowledge database can lead the learning process into a clue to new knowledge. The collection of groups in the recent cases and groups in the past can scale the knowledge into proportion together. The knowledge processing unit would find all probability of each group of objects that

leading us to the level of confidences, mention earlier, to the knowledge database. If the implementing program on the knowledge processing unit uses pointer concepts, the knowledge processing unit can collection these objects from the forward and backward pointers that indicate the coordination among objects currently using.

The second level of learning process by the knowledge processing unit is to learn the relationship between two objects, some cases among many objects. The learning process uses the inference deduction to conclude the new knowledge from the relationship between objects. The new outcome may be expected from the knowledge processing unit with the finite bound based upon the previous knowledge in the knowledge database. For example, in the medical system, the relationship between two diseases on common causes or locations of occurrences can lead the knowledge processing unit to learn more about these two diseases that may turn out to be the same or similar disease based upon the current knowledge in the knowledge database. The deduction inference will conclude and summarize into the knowledge database if it has never existed. Confirmation from the experts may be needed if the new knowledge is crucial. If confirmation has not been provided, the knowledge processing unit will give a low value for the level of confidences for later applications.

However, the relationship between objects must not be infinite. Otherwise the knowledge processing unit may be infinitively learning the relationship from object to objects to objects indefinitely for which the knowledge processing unit cannot conclude for new learning knowledge.

The third level of learning process involves the external resources and actions to modify rules. When there is action(s) from an administrator of the knowledge machine,

the knowledge processing unit extracts and analyzes the action(s) to discover a reason. So the knowledge processing unit can provide this new reason into the knowledge database as a new rule or updating existing rule(s). However, if the new rule or changing of the current rules is inconsistent to the existing rules, the knowledge machine must notify the expert of the knowledge machine to clarify the conflict. If the conflict cannot be resolved, the new rule or the processing of updating the current rules must be temporarily blocked to prevent the usage of it.

The fourth level of learning process is to adjust the strategy of the solving mode. The knowledge processing unit learns new knowledge by using the inference generation from the results and solutions of the solving mode. When there are many alternative strategies to solve a problem, the knowledge processing unit adapts the strategy based upon its interest such as time of execution, cost of execution that includes the minimization of cost, and maximization of profit and revenue, the risk of execution that would be avoided, and so on. The strategy can be chosen by using the theories of decision making to execute the next step. The learning mode must save this strategy, if new, into the knowledge database, for the later execution. When the knowledge machine encounters the same problem, which has already had a strategy in the knowledge database; the knowledge machine does not need to repeat the strategy analysis again.

Chapter 4 Medical Distributed Databases

4.1 Distributed Databases

As the internet technology has enormously emerged, computers around the globe are connecting through inter-network servers. As more and more internet clients need the data from databases on the internet, the demand for advanced database technology has been increasing. Standalone database servers used to be popular earlier when networks were local inside a company. The internet technology connects the standalone servers together to let clients explore information inside the standalone database. That makes them become multiple-connected databases. When they were connected, the term of multi-connected databases had been defined such as Centralized database systems, distributed systems, parallel databases, and distributed databases.

4.1.1 Standalone Database Servers

Since early 1980's, database technology allowed for concurrent access. Many clients could connect to a database server via a server of a local area network in a company. Each client used an interface built by programmers who wrote embedded-SQL programs. The embedded programs connected to a database and allowed clients to log in to a database server. Clients can manipulate –insert, update, and delete- information from the databases.

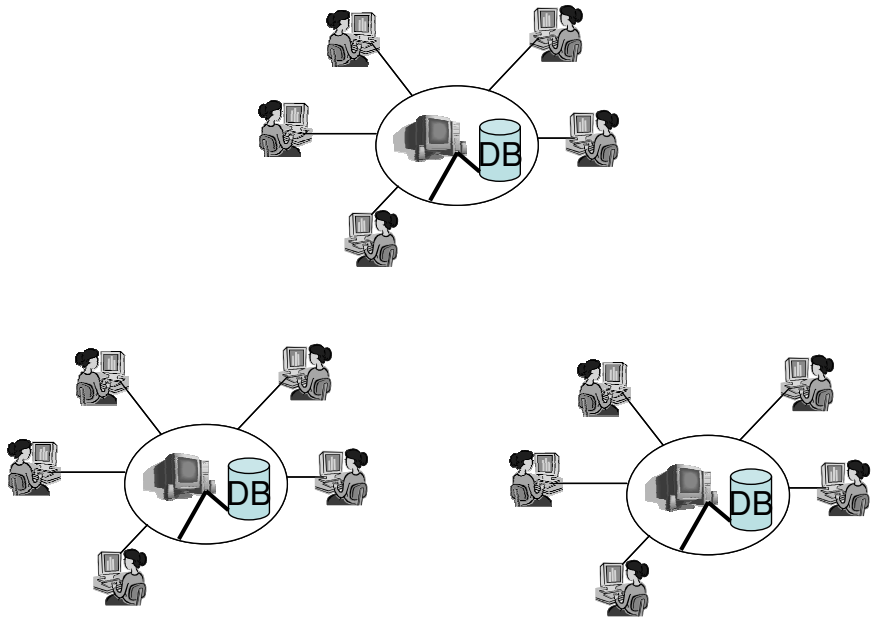


Figure 4- 1 Standalone Databases

The information can be shared if it is inside the same LAN sharing network, databases and application server. At the same time, the technology of storage was growing. The information of many departments can be stored and used together. The relational database of information among departments was design together as one database system of a company.

The expansion of a company has grown outside one location of a company. Companies had many branches. Each branch has their own LAN's and database systems. The information needed to be shared not just inside a branch but across their local networks. Also some information was stored in more than one place. That was leading to **data redundancy**. At the same time, advances in computer technology led to better processors, memory, storage, and, importantly, networks. Business demanded the connecting of all databases together or combining their databases together. So the business can share the information across branches.

4.1.2 Centralized Database Systems

When the interconnection of networks had advanced, the WAN, wire area network, technology can handle the idea of shared information across the network. The centralized database system has been deployed to the business. The centralized database gathers and stores information of all databases together. All users connect to their own local area network. They submit a query via their LAN server. The LAN server forwards the query through the WAN passing routers. The central WAN server receives a query and sends it to the database server. The database engine analyzes and retrieves information from the database and sends it backward all the way to the user who requests the information.

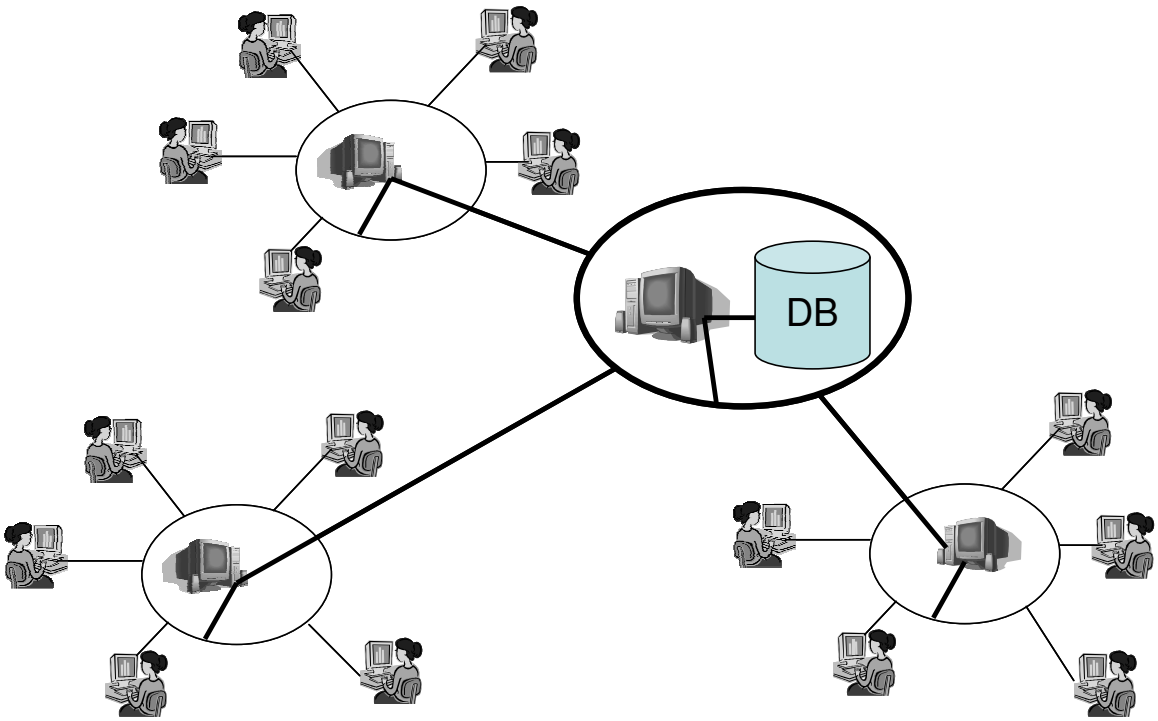


Figure 4- 2 Centralized Databases

There are advantages of using a central database system. The data redundancy is solved because there is only one central database for all users to use. All clients can share information together. The information is up-to-date. When one user makes a change to data, all others can view it on the same time. It is easy for programmers to implement an application to users because they use the same database scheme.

There are also disadvantages of using the centralize database systems. The failure of the central database brings down the whole system, using the central database. The backup must be ready and the central database must really be reliable. Even if the central database is not down, there are other disadvantages. When all users share the same database, they also share the same network that connects to the central database. The network congestion may occur because of too much traffic. A similar problem is the bottleneck at the database server. One database engine has to response to all requests from users. The database server may not be able to keep up all requests. Some request may need to be discarded or, worse, resubmitted. The result is an increase in the number of requests because the same request is submitted twice or more from the same client until the client receives a response. The transaction of concurrency access must be considered. However, it is similar to the standalone database server because two users can update information at the same time. But, for a centralized database, two users may send a transaction from a different network. The transaction must be considered globally not locally as in the standalone system.

4.1.3 Distributed Systems

Before a distributed system was explored, there were implementations of a multi-programming system, a multi-processing system, and a parallel system. A multi-programming is an old architecture of operating system to allow many programs of a user to run at the same time in the same system. Memory is divided for each program, but no sharing. When one program runs, others must wait if the system uses a uni-processor. However one program can submit only one job at a time. The multi-processing system allows a program to be divided into subprograms, called processes. In the multi-processing system, when one process of a program cannot run, another process of the same program can run. So the program does not have to give up its time–slice for the processor to other processes.

In a parallel system, there are client-server and distributed architectures. The client-server architecture has a server acting a master whose responsibility is to accept all requests from clients, process them, and send the responses back to clients. The architecture is simple and adopted by many commercial systems. The main concentration is on the server side. The system designers must guarantee the support of the reliable server and heavy traffic from all clients. The bottleneck problem still exists in the client-server architecture on the server side. Also if the server has a problem such as hang, or crash, the whole system is down. The practical solution is to use a reliable server or a mirror server. The mirror server system has two machines running on the same time. One machine acts as a server. Another machine copies all change that occurs to the server. If the server is down, the mirror will act as a server and continue the job of the dead server. When the dead server is rebooted, the dead server will become a mirror machine. The

centralized database used the client-server architecture. But they share only the information inside the database, but not files or other hardware resources at the server side.

Unlike the client-server architecture, a distributed system is a collection of connected independent computers that are transparent to users. Computers can communicate to others using the **distributed message passing**. Two main functions are needed, send and receive. The sending function specifies a destination of the message and the message content. The message is sent via a buffer. The receiving function promptly checks the buffer whether its message is in by comparing with its identifier. If so, the message is retrieved. One model of a message passing is the **remote procedural call** (RPC). Its technique is to allow programs on different machines to communicate using a procedure call and procedure return. The two programs on different machines will act like they are calling/returning a procedure from the same machine that share memory.

A distributed system is also categorized as a symmetric multiprocessor (SMP). However a symmetric multiprocessor is not a distributed system if two processors are sharing memory. Clustering system is another example of the distributed system.

4.1.4 Parallel Databases

To provide the reliability and efficiency, the parallel database system has a copy of the same information stored in many locations. The client can retrieve information from the local database as from the centralized database. There is no problem of bottleneck, network congestion at the central database, or the failure of the central database. Bottleneck does not occur because there are two or more databases to query

form clients. If the local database has lot of queries piling up, the query is forwarded to another available database. Network congestion of the central database cannot occur because there is no such central database. If failure of a local database occurs, a client can still request information from other databases. By the same time a copy of the database from any other locations can be sent to the location that has lost the database. Therefore, the parallel database system is much more reliable than the centralized database system.

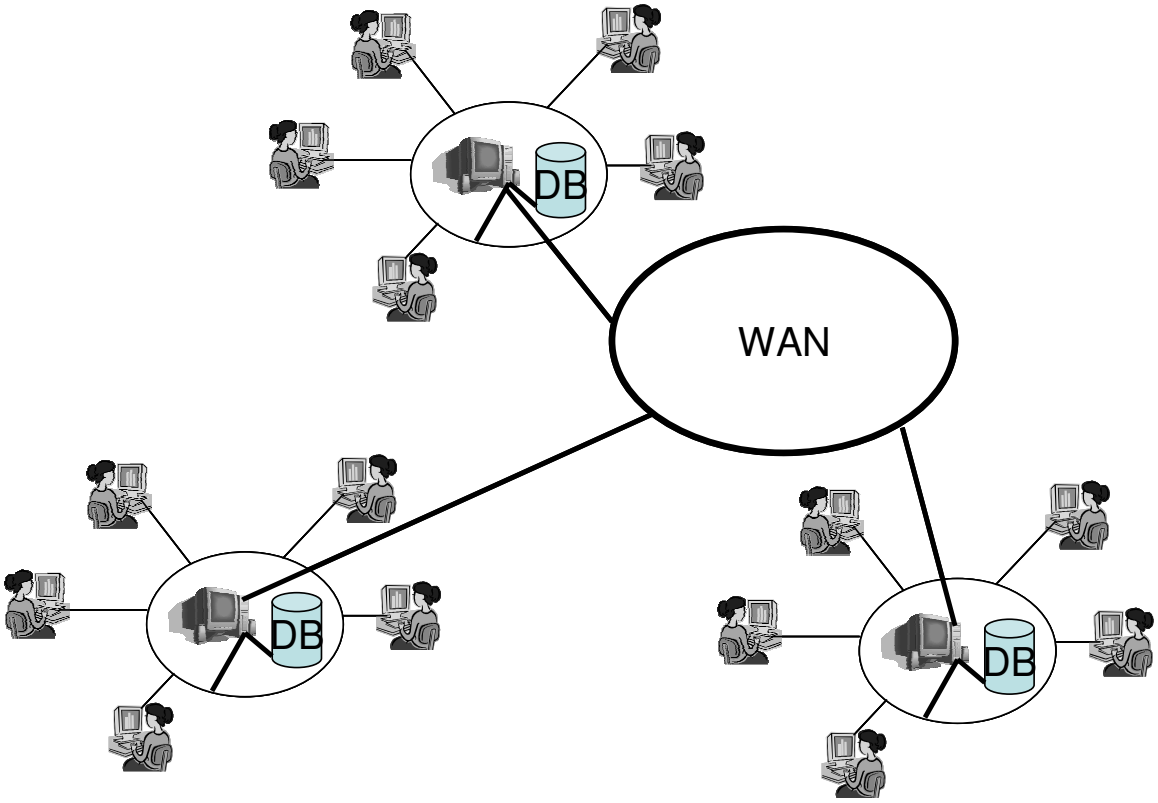


Figure 4- 3 Parallel Databases

Since a query is executed at its own database server, the query processing would be faster than sending it outside the LAN to the central database. Time of communication costs the overall time from submitting of a query to returning its result. Because the cost of technology is not so high as earlier, having a database server at each local site does not

economically cost a company, especially, comparing having many medium size servers with one high performance server.

Parallel databases, having duplicated data, also have disadvantages. The transaction management becomes more complicate. When two clients update the same object but on their own database, the same object will have different values on two different databases. Such problem is called data inconsistency. At certain time or periodically, data among different databases must be synchronized to maintain data consistency of all databases in the parallel database system. It does not need to do so in the centralized database. Another solution is to lock all other databases while one database is updating the object. The concept of serializability and two-phase locking can be used here. Both solutions cause the system to spend some time to prevent the problem. Nonetheless, it improves the overall throughput comparing with the bottleneck and network congestion problems of the centralized database system.

4.1.5 Distributed Databases

The distributed database system has multiple databases in the system as in parallel databases. The difference is that not all databases in the distributed database system have exactly the same copy of databases through out the system as in the parallel database systems.

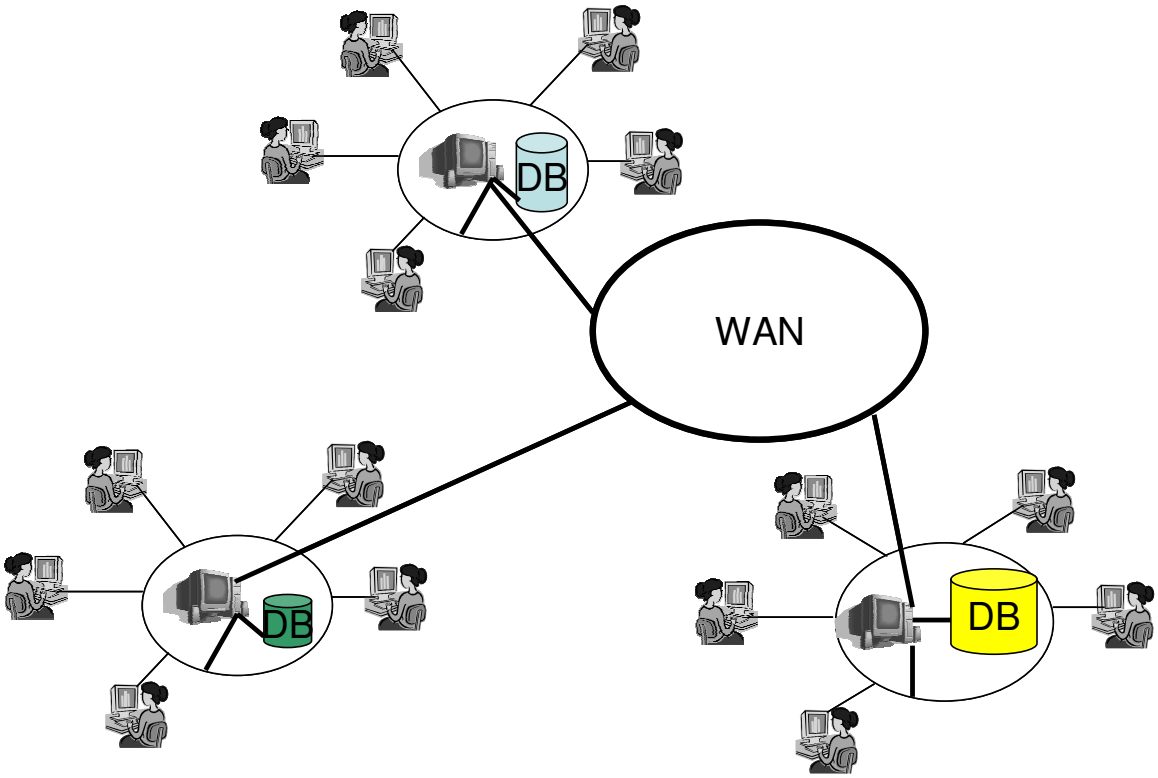


Figure 4- 4 Distribute Databases

The strategy of partitioning the database is called fragmentation. Each fragment is assigned to one site according to the frequency of usage, locality, and high reference. There is no cost of complete duplication. So the storage costs less than the parallel database. Similar to the parallel databases, the reliability and availability of distributed databases are better than the centralized databases. Also now, the communication cost is lower. If designed properly, the performance would be better.

Because of the locality, the usage of data for each location is narrow to some part. No complete information of the database is needed. That affects the performance of the database engine at each local because the amount of data is less.

The location of data is decided by where the data is needed most. So the location that needs data most, gain the fastest access. The chance of accessing across the network is less. That saves both time of accessing and the cost of network.

Because unnecessary data does not need to be stored in all locations, the access of an unauthorized user should be less.

However, there still exists some degree of duplication of the parallelism. Some data may be stored in more than one location due to the locality of access that speeds up the faster access. Then the concurrency and transaction management must be considered.

In the opposite of the parallelism, when data is disjoint, some special requesting queries may require a joining of data from different fragments across the network. It would reduce performance. But such query does not occur often.

4.2 Designing Distributed Databases

To design a distributed database, there are three considerations, data fragmentation, data replication and fragment allocation.

4.2.1 Data Fragmentation

There are three types of fragmentation: horizontal, vertical, and mixed. Horizontal fragmentation partitions data as a subset of tuples for each location. Vertical fragmentation partitions data as a subset of domains for each location. Mixed fragmentation combines both horizontal and vertical fragmentations.

4.2.2 Horizontal Fragmentation

As another dimension, the horizontal fragmentation divides a relation horizontally by rows. All databases have the same relation schemas. But the data will be fragmented depending on which rows are related to the information of its location. Some rows will be

stored in a database but not in others. The domains (or size) of a tuple in different databases are the same. The number of blocks storing data in a distributed database is smaller than in a centralized or parallel database because the number of rows is usually less than or, at most, equal. The performance of the database engine of the distributed databases at the local site will be better than using the centralized or parallel databases. **A derived horizontal fragmentation** applies the partitioning of the primary relation to secondary relations using the primary key of the primary relation with the foreign key of the secondary relation referencing to the primary relation. Using the derived horizontal fragmentation, the data of primary relation and secondary relations can be fragmented relationally along with the relationships among them.

To combine data back together among different databases as in the centralized database, union operation is used because data are separated by rows.

4.2.3 Vertical Fragmentation

The vertical fragmentation divides a relation vertically by columns. Not all sites need information of all columns. The duplication will not be completely used. Wasted space on unused columns for a particular can be saved. The performance by the database engine would be better than the centralized or parallel databases because the size of a tuple is reduced in each database of the distributed database system. Each block in storage can store more tuples. A smaller number of blocks is needed. Time required for searching the database will be less.

Two approaches are used to vertically fragment data in a database.

Grouping The database designer considers what common attributes can be used together in a particular location. At each step, the database designer joins some fragments together until it satisfies the criteria of a relation.

Splitting Just like the name, the database designer considers the original relations and chooses which columns would be needed for each location. The access of applications to attributes is the main consideration.

It is not totally distinct columns when a relation in a distribute database is vertically fragmented. The reason is because we have to be able to join back data together as in the original database. The splitting is considering only columns that are non-prime attributes. The global keys, usually primary keys of the tables before distribution, will be replicated among databases. But not all global keys are distributed to all databases. Let us consider that the vertical fragmentation of a relation, R is split into databases, D1 and D2. The primary key, PK1 of the relation R becomes a global key. It is global only in the database D1 and D2 but not in other databases that have nothing to do with the relation R.

The operation, needed to combine data among distributed databases into one set, is the joining (or Cartesian product). Because the splitting into distributed databases separates rows by columns, the join operation concatenates two rows together side by side.

4.2.4 Mixed Fragmentation

Mixed fragmentation applies both horizontal and vertical fragmentation to databases in a distributed database system. To take advantage of both vertical and horizontal fragmentations, the mixed fragmentation satisfies most applications for the

distributed database. Each site needs some columns and some rows of data, but not all.

The level of nesting is limited to only two levels, horizontal then vertical, or vertical then horizontal. To join combine data among distributed database back together, we need both join and union operations. It is at most two operations because the nesting is only two levels when splitting. First, the joining applies between two relations. So tuples in different relations will be compatible (the same number of columns and the same domain between $column_i$ of relation R1 and $column_i$ of relation R2.) Then after they are compatible, we can combine them together using union operation.

4.2.5 Data Replication and Fragment Allocation

Assigning each fragment to store at which particular site is called data allocation. To improve the performance of the database engine, some data may be stored in two or more sites. This data redundancy is called data replication.

If two databases have a complete copy of another, it is called fully replication. The fully replication to among databases can improve the availability. If the database at a site is down, another (or other) database is available for the user whose database is currently failed. Also the performance of query retrieval at each site is improved because each site can find information at its database.

As in a parallel database, the costs of updating data will be high. Once one database updates data, another (or other) replicated database must be informed. That makes the transaction, concurrency control and recovery technique becoming more expensive. Therefore, no replication is considered. All databases become disjoint. But then it will lose the improvement of availability and retrieval performance. To take

advantages of the replication, the distributed databases use a degree of partial replication. Some fragment may be replicated among databases whereas others may not.

Both criteria of data allocation and data replication must be considered together when designing the distributed databases. More number of choices of sites that we choose for, more reliability. However, it is more complication on the transaction management. It is depending on the goal of the system, and the complication of the transaction submitted for each query from a site. For example, if the query is mostly retrieval, not updating, the high level of replication among different locations can be used because reading of the same object on two different sites is not a conflict between transactions of two submissions. If the updating is heavily operated, the fully replication should not be used. Even partial replication must be limited. However, if updating on a fragment occurs in one particular site only, that site must hold that fragment. Other sites can have a partial or full replication.

4.3 Query Processing of Distributed Databases

4.3.1 Query Processing

The main function of a relational database is to process queries correctly and efficiently. The correctness and efficiency of a query is how to process query from many strategies. For example, choosing a Cartesian product instead of natural join, if possible, costs more time and resources in term of efficiency. The detail is described in chapter 2 for the query processing in a relational database. For the distributed databases whose data are not physically stored in the same location, the correctness is depending on its design such as data fragmentation and data allocation and replication. The main concern when dealing with distribution is the efficiency of data processing when the networking is involved. The time and other resources consumed on the network are different when one query is processed using two different strategies on the distributed databases.

Beside the network load of query execution in distributed databases, the query transformation from a SQL query into relational algebra expressions is not simple as in the centralized database. In centralized database, all data is stored and executed in the same database engine. One user query can be transformed into an equivalent relational algebra expression. But in the distributed database, one user query may need to be broken down into subqueries and the subqueries executed in different sites. Data must be exchanged among sites. Both the order of query operation and the selection of site to execute a subquery must be well chosen.

The following is an example of comparison between execution by multiple sites and single site execution.

Suppose a doctor wants to find out who had a severe headache on Oct/01/04 and which clinic the patient visited. The doctor executes the following query

```
Select  patient.pname,  patientsymptom.clinic
From patient, patientsymptom
Where patient.pid = patientsymptom.pid
        And patientsymptom.date = '2004-10-01'
        And symptom = 'severe fever'
```

One promise of a distributed database is transparency. The query is submit by a user (doctor) as a query to one database engine without worrying that there are multiple databases distributed into many locations.

The equivalent relational algebra of the above query is

$$\pi_{\text{pname, clinic}} (\text{patient} \bowtie_{\text{pid}} (\sigma_{\text{date = '2004-10-01'} \wedge \text{symptom = 'severe fever'}} (\text{patientsymptom})))$$

Suppose the databases are distributed using the horizontal fragmentation as

$$\text{PATIENT1} = \sigma_{\text{pid} < 50000} (\text{patient})$$

$$\text{PATIENT2} = \sigma_{\text{pid} >= 50000} (\text{patient})$$

$$\text{CLINICGROUP1} = \sigma_{\text{pid} < 50000} (\text{patientsymptom})$$

$$\text{CLINICGROUP2} = \sigma_{\text{pid} >= 50000} (\text{patientsymptom})$$

One of strategies to execute the above query on the distributed databases is to let each site execute its subquery regarding the horizontal fragmentation and then the site of user unions the result together. For example, suppose sites A and B maintain information

of patientsymptom table according CLINICGROUP1 and CLINICGROUP2 respectively. Site A finds ID's of patient who visits a doctor on 10/01/04 with severe fever and has id less than 50000. The same is applied to site B but only the id that is not less than 50000. Suppose sites C and D maintain information of patient table according to PATIENT1 and PATIENT2 respectively. The result produced by sites A and B will be joined to find name of patients at sites C and D respectively. After sites C and D produce the patient names, site, E, who originally submits the query, can combine those two sets of results together using union operation as shown in the following Figure 4- 5.

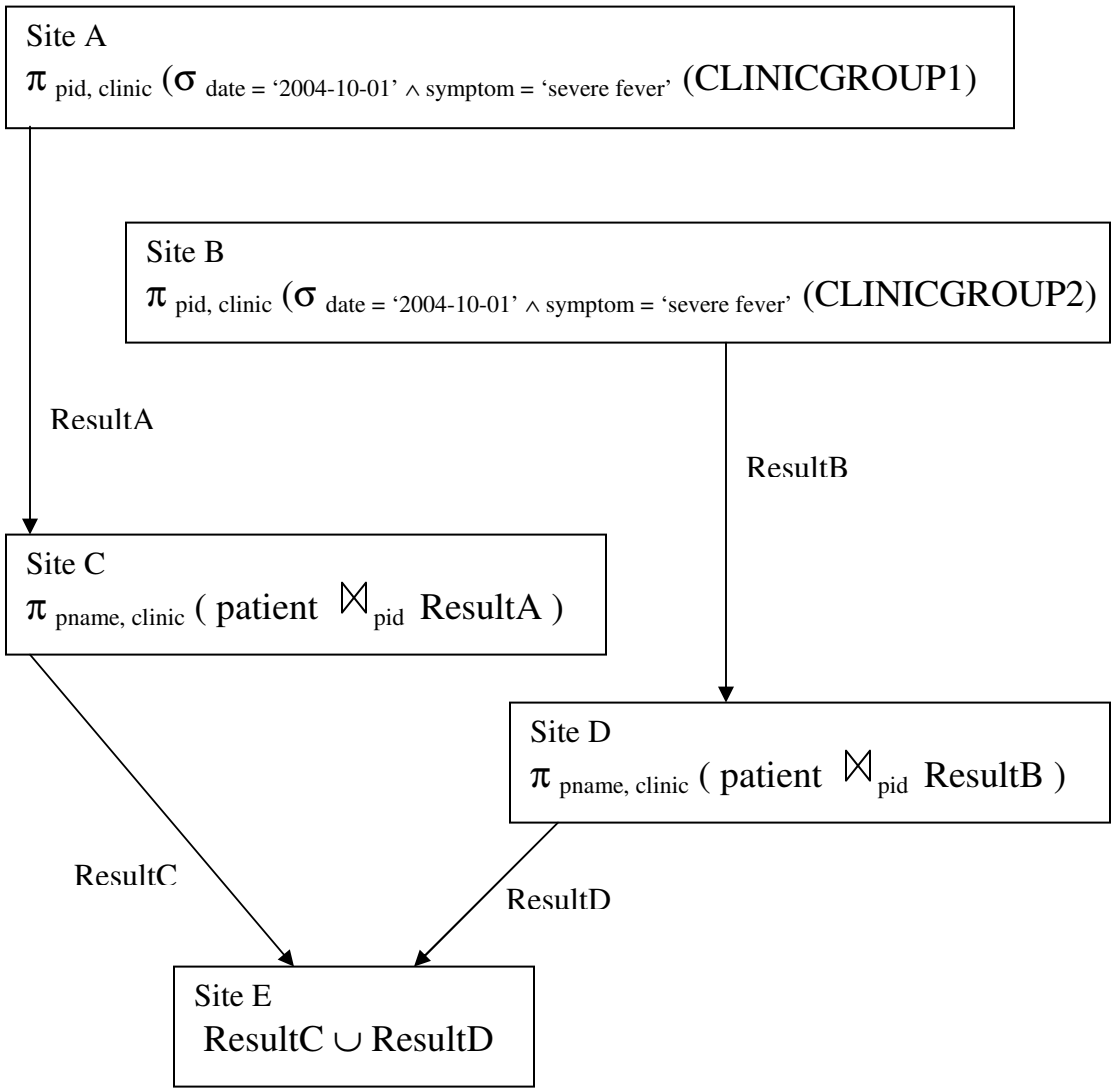


Figure 4- 5 Combining Results from Distribute Database by Union Operation

Another strategy is simpler than the previous strategy. Sites A, B, C and D process and transfer information regardless of the user's query. Sites A, and B execute the query CLINICGROUP1 and CLINICGROUP2, respectively, to submit all information of its PATIENTSYMPTOM to site E. Similar to sites C and D, all information of their PATIENT is submitted to site E regardless the result of sites A and B. Site E will process the results from sites A, B, C, and D together according to the user query.

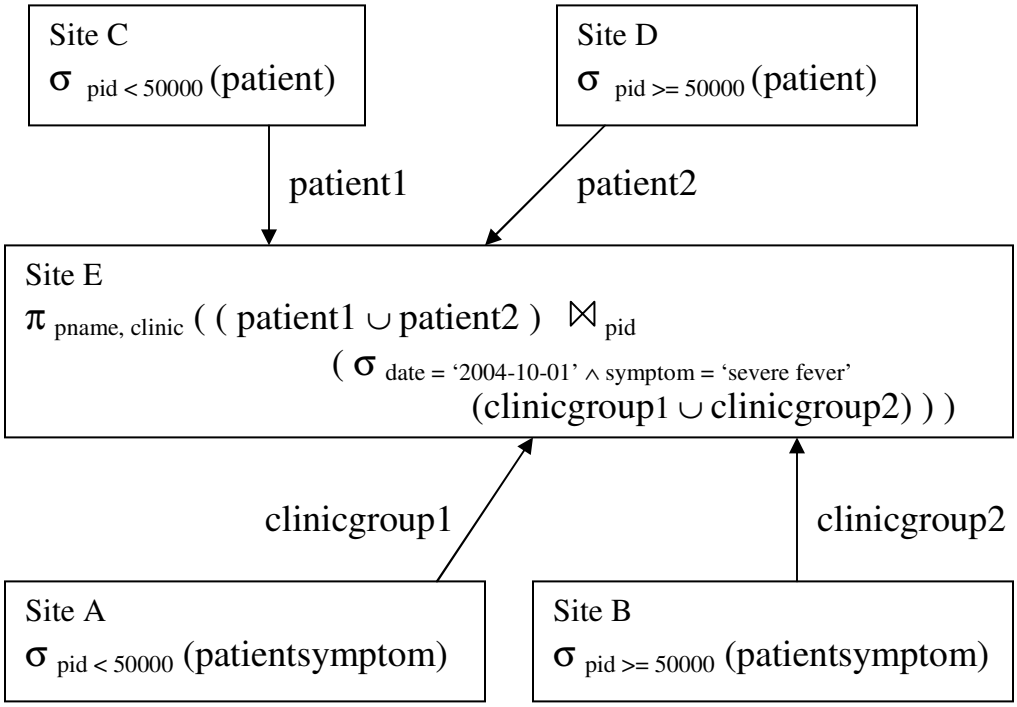


Figure 4- 6 Combining Results among uncoordinated Distributed Databases

The second strategy is simpler because there is no co-ordination among sites. All sites A, B, C, and D send their data to site E to process. However, the transmission cost of the second strategy is much higher.

Suppose there are 2,000 and 1500 patient records for sites C and D and there are 1000 and 500 patient records with symptoms for each sites A and B. Each record of

patient in sites C and D is 200 bytes. Each record of patient-symptoms in sites A and B is 50 bytes. The size of patient id, name, and clinic are 10, 30, and 20 bytes. Assuming that there 20 and 30 records of patients who have severe fever on Oct/01/2004 on sites A and B.

The cost of the second strategy can be summation of:

1) Sending patient records from site C to site E is 2000 * 200 = 400,000 bytes

2) Sending patient records from site D to site E is 1500 * 200 = 300,000 bytes

3) Sending patient-symptom record from site A to site E is 1000 * 50 = 50,000 bytes

4) Sending patient-symptom record from site B to site E is 500 * 50 = 25,000 bytes

The total cost of the second strategy is = 775,000 bytes

The cost of the first strategy can be summation of:

1) Transferring pid and clinic of patients who have severe fever on Oct/01/2004 from site A to site C is 20 * (10 + 20) = 600 bytes

2) Transferring pid and clinic of patients who have severe fever on Oct/01/2004 from site B to site D is 30 * (10 + 20) = 900 bytes

3) Transferring pname and clinic from site C to E 20 * (30 + 20) = 1,000 bytes

4) Transferring pname and clinic from site D to E 30 * (30 + 20) = 1,500 bytes

The total cost of the first strategy is = 3,000 bytes

The total cost of transmission of the second strategy is about 250 times of the first strategy. It is unacceptable. However, we can reduce the cost of the first strategy by not sending a whole record, as the third strategy. We can send only the columns that are

needed for the query. We can send only id and name of patients from sites C and D to site E and only patient-ids and clinic name from sites A, and B to site E.

The cost of the third strategy can be summation of:

1) Sending patient records from site C to site E is 2000 * (10+30) = 80,000 bytes

2) Sending patient records from site D to site E is 1500 * (10+30) = 60,000 bytes

3) Sending patient-symptom record from site A to site E is 1000 * (10+20) = 30,000 bytes

4) Sending patient-symptom record from site B to site E is 500 * (10+20) = 15,000 bytes

The total cost of the second strategy is = 185,000 bytes

The cost of the third strategy is three times less than of the first strategy but is still sixty times more than of the second strategy. However, the transmissions of the first and third strategies can be done in parallel from four sites A, B, C and D to site E. The transmission of the second strategy can be parallel between sites A and B but the sites C and D have to await the results from sites A and B to transmit to site E.

Suppose the transmission rate is 1k/ms. The waiting time at site E before processing the query for the first and third strategy is the maximum time that simultaneously transmits from all sites to site E. For the first strategy, the waiting time at site E is the transmission time from site C to site E. It is 400,000/1k that is about 400 ms. The waiting time before processing the query of the third strategy is 80,000/1k that is about 80 ms. For the second strategy, two halves of waiting time are required. The first half is the maximum time between transmitting from site A to site C and from site B to site D. The second half is the maximum time between transmitting from site C to site E and from site D to site E. Therefore the waiting time at site E before processing the query is 900/1k + 1500/1k. It is about 1+1.5 = 2.5 ms. Even though the waiting time of the

second strategy is in two steps of pipelines, it is still much smaller than of the first and third strategies.

Another set of waiting times that can be considered in the second strategy is the processing time of the query at sites C and D before sending to site E. However, there are two reasons that we omit it here. The CPU speed for processing a query is much faster than the transmission rate. Also the processing time at site E of the first and third strategy is more than the processing time at site E of the second strategy because the number of records that will be processed at site E by the second strategy is much smaller than of the first and third strategy.

There are other considerations when processing a distributed query such as statistics, decision sites, and network architectures.

4.3.2 Statistics

To minimize the cost of transmission, statistics of prior query processing must be accounted for in future query processing. The statistics includes choosing which operation should be processed first because the data are fragmented depending on the size and the number of distinct values of each attributes. The detail of data fragmentation is explained earlier in this chapter. The selection of some entities, projection of some attributes, or combination of both cases provides different time of transmission. However, collection of the statistics of query processing costs the management to the system every time before query is terminated. It also costs the data manipulation more time and resources. For example, when the data needs to be updated, the system must collect what has been changed because after updating, the same query, executed earlier, may be

executed more efficiently by a different strategy. To reduce the cost of collection of statistics, a periodic updating of statistics can be used instead of updating statistics every time when data is manipulated.

4.3.3 Decision Sites

When a query is submitted from a user site, which sites should make a decision for the query optimization? A simple approach is to let the central site, which has statistics, database schema, and all knowledge of databases, making a decision to choose the best strategy to process. It is simple because only the central site has to maintain the statistics and all knowledge about databases. The local sites do not need to maintain such information. However, the bottleneck problem may occur at the central site. In the opposite, if we let the local sites make the decision all local sites must continuously update statistics even though it may not be used in near future. This costs both in traffic and resources of all local sites. The hybrid approach was introduced. It combines both central and local decision making. Only the major decision is made by the central site.

4.3.4 Network Architectures

The cost and execution strategy is depending on the network architecture. On the wide area network, the communication cost dominates the consideration. The selection of global execution strategy bases on the inter-site communication. For the selection of the local execution strategy bases on the centralized query optimization algorithm. The knowledge banks are linked via a high speed network. Isolated packets of information may arrive from numerous Integrated Medical Systems. The identifier of the knowledge

bank must be consistent in the information stored in that particular bank, so the switching time to these massive information storages can be reduced. For the local area network, the communication cost is comparable to I/O cost. Our main consideration for the best strategy is to increase the parallelism of query execution among the distributed databases. A transaction between a single Integrated Medical System and knowledge banks are symmetrically processed. The output can be accumulated from sub-procedures to the main procedure.

4.4 Four Levels of Query Processing in Distributed Databases

We break down the query processing in the distributed databases in four levels: query decomposition, data localization, global optimization and local optimization. A query, submitted from a user, must be decomposed, syntactically check with a global schema, and rewrite it at the level of query decomposition. After that the query will be broken down using the fragmentation schema at the data localization level. Before sending fragmented queries to local distributed databases, the global optimization level uses statistics on fragments to optimize the fragment queries. The global optimized queries with minimum communication cost are sent to the local sites to locally optimize and process.

4.4.1 Query Decomposition

Query decomposition is the first layer of query processing that transforms a complex query into a semantically correct query and no-redundancy work for query processing. Using global schema, there are four steps –normalization, analysis of query

correctness, redundancy elimination, and rewriting – to decompose a user query into algebraic query distributed relations.

- Normalization deals with the predicate of the query that may be complexly written by users. In SQL, the predicate is in the WHERE clause. The connectives AND and OR can be any combinations. The transformation is to normalize it into two possible normal forms, conjunctive normal form and disjunctive normal form.

The conjunctive normal form is a conjunction of disjunction as in following form:

$(p_{a1} \vee p_{a2} \vee \ldots \vee p_{ai}) \wedge (p_{b1} \vee p_{b2} \vee \ldots \vee p_{bj}) \wedge (p_{z1} \vee p_{z2} \vee \ldots \vee p_{zk})$

where each p_{mn} is a simple predicate.

The disjunctive normal form is a disjunction of conjunction as in following form:

$(p_{a1} \wedge p_{a2} \wedge \ldots \wedge p_{ai}) \vee (p_{b1} \wedge p_{b2} \wedge \ldots \wedge p_{bj}) \vee (p_{z1} \wedge p_{z2} \wedge \ldots \wedge p_{zk})$

where each p_{mn} is a simple predicate.

Using the following equivalency laws, a complex predicate can be transformed into one of two normal forms

1) Commutative	$P \wedge Q \equiv Q \wedge P,$	$P \vee Q \equiv Q \vee P$
2) Associative	$P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R,$	$P \vee (Q \vee R) \equiv (P \vee Q) \vee R$
3) Distributive	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R),$	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
4) De Morgan's Law	$\neg (P \wedge Q) \equiv \neg P \vee \neg Q,$	$\neg (P \vee Q) \equiv \neg P \vee \neg Q$
5) Double Negation	$\neg (\neg P) \equiv P$	
6) Inverse	$P \wedge \neg P \equiv F_0,$	$P \vee \neg P \equiv T_0$
7) Domination	$P \wedge F_0 \equiv F_0,$	$P \vee T_0 \equiv T_0$
8) Idempotent	$P \wedge P \equiv P,$	$P \vee P \equiv P$
9) Identity	$P \wedge T_0 \equiv P,$	$P \vee F_0 \equiv P$

10) Absorption $P \wedge (P \vee Q) \equiv P$, $P \vee (P \wedge Q) \equiv P$

For example, consider the following query:

Select patient.pname, patientsymptom.clinic

From patient, patientsymptom

Where patient.pid = patientsymptom.pid

And patientsymptom.date = '2004-10-01'

And symptom = 'severe fever' or symptom = 'vomit'

The WHERE clause can be normalized into a conjunctive normal form as

(patient.pid = patientsymptom.pid

And patientsymptom.date = '2004-10-01'

And symptom = 'severe fever')

Or

(patient.pid = patientsymptom.pid

And patientsymptom.date = '2004-10-01'

And symptom = 'vomit')

One comment is that a redundancy work may occur but it will be eliminated at the third step.

- Analysis of Query Correctness is a step to prevent an error that may occur. There are two kinds of incorrect query, typo or semantically incorrect. The typo is an unintentional error by a user who submitted the query. It can be detected by checking with the data dictionary that maintains information about all objects in databases such as tables name, columns name, and etc. After the typo is detected, the query is stopped processing and error message about the error object is returned to the user.

Semantic incorrectness may occur and produce a spurious output. Joining predicate is most common incorrectness by user who is unaware of joining condition among tables in the FROM clause. It causes a database engine to produce unnecessary, usually wrong, output. The error can be detected by using a graph. Each node of the graph is a relation. Each edge is either a join condition between tables, conditions on the node (relation), or an output from a node to another node. The incorrectness occurs if there is an unconnected node left on the joining-graph (not the query graph) of query. For example, consider the following query:

Select patient.pname, patientsymptom.clinic, disease.dname

From patient, patientsymptom, diseasesymptom, disease

Where patient.pid = patientsymptom.pid

And patientsymptom.sid = diseasesymptom.sid

And patientsymptom.date = '2004-10-01'

And symptom = 'severe fever'

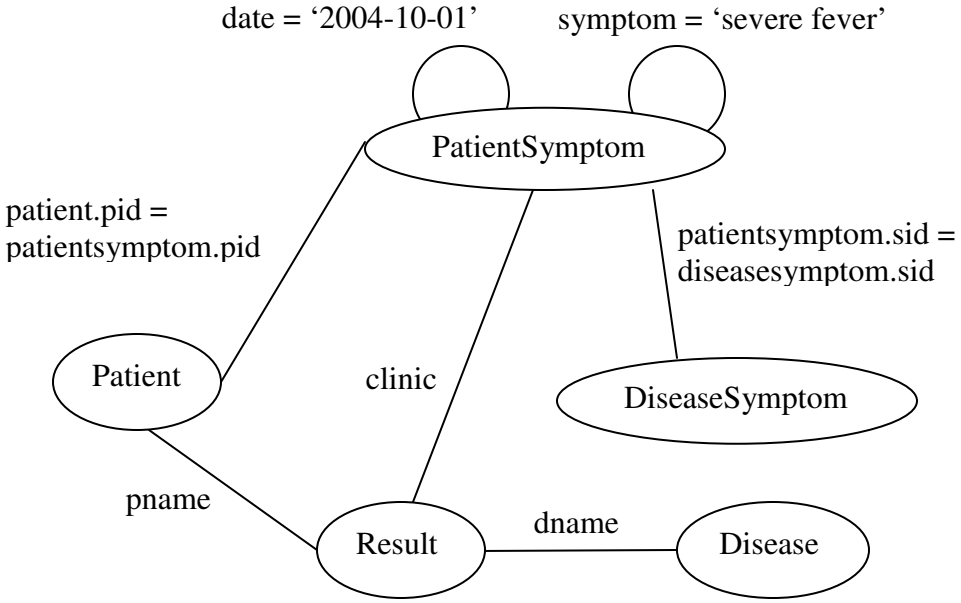


Figure 4- 7 Query Graph

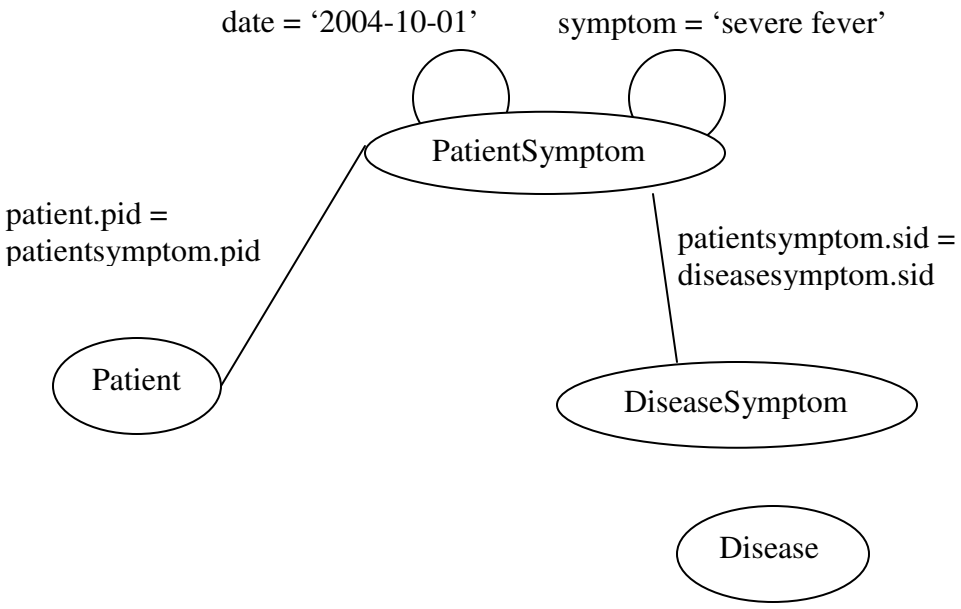


Figure 4- 8 Joining Graph without Results

The node disease is left unconnected. Therefore this query seems to be incorrect. The solutions when the incorrectness in the joining is detected are

1) Stop processing the query and notify the user.

2) Let it continue by assuming that the user intended to do the Cartesian product, not join between two tables, Disease and DiseaseSymptom for this example.

3) Solve the joining incorrectness by adding the joining predication, regarding the global schema, to the query. For this example, the predicate "disease.did = diseasesymptom.did" is added to the query at the analysis step of the decomposition layer.

- Redundancy Elimination After the correctness was solved, the redundancy of unnecessary work to a database engine on non-simplify predicate should be eliminated by using the law of equivalency, mentioned earlier, and the following deductive rules.

1) $A \theta B \wedge B \theta C \Rightarrow A \theta C$ where θ is =, >, <, >=, or <=.

2) $A \theta B \wedge B = C \Rightarrow A \theta C$ where θ is >, <, >=, or <=.

3) $A \theta B \Leftrightarrow B \alpha A$

where θ is > or >= and α is < or <= respectively.

For example, consider the following query:

Select patient.pname, patientsymptom.clinic, disease.dname

From patient, patientsymptom, patientcause

Where patient.pid = patientsymptom.pid

And patientsymptom.pid = patientcause.pid

And patient.pid = patientcause.pid

And (symptom = 'severe fever' or symptom != 'severe fever')

Let patient.pid be A, patientsymptom.pid be B, patientcause.pid be C and symptom = 'sever fever' be predicate P. The query condition is

$$A = B \wedge B = C \wedge A = C \wedge (P \vee \neg P)$$

Using inverse law: $P \vee \neg P \equiv T_0$

$$A = B \wedge B = C \wedge A = C \wedge T_0$$

Using Identity law, $P \wedge T_0 \equiv P$

$$A = B \wedge B = C \wedge A = C$$

Using rule 1), $A = B \wedge B = C \Rightarrow A = C$, we can eliminate the predicate A = C.

$$A = B \wedge B = C$$

Therefore the redundant query can be eliminated as:

Select patient.pname, patientsymptom.clinic, disease.dname

From patient, patientsymptom, patientcause

Where patient.pid = patientsymptom.pid

And patientsymptom.pid = patientcause.pid

- **Rewriting** To reduce the cost of query processing and communication, we construct a query as a tree and then apply the transformation rules to reconstruct an equivalent tree that has the least predicted cost.

From a SQL (or relational algebra), a query tree can be constructed by:

- Each relation in the FROM clause is a leaf of tree.
- Joining of each pair of relations is a parent of the relational leaf.
- After joining all the relation, each simple condition in the WHERE clause is a node uprising one after another of each simple condition.
- The root of tree is an expression list of projection in the SELECT clause.

For example, consider the following query:

Select patient.pname, patientsymptom.clinic, disease.dname

From patient, patientsymptom, diseasesymptom, disease

Where patient.pid = patientsymptom.pid

And patientsymptom.sid = diseasesymptom.sid

And diseasesymptom.did = disease.did

And patientsymptom.date = '2004-10-01'

And symptom = 'severe fever'

Using the above method, we can construct a query tree as:

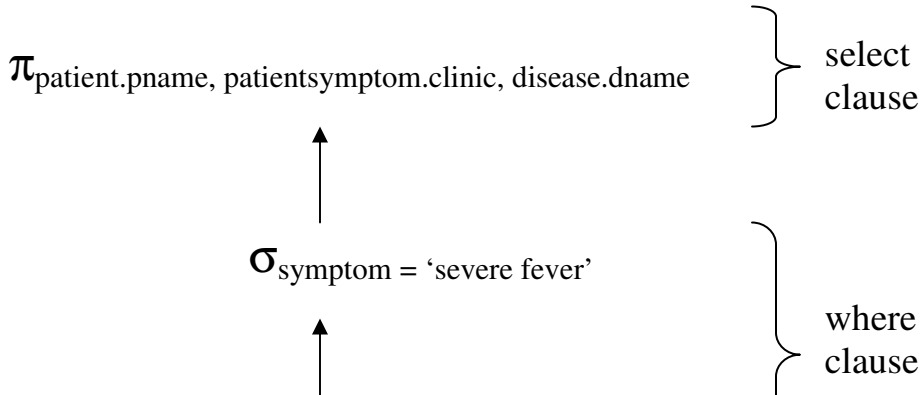


Figure 4- 9 Constructed Query Tree from SQL

The transformation rules can apply to reconstruct an equivalent query tree to reduce cost of query processing among distributed databases. The most useful rules are:

1) Commutativity of Cartesian product (including join operation)

$R \times S \Leftrightarrow S \times R$

$R \bowtie S \Leftrightarrow S \bowtie R$

2) Associativity of Cartesian product (including join operation)

$(R \times S) \times T \Leftrightarrow R \times (S \times T)$

$(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)$

3) Idempotence of Projection and Selection

$\pi_A (\pi_B (R)) \Leftrightarrow \pi_A (R)$

where A and B are sets of attributes and $A \subseteq B$.

$\sigma_{P1(A)} (\sigma_{P2(B)} (R)) \Leftrightarrow \sigma_{P1(A) \wedge P2(B)} (R)$

where P1 and P2 are predicates and A and B are sets of attributes using in P1 and P2 respectively.

4) Commuting Selection with Projection

$\pi_A (\sigma_{P(B)} (R)) \Leftrightarrow \pi_A (\sigma_{P(B)} (\pi_C (R)))$

where P is a predicate, A and B are sets of attributes, B is using in P, and $A \subseteq C$ and $B \subseteq C$.

5) Commuting Selection with Cartesian product or Set operations

$\sigma_{P(A)} (R \times S) \Leftrightarrow (\sigma_{P(A)} (R)) \times S$

where P is a predicate, and A is a set of attributes on the relation R and is using in P.

$\sigma_{P(A)} (R \cup S) \Leftrightarrow (\sigma_{P(A)} (R)) \cup (\sigma_{P(A)} (S))$

$\sigma_{P(A)} (R \cap S) \Leftrightarrow (\sigma_{P(A)} (R)) \cap (\sigma_{P(A)} (S))$

$\sigma_{P(A)} (R - S) \Leftrightarrow (\sigma_{P(A)} (R)) - (\sigma_{P(A)} (S))$

where P is a predicate, and A is a set of attributes on both relations R and S that are compatible.

6) Commuting Projection with Cartesian product or Set operations

$\pi_A (R \times S) \Leftrightarrow \pi_B (R) \times \pi_C (S)$

where P is a predicate, B, and C are sets of attributes on the relations R and S respectively, and $A \subseteq B \cup C$.

$$\pi_A (R \cup S) \Leftrightarrow (\pi_A (R)) \cup (\pi_A (S))$$

$$\pi_A (R \cap S) \Leftrightarrow (\pi_A (R)) \cap (\pi_A (S))$$

$$\pi_A (R - S) \Leftrightarrow (\pi_A (R)) - (\pi_A (S))$$

where A is a set of attributes on both relations R and S that are compatible.

Using the transformation rules, a query tree can be reconstructed in many equivalent trees. Some of trees may decrease or increase of cost of query execution. In a distributed database, we should use the Commuting Selection with Cartesian product and then the Commuting Selection with Projection to reduce the cost of communication among relations across networks. After each individual relation reduces the number of tuples, the Commuting Projection with Cartesian product can be used to join them. The tree of the previous example can be reconstructed as:

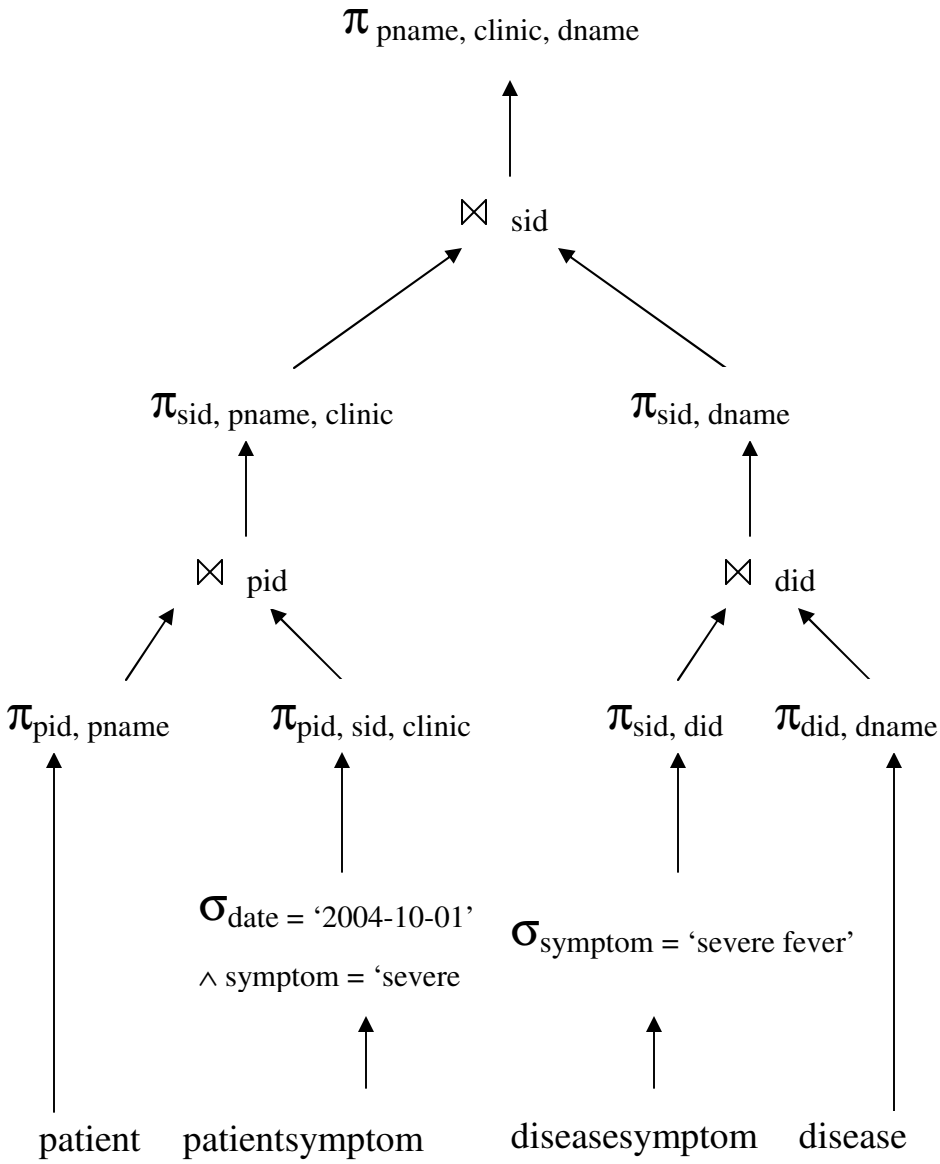


Figure 4- 10 Reconstructed Query Tree using Transformation Rules

4.4.2 Data Localization

In the previous section, we presented four techniques of query decomposition: by normalization, analysis of query correctness, redundancy elimination, and rewriting. These four techniques can be applied at the central site if it is used in the centralized system. They can be applied at the user site in the distributed system that maintains a

global schema. In this section, we shall consider how a query should be executed based on the fragmentation. As describe in section 4.2, there are three types of data fragmentation: horizontal, vertical, and mixed fragmentations. For each type, we present reduction techniques that simplify and optimize queries.

- Reduction Techniques for Horizontal Fragmentation

The data that was horizontally fragmented are divided by records. One set of records is in a database. While other sets of records are stored in other databases. To combine all records together, UNION operation is used.

For example, a set of illness knowledge is horizontally fragmented in different databases based on the common illness that occurs in each region. Suppose the disease knowledge is localized by the following predicates:

$$DIS1 = \sigma_{did <= 100} (disease)$$

$$DIS2 = \sigma_{did > 100 \text{ and } did <= 200} (disease)$$

$$DIS3 = \sigma_{did > 200} (disease)$$

A generic query to retrieve data from DISEASE will replace DISEASE by

$$DISEASE = DIS1 \cup DIS2 \cup DIS3$$

The reduction techniques are considered differently in selection and join.

The reduction of selection considers the contradiction between the predicates of user query and the predicates of the fragmentation. The retrieval records would be empty in cases of contradiction. Consider the following query:

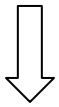
Select *

From disease

Where did = 240

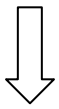
From the generic query, disease is a union of all three fragments. The generic query must retrieve from DIS1 and DIS2 that will return an empty relation because of the contradiction with the predicate “did=240”. Therefore we can reduce the generic query from retrieving three relations to only one relation, DIS3.

$$\sigma_{did=240}\ (\ DISEASE\ )$$



Query Generalized by fragmentation

$$\sigma_{did=240}\ (\ DIS1\ \cup\ DIS2\ \cup\ DIS3\ )$$



Query Reduction by contradiction

$$\sigma_{did=240}\ (\ DIS3\ )$$

The reduction of join considers the join attribute with the attributes in the predicates of fragmentation. The contradiction on join attributes and attributes of fragmentation return useless records that should be not retrieved.

Suppose the symptom knowledge is localized by

DISSYMP1 = $\sigma_{did <= 100}$ (diseasesymptom)

DISSYMP2 = $\sigma_{did > 100}$ (diseasesymptom)

If a query uses join operation between DISEASE and DISEASESYMPTOM relations as

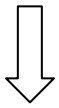
Select *

From disease, diseasesymptom

Where disease.did = diseasesymptom.did

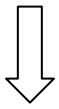
Both DISEASE and DISEASESYMPTOM must be generalized using UNION of all their fragments. However it is useless join the whole results of UNION operations such as joining between DIS1 and DISSYMP2 according to their predicates of fragmentation. DIS1 should join only to DISSYMP1. DIS2 and DIS3 should join only to DISSYMP2, not DISSYMP1.

$$DISEASE \bowtie_{did} DISEASESYMPTOM$$



Query Generalized by fragmentation

$$(\ DIS1 \cup DIS2 \cup DIS3\ ) \bowtie_{did} (\ DISSYMP1 \cup DISSYMP2\ )$$



Query Reduction by contradiction

$$(\ DIS1 \bowtie_{did} DISSYMP1\ )$$

$$\cup\ (\ DIS2 \bowtie_{did} DISSYMP2\ )$$

$$\cup\ (\ DIS3 \bowtie_{did} DISSYMP2\ )$$

- Reduction Techniques for Vertical Fragmentation

The data that was vertically fragmented are divided by columns. All records are stored in every database but not all columns. Some columns are in a database while

others are in another (or other databases) by using projection operation. To combine a record of columns in different databases together, join operation is used.

For example, a set of patient information is vertically fragmented in different databases. One database stores id, name and address of patients and another database stores id and telephone number of patients. (For simplicity, only some columns are shown.) The patient information is localized by the following predicates:

$$PAT1 = \pi_{id,\ name,\ address}\ (\ patient\ )$$

$$PAT2 = \pi_{id,\ tel}\ (\ patient\ )$$

A generic query to retrieve any columns from PATIENT will join the PAT1 and PAT2 together:

$$PATIENT = PAT1 \bowtie_{id} PAT2$$

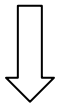
Queries can be reduced by determining the useless attributes. Fragmented relations, containing the useless attributes, must not be joined as in the generic query. Consider the following query:

Select name, address

From patient

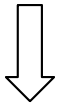
Using the generic query, patient will be replaced by the joining of PAT1 and PAT2. However the columns in the PAT2 are useless attributes. They must be reduced. Therefore the PATIENT from the user query will be replaced by only the PAT1.

$\pi_{name, address}$ (PATIENT)



Query Generalized by fragmentation

$\pi_{name, address}$ (PAT1 $\bowtie_{id}$ PAT2)



Query Reduction by contradiction

$\pi_{name, address}$ (PAT1)

- Reduction Techniques for Mixed Fragmentation

A mixed fragmentation is a combination of both horizontal and vertical fragmentations. Both record and column are split using selection, join, and projection operations. The reduction involves union, and join fragments.

For example, a set of patient information is mixed fragmented in different databases by the following predicates:

PAT1 = $\sigma_{id <= 100}$ ($\pi_{id, name, address}$ (patient))

PAT2 = $\sigma_{did > 100}$ ($\pi_{id, name, address}$ (patient))

PAT3 = $\pi_{id, tel}$ (patient)

A generic query to retrieve any columns from PATIENT will first union PAT1 and PAT2 together and then join the result with PAT3

PATIENT = (PAT1 $\cup$ PAT2) $\bowtie_{id}$ PAT3

Queries can be reduced using the following rules:

1. The contradiction on join attributes and attributes of fragmentation return useless records that should be not retrieved.

2. Determine the useless attributes. Fragmented relations, containing the useless attributes, must not be joined as in the generic query.

3. If join is still needed, distribute join over the union.

Consider the following query:

Select name, address

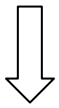
From patient

Where id = 150

Using the generic query, patient must be replaced by the joining of PAT3 with the result of union between PAT1 and PAT2. To reduce the cost of operation, the first and second rules can apply to it. Considering the WHERE clause, rule 1 can be used to for the contradiction of patient id. The PAT1 contains contradiction. Rule 2 is applied with the SELECT clause to get rid of the useless attribute that is in the PAT3.

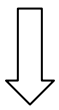
Therefore the PATIENT from the user query will be replaced by only the PAT2.

$$\pi_{\text{name, address}} (\sigma_{\text{id = 150}} (\text{PATIENT}))$$



Query Generalized by fragmentation

$$\pi_{\text{name, address}} (\sigma_{\text{id = 150}} ((\text{PAT1} \cup \text{PAT2}) \bowtie_{\text{id}} \text{PAT3}))$$



Query Reduction by contradiction

$$\pi_{\text{name, address}} (\sigma_{\text{id = 150}} (\text{PAT2}))$$

4.4.3 Global Optimization

After the data localization layer, the query is optimized by eliminating the redundant expressions, and reducing relations containing contradiction or useless attributes. With global optimization, the communication cost must be considered with various strategies as mention in the earlier section. Global optimization is to find the best ordering of operations in the fragment query in order to minimize the cost function. The main cost function is the communication operation. It also includes the cost of resources such as memory, I/O spaces, and CPU cost. Generally it is the cost of three basic elements – CPU, memory and communication. The cost of communication is the most because it is depending on the distance among the databases distributed. However, the technology of communication network has improved significantly both speed and bandwidth. It looks like we do not need to consider it. Nevertheless, the databases have also grown in size. The join ordering still needs to be considered such as semijoin.

4.4.4 Local Optimization

Optimization is a process of finding an execution strategy to minimize the distributed cost function. In general, the cost of distributed execution can be expressed as the total time and turnaround time. The total time is summation of all times. But the turnaround time is a time from starting the process until completion as parallel execution. The general formula of the total time is:

Total_time =

T_{CPU} * num_insts + $T_{I/O}$ * num_I/Os + T_{MSG} * num_msgs + T_{TR} * num_bytes

The T_{CPU} and $T_{I/O}$ are local time. T_{CPU} is time of CPU execution regarding the amount of instructions. $T_{I/O}$ is time of I/O disk to retrieve data. For the distributed databases on the non-local network, the communication time must be considered as in the last two factors. T_{MSG} is fixed for each message spending on initialize and receiving a message. Opposed to the T_{MSG}, the T_{TR} is time to transmit data from one site to another (or others) whose distances are different. However, to simplify the query optimization at this time, we will assume that all sites are equidistant. The transmission time is factored by the number of bytes (sum of all messages) of transmission.

The turnaround time in the distributed system is considering in parallelism for both local execution and communication cost. Thus each factor of the turnaround time is considered only the maximum cost of each factor of the total time.

$$\text{Turnaround\_time} =$$

$$T_{CPU} * \max(\text{num\_insts}) + T_{I/O} * \max(\text{num\_I/Os}) + T_{MSG} * \max(\text{num\_msgs}) + T_{TR} * \max(\text{num\_bytes})$$

where the max(x) is the maximum value of all x's executed in sequential in each site. The factors of other sites, that is the maximum, will be ignored for distributed cost function.

For example, suppose a message is to be transmitted in parallel from sites A to C x bytes and from sites B to C y bytes. The total time of communication is

$$\text{Total\_time} = 2T_{MSG} + T_{TR} (x + y)$$

Because the transmission is done in parallel, the turnaround time is

$$\text{Turnaround\_time} = \max (T_{MSG} + T_{TR} *x, T_{MSG} + T_{TR} *y)$$

$$= T_{MSG} + T_{TR} * \max(x, y)$$

The local optimization is to choose the execution plan that minimizes the factors of cost function, the number of instructions, number of I/O's, number of messages, and the size of the messages. The factors of cost function can be reduced by using the reduction technique mentioned in the earlier section of this chapter.

However, it does not mean that more parallelism will always improve system execution time. The total time is increased because of more local processing time. But the resource utilization is increased and the throughput is improved.

Chapter 5 New Methodologies and Techniques for MKT

Having lists of information of illnesses on web pages and letting people searching on their own is time consuming. There are also other medical web engines to prompt and ask for symptoms and find precise answers. However, those are based on one knowledge database. It is just like meeting a doctor at the clinic. Would it be better, if there is more than one doctor meeting a patient at the same time? Consulting several physicians, whose opinions differ because their experiences differ, especially if they come from different countries, could produce an optimal treatment plan.

5.1 Distributed Database on MKT

Medical researchers are maintaining medical knowledge in their databases to facilitate the discovery of better treatments. The broad objective is to enable humans to enjoy a long, healthy life. When a new medical discovery is found, the knowledge is published. So the new information can be disseminated. But not all medical researchers add it to their medical knowledge base. The researchers, who add this new discovery, can open and share their knowledge databases with other medical researchers. Distributed databases must be employed to manipulate this knowledge, and to improve the dissemination of new medical discoveries among physicians, medical researchers and other people involved in patient care.

The architecture of the distributed databases, proposed and tested in this research, is explained below. The following picture is the concept architecture of our distributed deductive database.

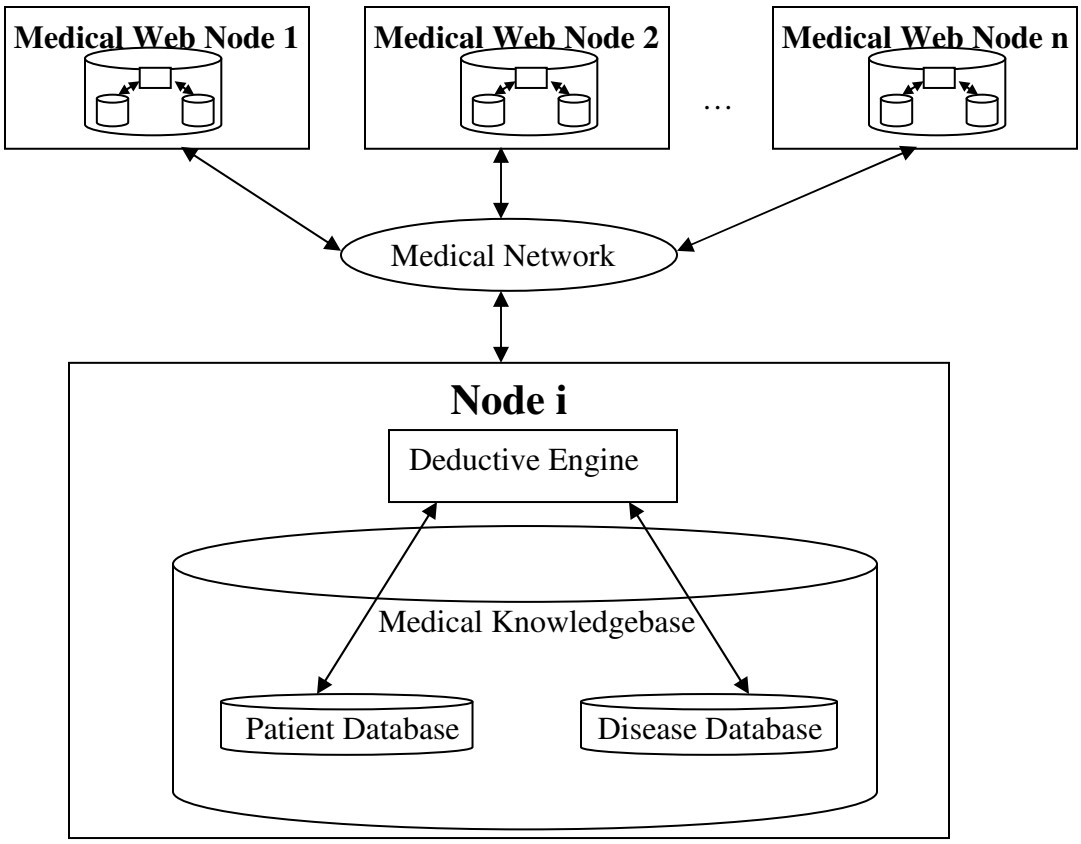


Figure 5- 1 Knowledge Nodes in Medical Network

The medical knowledgebase is functionally composed of two parts, the patient databases maintaining information for each individual patient, and the disease databases maintaining medical knowledge about illnesses.

The patient databases store two kinds of information, personal and medical information. Personal information such as name, id, address, and etc. are stored for administration work. The medical information store records of each individual patient who has different medical history such as allergy, or illnesses in the past. The personal medical data, such as blood type, or cholesterol are also stored as the medical information in the patient database. The medical information in the patient databases is used with the disease databases to analyze the patient's illness.

The disease databases emphasized in this research, store the knowledge of illnesses, both well-known diseases and just-discovered disease. The disease databases store medical knowledge in four major categories, illness names, symptoms, causes, and treatments. Many diseases are called differently. For example, human immunodeficiency virus is also called AIDS, HIV/AIDS or HIV. The illness name must store all its aliases, so the user can query with any name. Symptom and cause information is the first and most important for illness analysis using the deductive engine, which is explained later in this chapter. Last category is the treatment, which the patient wants to ask about.

The following is the diagram of the disease database in the medical knowledge base.

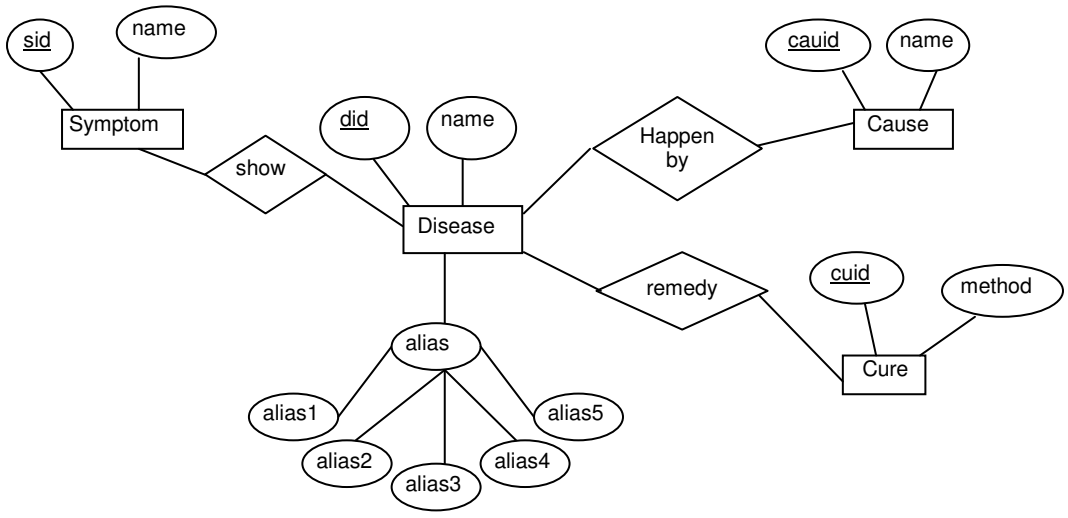


Figure 5- 2 Medical Knowledgebase: sid is symptom ID; did is disease ID; cauid is cause ID; and cuid is ID of method of cure

Each node in the medical network contains the same architecture as the distributed databases. However, the data inside the database may be different depending on the fragmentation. For this research the mixed (both vertical and horizontal)

fragmentation are used. The detail of the mixed fragmentation is explained in the chapter 4. Each node can use its deductive engine to query the information from the distributed databases, together, on the medical network.

In case of the private information of both patient and disease knowledge databases, one node contains two separately sets of databases. The shared medical knowledge is in one set that allows all nodes to retrieve its data. Another set blocks the outside retrieval. So the private information will not be provided and cannot be used outside the node. Private information of patient databases has been commonly defined by many organizations, including federal and non-profit groups. There are many unfinished or unapproved medical knowledge databases of on-going researches that should not be used in public. So this data must be blocked and used only for a specific group. When the deductive engine analyzes the illness for a user, only the shared medical knowledge can be used. Nevertheless, there are special databases that deal with genetics and inheritance of diseases. The information that is put into these databases is very carefully controlled. A committee of experts meets to decide what to put in.

The following picture is the concept architecture of our distributed deductive database considering private information.

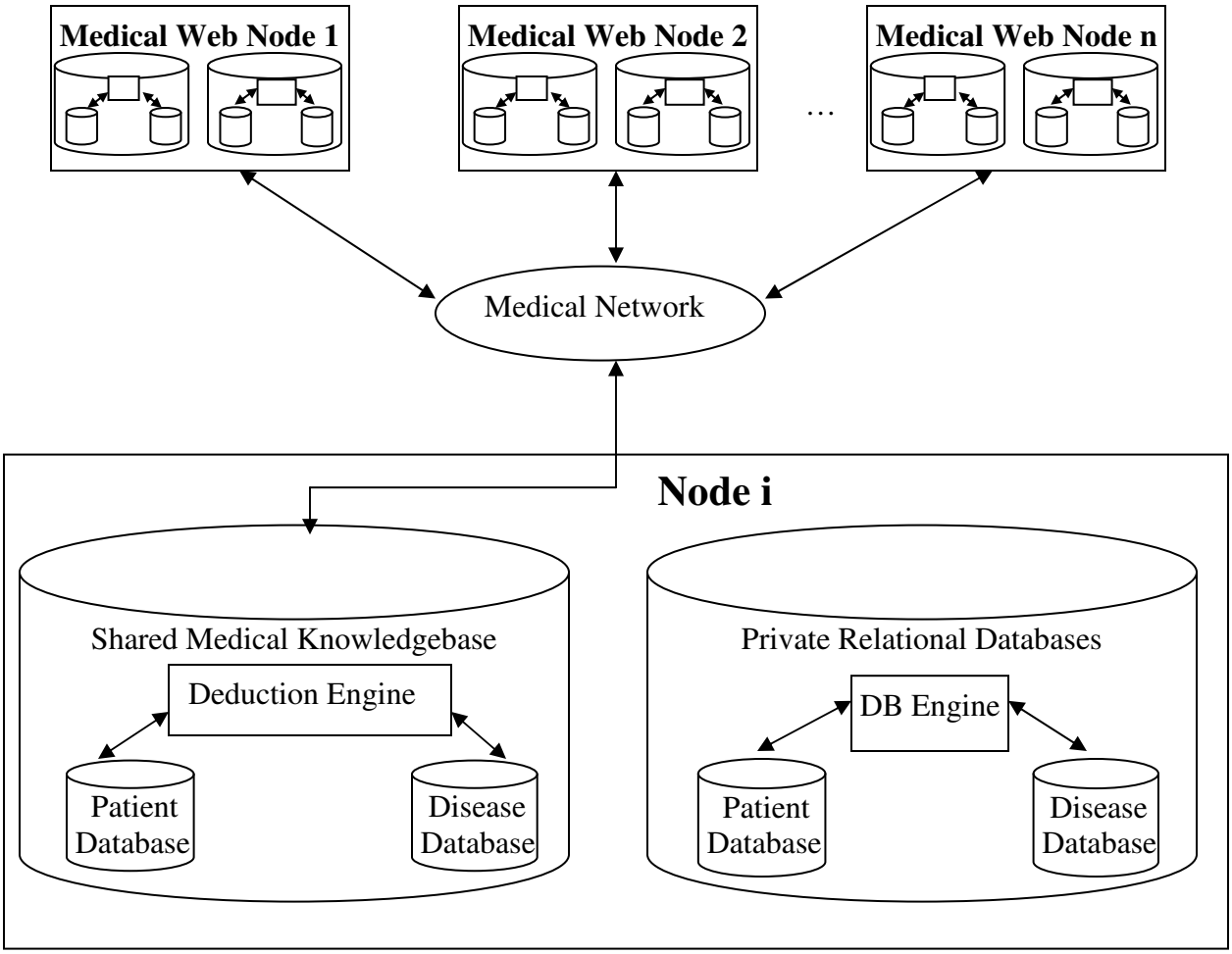


Figure 5- 3 Shared and Private Databases Connecting to Medical Network

5.2 Logical Breakdown

When a patient wants to find out about his/her illness, the query will be handled by the local medical node. The local medical node sends to all nodes of the medical web around the globe.

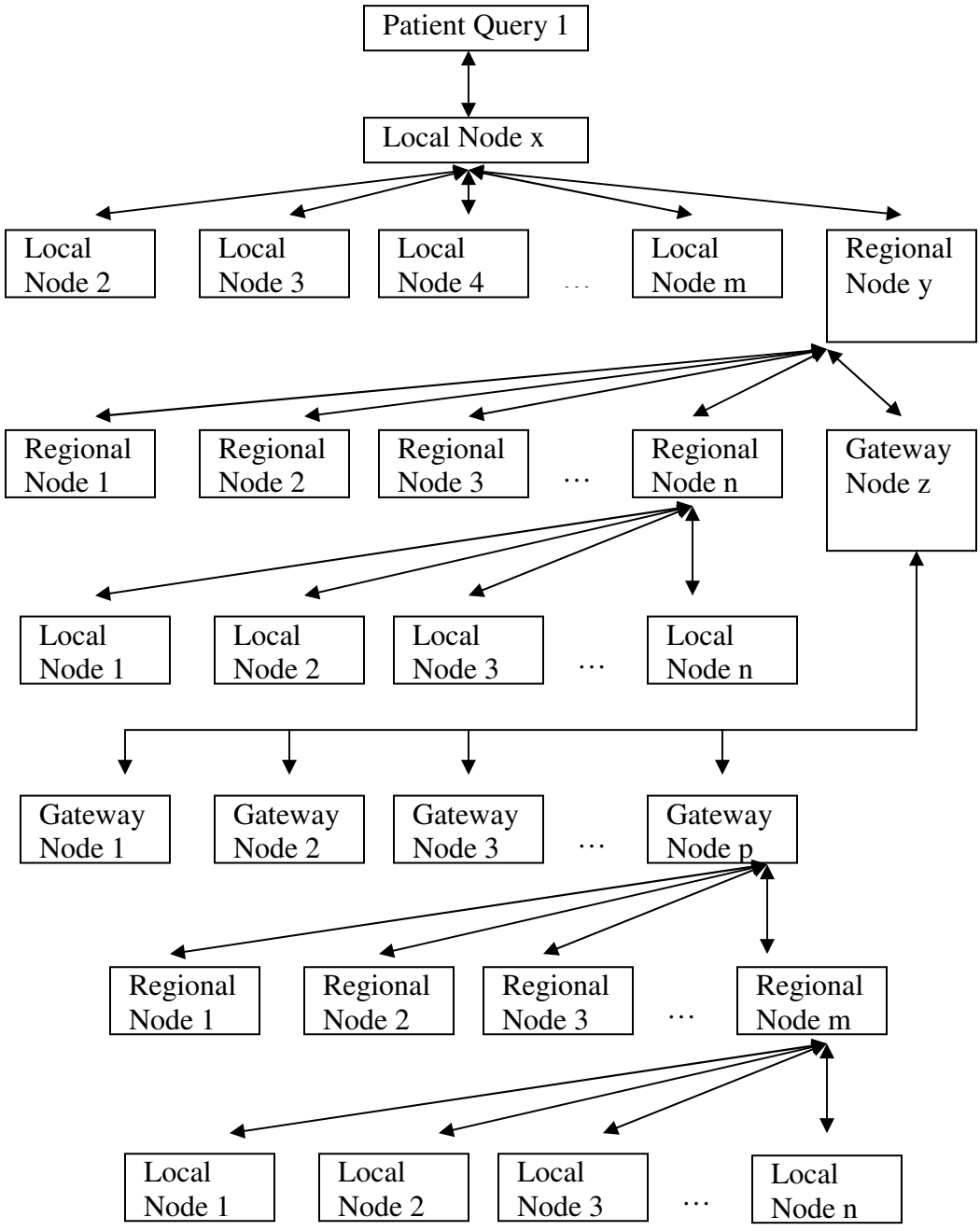


Figure 5- 4 Logical Connection when Patients Connect to Medical Network

The query is sent through as 3-Dimensional location, X-local, Y-Region, and Z-Gateway. At the original node, x, the query is sent only inside the same region.

[(1,y,z) (2,y,z) (3,y,z) … (x,y,z) … (m,y,z)]

The regional node, y, sends query to other regional nodes that compose of their local nodes.

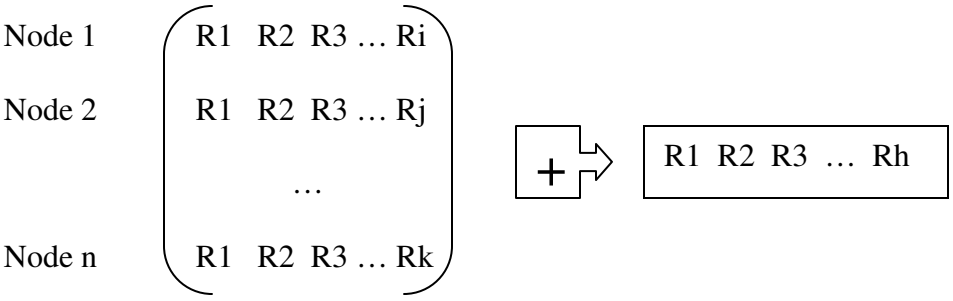
$$\begin{pmatrix} (1,1,z) & (1,2,z) & (1,3,z) & \dots & (1,n,z) \\ (2,1,z) & (2,2,z) & (2,3,z) & \dots & (2,n,z) \\ (3,1,z) & (3,2,z) & (3,3,z) & \dots & (3,n,z) \\ (x,1,z) & (x,2,z) & (x,3,z) & \dots (x,y,z) \dots & (x,n,z) \\ (m,1,z) & (m,2,z) & (m,3,z) & \dots & (m,n,z) \end{pmatrix}$$

The gateway node, z, sends the query to other gateway nodes that are composed of their regional nodes and local nodes

$$\begin{pmatrix} (1,1,z) & (1,2,z) & (1,3,z) & \dots & (1,n,z) \\ (2,1,z) & (2,2,z) & (2,3,z) & \dots & (2,n,z) \\ (3,1,z) & (3,2,z) & (3,3,z) & \dots & (3,n,z) \\ & & & \dots & \\ (x,1,z) & (x,2,z) & (x,3,z) & \dots (x,y,z) \dots & (x,n,z) \\ & & & \dots & \\ (m,1,z) & (m,2,z) & (m,3,z) & \dots & (m,n,z) \end{pmatrix} \leftarrow \begin{matrix} z = 1 \\ z = 2 \\ z = 3 \\ \dots \\ z = p \end{matrix}$$

The query is analyzed by each node. The result for each node may contain many answers. An answer of a node is composed of the illness and the level of confidence, depending on the location of the node.

Each node sends the result back to the original node through the structure of the medical web. Each particular answer is combined together from all nodes to adjust the level of confidence before it is sent it back to the patient.



This research uses the deductive database, a database with rules of inferences, to store the knowledge and distribute it around the globe. Both the address and a profile of the knowledge contents will be stored like the SCP that stores the address of the parties. For example, when a user (or doctor) encounters medical problems, the knowledge in the deductive databases from domestic and international can be gathered together by the judicious command by the user.

The first methodology is to store the knowledge at the gateway level of the intelligent telecommunication network. The query from a user will find the answer(s) from its gateway after interpreting the query to a detailed set of subqueries. If no answer is found at the gateway, or if it has been moved from its gateway, then the query will be sent to other logically linked gateways.

In comparison, the second methodology is to keep (or distribute) the knowledge at the local medical center, hospital, university and etc. When there is a query, it will ask at its local medical center first. If there is no answer or no highly confident answer(s), the query will be dispatched around the intelligent network of medical centers, corresponding to the location of the expertise of each illness, within the gateway. If there is very low confidence in the answer to the query, or if no answer is retrieved, then the query will be sent through the international medical network.

5.3 Definability of Confidence

5.3.1 Logical Deductive Tree

The step to finalize the conclusion of a patient illness is analyzed deductively. First of all, a doctor asks the patient what he/she feels. Patient responds with body reactions that are different from other regular days. For a doctor, these answers are symptoms with which the doctor can start with to diagnose the patient's illness. Considering the symptoms and the knowledge of the doctor, the doctor will have a list of likely illnesses that the patient might have. It is a list of many illnesses because there are many diseases sharing the same symptom. Using the symptom given by the patient, the doctor can cut down other unrelated illnesses that do have such symptom. The doctor may also consider other environmental factors, such as weather during last couple days, common or widespread disease during this time, history of the patient, etc.

After the doctor has a list of likely illnesses that the patient might have, she or he can ask the patient if the symptoms seen with these likely illnesses are present in the patient. These symptom questions can be asked deductively. If a symptom of a likely illness occurs to the patient, the likeliness value of this illness will be increased. The chance that the patient has this illness is greater. If a symptom of a likely illness does not occur to the patient, the likeliness value of this illness will be decreased. The chance of having this illness is less. However, it does not mean that the patient does not have this illness because a symptom may have not yet occurred or a particular symptom may occur to some people but not to everyone. So the doctor cannot completely rule out this illness. That is why the doctor has to confirm by asking about other symptoms of the same illness. The following diagram illustrates these symptom-analysis steps.

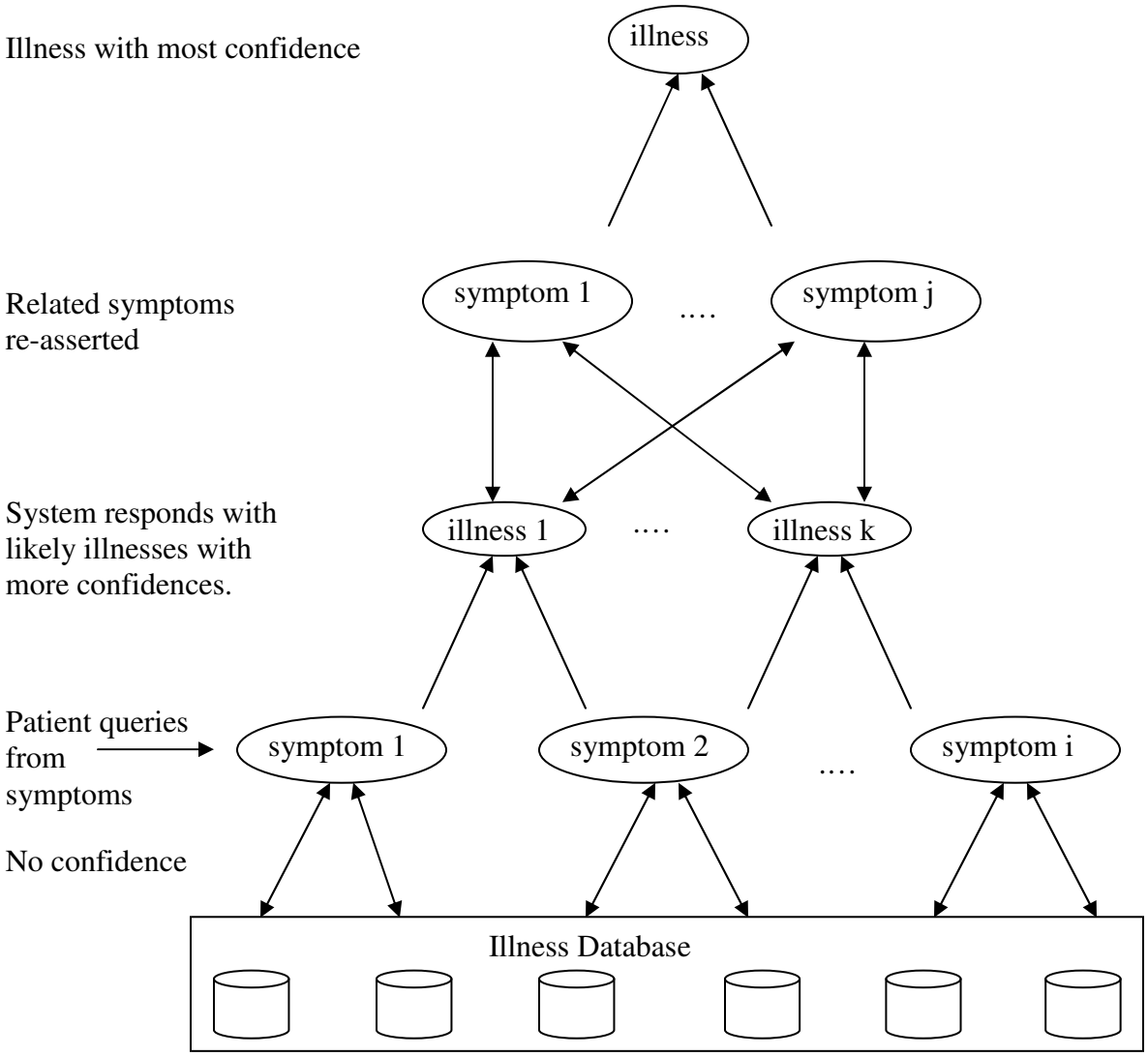


Figure 5- 5 Logical Deductive Tree on Symptoms

For an illness that has been continually decreased in value of likeliness after the patient says no in answer to questions about many symptoms, the chance of the patient having the illness will be less and, at some point of time, the doctor will stop asking about symptoms of this illness. In the opposite case, the illness whose symptoms have

been confirmed by the patient, the likeliness value will be increased. More symptoms of the illness that has high likeliness value will be asked the patient.

Until this time, the doctor has narrowed down from many illnesses to a few illnesses that have high likeliness value. Note that since there are many symptoms that may occur in the patient, it is faster for the doctor to start by asking about symptoms of the most common illnesses instead of asking the symptom that occur in rare diseases. That will help the processing of deduction move faster. After the common symptoms have been chosen, the rare (uncommon) symptoms will be chosen to ask the patient.

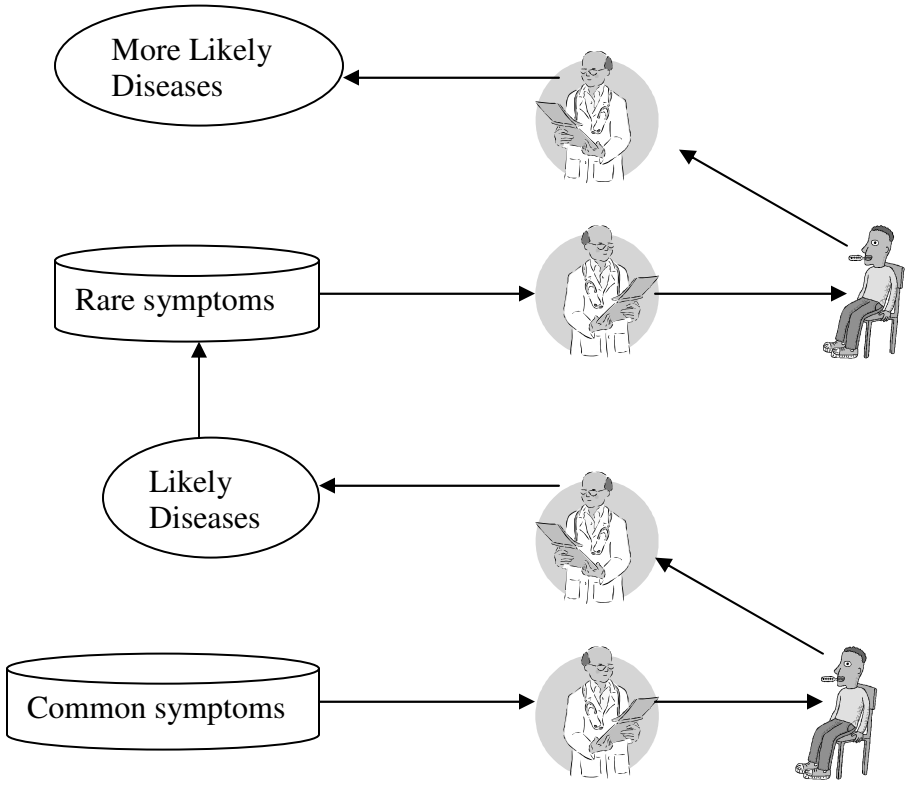


Figure 5- 6 Symptom Analysis

After checking the symptoms, the number of likely diseases for the patient would be reduced tremendously. The next process is to find out the causes of the disease by interviewing the patient with the cause list of each of these likely diseases. The history

record of the patient may be combined with the answers of the patient. Patient may or may not remember what he/she has done recently during this process. Therefore, the level of confidence may not increase. If a cause in the cause list matches with patient's record, either from interview or history record in the database, the level of confidence will increase. The system gives the patient the list of most likely diseases with their levels of confidence. Their choices of treatment are also given.

5.3.2 Layer of Deduction for Confidence Levels

Confidence level is the probability value associated with confidence interval given an estimated range of values which is likely to include an unknown population parameter, the estimated range being calculated from a given set of sample data.

Before we define the values of confidence levels using in our research, we will discuss the three layers of medical examination for this research.

Layer 1 Receiving the symptoms from a patient. Number of symptoms from patient is depending on how the patient is conscious of unusual occurrence to his health. At the first step of medical examinations, the patient will notify by him/her self. We called this layer "Patient Symptoms".

Layer 2 However there are symptoms that may occur unnoticed to the patient. The medical expert must gather the other symptoms from the likely illnesses whose symptoms were given at the first step. We call this layer "Mark Symptoms". The deduction of finding the right illness starts at this layer. The following are different deductive algorithms:

- If symptom(s) of a disease A, given by the medical expert, are not recognized by the patient, the second list of symptoms from disease B will be asked and so on for disease C, D, etc. This algorithm is considered one disease at a time.

- The second algorithm is considered all likely illnesses, whose symptoms were matched with first step, together. The common symptoms of all (or most amount of) diseases will by asked first. If one or more of the given symptoms occurs to the patient, other illnesses that does not have this symptom will be eliminated from the choice of likely diseases. However, if theses common symptoms do not occur to the patient, these diseases, having these common symptoms, will be eliminated and then the second, third, and lower common symptoms will be asked. The worst deductive case is that patient has a few and rarely occurring symptoms. This deductive step might be time-consuming for this worst case. However, because of the rarely occur and uncommon symptoms, it is a very small chance that this algorithm will reach the worst case. However, if the reported symptoms occur very frequently, the deductive process in this algorithm will probably find the likely illness faster the other algorithms.

- The third case is the combination of the first and second algorithms. The medical expert shows all symptoms together, not once at a time as in the second algorithm, but the order of symptoms is starting from the most common occurring symptom to the least common occurring symptom. There is no switching page-by-page for user's intolerance and reduce processing time by the program to deductively reevaluate for each symptom when there are lots of symptoms that

may occur to the patient. As for this research, we choose this algorithm in our testing programs.

Layer 3 Before the conclusion, the cause of illnesses can be used for deduction. Similar to the layer 2, the cause of illness may be unnoticed to the patient. Therefore, the previous three algorithms can be used to find out about the cause(s) of patient's illness. As for simplicity, the third algorithm is used for our research. However, the cause list will not be ordered by the commonality of causes, as symptoms in the second layer. Because the deductive medical analysis of occurring symptoms was evaluated in the second layer, it will be ordered by the likeliness of illnesses to the patient.

Conclusion After the last layer, the patient will be given the conclusion of the examination and so we called it "Disease Conclusion". Generally, final goal of visiting medical expert of a patient is to receive medical advice including method of treatment, health and life-style adjustments, etc. Mostly, the medical expert may give the medical prescription to the patient. Most physicians will prescribe the standard medical treatment, which medical experts generally agree is best. However, in some cases, for diseases that are not well understood, it may be useful to consider alternate treatments. These could be treatments for which there is preliminary evidence that they are beneficial, but the evidence is not yet conclusive, so the treatment is not considered part of standard treatment. The Internet can be useful in identifying such alternate treatments and the evidence to support their merits. (Unfortunately, sometimes the Internet is used by dishonest people who try to promote ineffective treatments to people who have incurable illnesses.) Therefore, many medical computerized experts over the Internet only show the

types of medicine and method of remedy suitable to the patient but do not prescribe the medical prescription.

5.3.3 Confidence Levels on Single Medical Agency

The confidence levels are defined after each layer of the medical deductive process. As for our research, we distribute the medical knowledge in multiple databases. For simplicity, the confidence level is defined for only one source of medical knowledge. For multiple sources, the confidence levels are defined in the next section.

After Layer 1

At the first layer, we give at most a 50% confidence level for the one medical knowledge database. Because we divide the medical examination into three layers, confidence level of the higher layer is one-sixth higher, $\frac{1}{3}$ of 50% = 1/6. Therefore, after the first layer, the constant ratio of the confidence level is 50% + 1/6 = 4/6.

Let α be the number of patient symptoms

β be the number of diseases having patient symptoms

i be the number of choices given to the patient

and CL be the confidence level.

The larger the number of symptoms reported by the patient, the better the analysis leading to a conclusion. The second factor is the ratio between the known symptoms by the patient, α, comparing to the number of allowed inputs, i, given to the patients. For example if a physician asks “Give me five symptoms”, then i is 5. The i could be any number that a patient could reasonably be expected to report. As for our research, i is 3.

The third factor is the number of diseases, β, having all patient symptoms. The highest confidence level is when β is 1, on which only one disease has all patient symptoms. The confidence is less when more diseases return from the query.

Therefore, at the first layer, the confidence level, CL is

$$CL = (50\%) + ((\tfrac{1}{3})*50\% * (\alpha / i) * (1/\beta))$$

$$CL = (3/6) + (1/6 * (\alpha / i) * (1/\beta))$$

After Layer 2

At the second layer, the constant factor is one-third higher, $\tfrac{2}{3}$ of 50% = 2/6, from the first layer because of three-layer methodology. Therefore the constant ratio of confidence level at the second layer is 50% + 2/6 = 4/6.

At the second layer, the patient is given n choices of symptoms of disease related to patient symptoms given in the first layer by the patient. If the patient recognizes other symptoms from the list, the confidence level will be higher. The number of new symptoms recognize by the patient is θ.

We separate the confidence level into two sets, the confidence level of each disease and the overall confidence level. The confidence level of each disease in the second layer is factor by the ratio between all the known patient symptoms including symptoms in both first, α, and second, θ, layer, and the all symptoms, n, of the disease, $[\alpha + \theta] / n$.

The third factor, applied only to the overall confidence level, is the number of diseases, β, but is not applied in the same approach as in the first layer. It is applied for the average of the confidence levels of all diseases.

Therefore, at the second layer, the confidence level of each disease, CLD is

$CLD_i = (50\%) + ((\frac{2}{3})*50\% * ([\alpha + \theta] / n_i))$

$CLD_i = (3/6) + ((2/6) * ([\alpha + \theta] / n_i))$

where α is the number of patient symptoms in the first layer

θ is the number of patient symptoms recognized from the list in the second layer

β is the number of diseases having patient symptoms

n_i is the number of choices of all symptoms of a disease i showing to the patient as a list

and CLD_i is the confidence level of each disease i.

The overall confidence level, CL, of the second layer is

$CL = (CLD_1 + CLD_2 + CLD_3 + \ldots + CLD_\beta) * (1/\beta)$

$CL = (\sum_{i=1}^{\beta} CLD_i) / \beta$

where $CLD_1, CLD_2, CLD_3, \ldots, CLD_\beta$ are the confidence levels of all disease having patient symptoms.

After Layer 3

At the layer 3, the causes of all related diseases are shown. The patient chooses his/her cause(s) recognized from the list. The causes are used for illness(s) inference together with the symptoms given in the first and second layers. If there is only one disease corresponding to the symptoms and causes, then the confidence level, whose value between 0 and 1, is 1.

If there are two or more diseases matching the symptoms and causes of the patient, then confidence of each disease i is delineated by two factors the symptoms and causes. The symptoms weight 5/6 as in the second layer and the causes weight the rest, 1/6. The cause factor is the ratio between the number of causes recognized by the patient (after the doctor reminds the patient of what the patient may have done in the past) of the disease i and the number of causes of disease i. Because the patient is likely to recognize only one or few causes, the cause factor is logarithm ratio, first know cause is worth more than second cause, and so on. σ_i is added by 1 because when $\sigma_i = 1$, $\log_{m+1} 2$ is not 0. If it is not added by 1 when $\sigma_i = 1$, $\log_{m+1} 1$ is 0, which means the one cause does not effect which is not possible. Therefore, the cause factor is logarithm of $(\sigma_i + 1)$, not logarithm of σ_i. Because of 1 is added to σ_i, m_i must be added by 1 to accommodate when $\sigma_i = m_i$. The overall confidence level is the average of confidence levels of all corresponding diseases. For the case that patient does not input any cause, the confidence level remains as in the second layer.

Therefore, the overall confidence level, CL, for the conclusion is

If $\beta 2 = 1$ then

$$CL = (5/6) + (1/6) = 1$$

Else if $\beta 2 > 1$ then

$$CLD_i = [3/6] + [(2/6) * ([\alpha + \theta] / n)] + [(1/6) * \log_{(m_i + 1)} (\sigma_i + 1)]$$

$$CL = (CLD_1 + CLD_2 + CLD_3 + \ldots + CLD_{\beta 2}) * (1/\beta 2) + (CLD_{\beta 1} + CLD_{\beta 1+1} + CLD_{\beta 1+2} + \ldots + CLD_{\beta}) * (1/(\beta - \beta 1))$$

$$CL = (\sum_{i=1}^{\beta} CLD_i) / \beta$$

Otherwise (when $\beta 2 = 0$)

$$CLD_i = (3/6) + [(2/6) * ([\alpha + \theta] / n_i)] + 0$$

$$CL = (CLD_{\beta 1} + CLD_{\beta 1+1} + CLD_{\beta 1+2} + \ldots + CLD_{\beta}) * (1/(\beta-\beta 1))$$

$$CL = (\sum_{i=\beta 1}^{\beta} CLD_i) / (\beta-\beta 1)$$

where α be the number of patient symptoms in the first layer

θ be the number of patient symptoms recognized from the list in the second layer

β be the number of diseases having patient symptoms

$\beta 1$ be the number of diseases having patient symptoms but not cause(s)

$\beta 2$ be the number of diseases having patient symptoms and cause(s)

$\beta = \beta 1 + \beta 2$

n_i be the number of choices of all symptoms of a disease i showing to the patient as a list

σ_i be the number of patient cause for disease i

m_i be the number of cause for disease i

and CLD_i be the confidence level of each disease i.

5.3.4 Confidence Levels on Multiple Medical Agencies

There are three situations of agreement when multiple agencies coordinate the deductive process.

- **Complete Agreement** All agencies agree on the same conclusion. As for our research, all multiple medical sources agree on the same disease for conclusion.

- **Partial Agreement** Some agencies, but not all, agree on some conclusions, and also do not agree on some conclusion.
- **Complete Disagreement** Conclusion of each agency has not agreed with the conclusion of any other agencies.

After Layer 1

The confidence level for the multiple medical sources is the extension of the first 50% (or 3/6) factor of the confidence level in the layer 1 of the single medical agency. It is also divided 3 into three layers. Therefore, in layer 1 of multiple agencies, it is one-third of 50% = 1/6. If all knowledge sources agree on the same conclusion, the 50% factor of confidence level remains the same as of single agency.

$CL = ((1/3)*50\%) + ((1/3)*50\%) * (\alpha / i) * (1/\beta))$ when all agencies agree.

If none of conclusion of any agency agrees with others' conclusion, then 50% is discarded.

$CL = 0 + ((1/3)*50\%) * (\alpha / i) * (1/\beta))$ when no agreement at all.

In the case that there is agreement on some conclusion(s), the 50% factor is rationalized by the ratio of the amount of agreeing sources to the amount of all sources.

$CL = [(1/3)*50\% * \lambda / s] + [(1/3)*50\% * (\alpha / i) * (1/\beta)]$

when some agencies agree

where λ is the number of sources agreeing

s is the number of all sources

The case of some agreement can be generalized for the cases of all sources and no source agreeing. Therefore, the generalized term of confidence level for the level 1 is

$$CL = [(1/3)*50\% * \lambda / s] + [(1/3)*50\% * (\alpha / i) * (1/\beta)]$$

$$CL = [(1/6) * \lambda / s] + [(1/6) * (\alpha / i) * (1/\beta)]$$

where α is the number of patient symptoms

β is the number of diseases having patient symptoms

i is the number of choices given to the patient

λ is the number of sources agreeing

s is the number of all sources

and CL is the confidence level.

After Layer 2

At the second layer, the factor of multiple sources becomes two-thirds of 50%, that is 2/6. Similar to the single source, the confidence level at this layer of multiple sources is, still, separated by each disease, and the average of confidence levels of all diseases is the overall confidence level. However, because of multiple sources, confidence level of each disease must be categorized for each knowledge source. Two methodologies are proposed, the classification by diseases of each knowledge source, and the agreement of sources on each disease.

- The classification by diseases of each knowledge source defines the confidence level of each disease in same way as of the single source but for each source.

$$CLD_i S_j = (2/6) + [(2/6) * ([\alpha + \theta] / n_i)]$$

where α is the number of patient symptoms in the first layer

θ is the number of patient symptoms recognized from the list in the second layer

n_i is the number of choices of all symptoms of a disease i showing to the patient as a list

and $CLD_i\ S_j$ is the confidence level of each disease i for each source j.

The confidence level of all diseases of source S_j, CLS_j, of this layer is the average among the diseases of the same source.

$$CLS_j = (CLD_1S_j + CLD_2S_j + CLD_3S_j + \ldots + CLD_\beta S_j) * (1/\beta)$$

$$CLS_j = (\sum_{i=1}^{\beta} CLD_iS_j) / \beta$$

The overall confidence level CL, of the second layer for the classification by diseases is the average among all knowledge sources.

$$CL = (CLS_1 + CLS_2 + CLS_3 + \ldots + CLS_s) * (1/s)$$

$$CL = (\sum_{j=1}^{s} CLS_j) / s$$

$$CL = (\sum_{j=1}^{s} [\sum_{i=1}^{\beta} CLD_iS_j] / \beta) / s$$

where β is the number of diseases having patient symptoms

s is the number of knowledge sources

$CLD_i\ S_j$ is the confidence level of each disease i for each source j.

and CLS_j is the confidence level of each source j.

- The agreement of sources on each disease rationalizes the two-third of 50% factor by the ratio between the number of agree sources and the number of all sources. Similar to the first layer but defined for each disease, the two-third of 50% factor remains for conclusion of agreement of all sources, and the two-third of 50% is discarded for a

case of no agreement. Therefore, the confidence level of each disease for multiple sources and the overall of the confidence level is:

$$CLD_i = [(2/6) * (\lambda_i / s)] + [(2/6) * ([\alpha + \theta] / n_i)]$$

$$CL = (\sum_{i=1}^{\beta} CLD_i) / \beta$$

where α is the number of patient symptoms in the first layer

θ is the number of patient symptoms recognized from the list in the second layer

β is the number of diseases having patient symptoms

n_i is the number of choices of all symptoms of a disease i showing to the patient as a list

λ_i is the number of sources agreeing on symptoms for disease i

s is the number of all sources

CLD_i is the confidence level of each disease i.

and CL is the overall confidence level after the layer 2.

The first methodology is simpler but less accurate than the second methodology. Therefore, we choose the second methodology in our research.

After Layer 3

Now the third portion of 50% factor is considered in the layer 3 for the multiple agencies. As for our research, it is the factor of patient's causes whether they are agreed on multiple sources. Therefore, the third portion of 50% factor is rationalized by the ratio of the number of agreeing sources on user cause(s) to the number of all sources. Other factors of confidence levels remain the same as in the single knowledge source.

Therefore, the overall confidence level, CL, for the conclusion is

If $\beta 2 = 1$ and $\lambda 2_i = s$ then

$$CL = (5/6) + (1/6) = 1$$

Else if $\beta 2 > 1$ then

$$CLD_i = [(2/6) * (\lambda_i / s)] + [(1/6) * (\lambda 2_i / s)] + [(2/6) * ([\alpha + \theta] / n)] + [(1/6) * \log_{(m_i + 1)} (\sigma_i + 1)]$$

$$CL = (CLD_1 + CLD_2 + CLD_3 + \ldots + CLD_{\beta 2}) * (1/\beta 2) + (CLD_{\beta 1} + CLD_{\beta 1+1} + CLD_{\beta 1+2} + \ldots + CLD_{\beta}) * (1/(\beta - \beta 1))$$

$$CL = \left(\sum_{i=1}^{\beta} CLD_i \right) / \beta$$

Otherwise (when $\beta 2 = 0$ of all sources)

$$CLD_i = (2/6 + 0) + [(2/6) * ([\alpha + \theta] / n_i)] + 0$$

$$CL = (CLD_{\beta 1} + CLD_{\beta 1+1} + CLD_{\beta 1+2} + \ldots + CLD_{\beta}) * (1/(\beta - \beta 1))$$

$$CL = \left(\sum_{i=\beta 1}^{\beta} CLD_i \right) / (\beta - \beta 1)$$

where α is the number of patient symptoms in the first layer

θ is the number of patient symptoms recognized from the list in the second layer

β is the number of diseases having patient symptoms

$\beta 1$ is the number of diseases having patient symptoms but not cause(s)

$\beta 2$ is the number of diseases having patient symptoms and cause(s)

$\beta = \beta 1 + \beta 2$

n_i is the number of choices of all symptoms of a disease i showing to the patient as a list

σ_i is the number of patient cause for disease i

m_i is the number of cause for disease i

λ_i is the number of sources agreeing on symptoms for disease i

$\lambda 1_i$ is the number of sources agreeing on symptoms but not cause(s) for disease i

$\lambda 2_i$ is the number of sources agreeing on symptoms and cause(s) for disease i

$\lambda_i = \lambda 1_i + \lambda 2_i$

s is the number of all sources

and CLD_i is the confidence level of each disease i.

For future research, the confidence levels can be enhanced by:

- influence factor for causes may have weight factor scale between, let us say, 1 to 5 depending on the commonality of occurrences among patients or among diseases.

- influence factor for symptoms that can split into two levels, user symptoms and database shown symptoms. Each level may have a different weight value of factor.

5.4 Query Analysis and Evaluation & Level of Programmability

The level of programmability is broken down into four levels, user level, query analytical level, query evaluation level, and presentation level. All levels are implemented in four steps of programmability -symptom input, symptom marking, reassertion, and concluding.

The first step is the symptom input. A user gives his/her symptoms through simple textboxes arbitrarily and/or list-box given choices of usual symptoms. Query analyzer builds a generic query according to user input. Considering data fragmentation on different distributed databases, query analyzer reproduces the generic query into dynamic SQL codes. The dynamic SQL codes are sent to separating databases to retrieve other symptoms of diseases having the same symptoms given by the patient. After each database engine sends back the result, query evaluation combines and categorized the results differently depending on the level of confidence. The presentation level generates dynamic HTML code to user browser. The user can easily navigate on the result.

The narrowed-down symptoms from the first step are used as the input for the user level in the second step, symptom marking. User can simply choose any of these narrowed-down symptoms occurring to the patient. Unlike the user level of the symptom-input step, the user level of the symptom marking step adds the symptom-id, sent by the presentation level through the query evaluation level of the first step. The query analyzer in the second step will use the symptom-id, instead of text-type symptom from user, to generate a new generic query. Therefore, the database engines will process the queries on less work load. The query analyzer creates a generic query according the new user symptom id and the original user input. The new generic query is to find cause of disease

that is likely to occur according to the user symptoms. Although the query may be longer because of more given input to be considered, the result will be better. Also, blending with the data fragmentation on distributed databases, the query analyzer filters the generic query into dynamic queries that should be less complex. The database engines run the dynamic queries, and send the results to the query evaluation level. The new results, causes of likely-occurring diseases, from various databases are combined at the query evaluator who recalculates the level of confidence. With the new proper results, the level of confidence would be increased. The presentation level generates HTML source code from the results of the first and second steps, with the level of confidence.

With a higher level of confidence, the program must reassure the results before giving the conclusion. It is the third step, called reassertion, to collect more data in different angles from user. The third level deals with the cause of disease that may occur to the user. A list of causes of likely diseases is chosen for the patient provided at this user level. There are some cases in which a patient may not know or remember what he or she has done. This step may be skipped by the user. If the user knew the cause(s), the user level will pass the cause-id to the query analytical level. Query analyzer combines the user cause(s) with his/her symptoms from two previous steps to create a generic query to find a final answer of the most likely disease that may occur to the patient. Just the same as the previous steps, the query analyzer generates dynamic queries for distributed databases to find the final answer. The database engines retrieve diseases and their methods of treatment by processing the dynamic queries and send them to the query evaluation level. The query evaluator collects the results among the distributed databases and defines the level of confidence to user. The presentation produces the HTML source

code to show the final answers composed of most likely disease(s), level of confidence(s), and their method of treatment to the user.

As user the point of view, the conclusion level is done in the third step. As of programmability, the system must collect the result for statistics in the fourth level. The system collects all result and the performance such as time of execution to the database. For future analysis, the system can use these data for better perform and accuracy. The only level performs the work here is the query analyzer. However, the query here is not for user request. The query analyzer produces a generic query only for the main database to collect result for research. However, for a better system, the query analyzer may generate dynamic queries to distribute on all distributed databases, so they can keep records under their performances. For this research, only the main database collects the statistics.

Figure 5- 7 demonstrates the creation of dynamic query using in our programs. When user submits a query to our disease analysis, the system will decompose the query into classified words. These classified words are used to create dynamic queries depending on the type of words and the number of words. The generated queries are dynamic because of the data fragmentation which is depending on the deductive distributed databases.

Queries are distributed to the databases. Each database generates raw results which will be evaluated for the confidence level. After the evaluation, the system will generate a dynamic HTML program. The programs are dynamically changed according to the results after the evaluation of raw answers. The structure of the dynamic program is the same, but not the content.

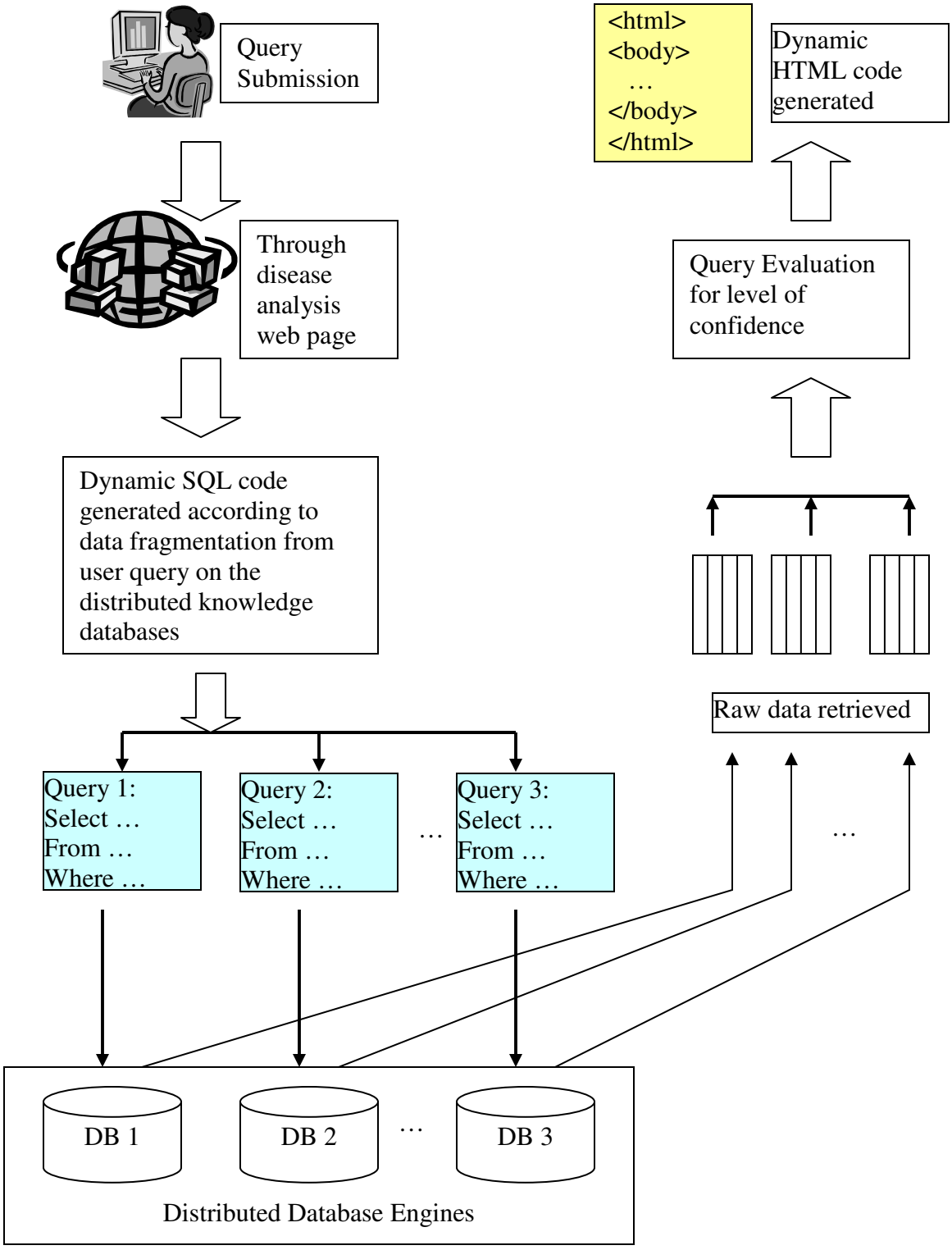


Figure 5- 7 Dynamic Query

5.5 Intelligent Medical Search Engine by Knowledge Machine

Many modern search engines for information retrieval system retrieve a large number of documents from a word-search query. Yet a simple question in natural language requests only an answer that is simple, accurate, and likely only one. A whole document of search result causes the inquirers to read through or find the answer in the document. Also too many documents in return distract the inquirers from their concern. We propose an idea of using knowledge machine to retrieve one finest answer for one-question-one-answer query. A search engine for natural-language application on medical field is introduced and its possible impact is discussed.

There are many search engines accepting natural language submitted by Internet users browsing for answer(s) of their questions. Google, www.google.com, and AltaVista, www.altavista.com, use relevant terms given by the user to rank the return results. The greater the number of links to a document, the higher the rank to the document ranking by Google. Both popular search engines use Boolean logic search such as AND, OR and the proximity search by default. AltaVista search engine uses "+" sign to filter search to a partial domain range of URL while Google offers, www.google.com/unclesam to search only on .gov and .mil domains. Results returns by both search engines are an enormous number of documents. Nevertheless AltaVista avoids multiple results from a single site by clustering results together into one result, called results grouping. Ask-Jeeves, www.ask.com accepts plain English questions that look for a fact-based answer. The answer returned by Ask-Jeeves is marked with "[Web Answer]" on the top rank of results if the question is included in the Ask-Jeeves question database. Otherwise the results might not be term-relevant to the question. Ask-Jeeves would be the most intellectual

search engine if its database can contain answers of all possible questions. But it is not likely to do so because the world knowledge, especially facts, is growing every second. We are proposing an achievable intelligent search engine in two visions. The knowledge database would be capable of answering any questions if it is demonstrated on a specific subject that growing (or changing) at a slow pace, such as the medical field. Another point of view is that most of the returning results from search engine are not relevant to the intention of a question from an inquirer. One direct answer would be preferred or at least highlighted or showed at the top of a long answer before showing all the other information related to it.

5.5.1 Natural Language

Humans have developed a complex structured system of signs and symbols to communication among them. They call it language. Animals have also developed a way to communicate such as roaring of lions indicates their aggressivity or territoriality. With a small number of signs and symbols for each species, their communication is limited. In contrast, human's language has many different signs, symbols, and mixture of them, and can create an unlimited number of reliable and different messages. There are more than 5,000 human languages [66]. In this section, we will explain how human communication works and present a simplified version of English.

Humans perform an action to produce language. This action is called statement. We categorize the statement acts in the following seven different groups:

Inform is a sentence presenting a fact, both confirmed and unconfirmed, where the confirmed fact provides data to the knowledge databases: *There are fifty states in United States of America*.

Query is a question that is looking for an answer for an aspect of the fact: *Who was the first king of England?*

Answer is an informing in reply to a query: (to answer the above question) *Egbert was the first king of England.*

Command or request requires a response back in return: *Please sit down.* "Please" word provides the politeness from the speaker whereas *"Sit down"* is still a request or command showing the toughness or rudeness that would be more appropriate for some cases such as commanding to a dog.

Promise or offer suggests an act that will occur or may occur if a condition is met: *I will buy you a toy if you behave today.*

Acknowledge is a response statement for the command or promise *Thank you.* It is not an answer that responds to a query.

Share provides feeling, experience or opinion to a fact *I like him a lot*.

The last four groups of statements –command, promise, acknowledge, and share- provide an advance of communication to languages with which humans can express their different emotions and conditions. The first three groups of statements –inform, query, and answer- provide fundamental statement without showing emotions. Humans would be quite boring if using only the first three groups. However, they are very important for expanding our knowledge base, which provides the means through which our civilization can grow. As for search engine, we will concentrate on only the first three groups.

Before discussing the first three groups of statements in the knowledge machine for an intelligent search engine, the fundamentals of language must be reviewed. There are three fundamentals of any languages:

- Lexicon is a list of allowable vocabulary words, digits, and special characters in dictionary of a language. Words in an English dictionary are categorized in nine types. **Nouns** identify things such as table. **Pronouns** are used to refer to a noun mentioned previously such as she, you, me, it. **Names** identify individual such as Tom, Katrina. They are arbitrary. So they are unbound in a dictionary. **Verbs** are acts or events such as born, go, be (is, am, are). **Adjectives** are used to modify nouns. To modify verbs, a language provides **adverbs**. **Articles** are used to refer any member of a group or a specific member of a group. There are only three articles –a, an, and the- in English dictionary. **Prepositions** expresses movement, such as to, in; location such as in, on, at; and time, place, and objects such as near, at, around. **Conjunctions** connect two or more lexicons or statements together.

- Syntax or grammar is a set of validation rules to combine lexicons together in an order that is agreed by all users (or inventors) of the language. There are many algorithms for parsing a statement, by using syntax, into a tree structure for semantic interpretation.

- Semantics is the meaning of a statement from which combination of lexicons apply to the syntax. Semantics plays an important part of understanding what is representing in the statement. Robust interpretation [69] can be used for text interpretation.

5.5.2 Collection of Knowledge

The knowledge database unit collects the facts from the logical statement that is either True or False, such as inform and query-answer statements. The other kinds of statements, such as commands or acknowledgements; are not logical statements that require complex systems to gather them into knowledge database. The knowledge machine stores facts from the real world statement into the database unit but not all the facts depending on their types. Some facts can be stored while others may just to be referred to.

An inform statement can be either a confirmed or unconfirmed fact. Also it can either be instant (time-vary) or constant (not-time-vary) fact. "The world population is 6,472,220,871 on the October 12, 2005 at 2:44 p.m. EST" is an instant confirmed fact whose value is changed every minute and confirmed by www.census.gov [67]. While "Earth is 12,742 kilometers (7,900 miles) in diameter" is a constant confirmed fact whose values is unchanged (for long period of time) and confirmed by www.nasa.gov [68].

Considering the trust of confirmation of fact, we define it in four levels of confidence in the knowledge database for knowledge processing.

1) Absolute Confirmed Facts These facts are absolutely true. They include:

- the past fact that has been recorded such as "George Washington was born on February 22, 1732" or "President George W. Bush is the 42nd president of United States".

- a fact that is already proved, measured, or tested such as "Newton's physics law: For every action there is an equal and opposite reaction" or "Earth is 12,742 kilometers (7,900 miles) in diameter".

- the agreement or standard fact such as “1,000 meters = 1 kilometers.” or “There are 26 letters in the English language”.

Because these facts are not likely to change, the knowledge machine stores them in the knowledge database with a very high degree of level of confidence. (The degree of level of confidence is later used to calculate the level of confidence when the knowledge machine concludes the answer of a query.)

2) Trusted Confirmed Facts These facts are likely to be true. They are confirmed or estimated by organization(s) or group of trusted experts such as “The temperature in the core of sun is estimated to be over 15,000,000 degrees Celsius” [70] that is estimated by NASA or “The US population is 297,411,874 on Oct/13, 2005 at 3:10 p.m. EST” [67] that has been recalibrated by the U.S. Census Bureau, Population Division of the US government. Both NASA and U.S. Census Bureau are organizations trusted by people in United States. NASA is also likely to be trusted internationally in the matter of stars and universes. Therefore, when the knowledge machine defines the degree of level of confidence, it must assort the degrees according who-trust-who in which topics. Since these facts are not absolute, the degree of level of confidence of these facts must be lower than the degree of the absolute confirmed fact.

3) Facts to be Confirmed There are lots of facts especially on the Internet. Many of these facts have been discovered or estimated by a person or group of persons who are not well-known or trusted. These facts also include the facts that have not been proved or cannot be proved scientifically such as “There are lives after death”. However it does not mean that these facts are false. These facts must be stored in the knowledge database and may be processed by the knowledge processing unit as facts that lack confirmation or

cannot be confirmed. Therefore, the knowledge database unit must assign the degree of level of confidence to them whose values are less than of the first two levels of facts. To define the diversity of the degree of level of confidence, the knowledge machine needs help from experts in the topics in which they specialize. In case of no available expert, majority of polling or maximum record of counting may be used to define the diversity of the degree.

4) False Premises The knowledge database in the knowledge machine could store the false premise to prevent the invalid knowledge collection. Results from knowledge processing would be more reliable for users who query information from the knowledge machine. However, the false premises are indefinite. The knowledge machine may ignore, or minimize them and processes a query only from the first three levels of facts. Nevertheless, there are user questions that may need answer(s) from the false premise and commonly ask. To reduce processing time of such questions, the database should collect some false premises.

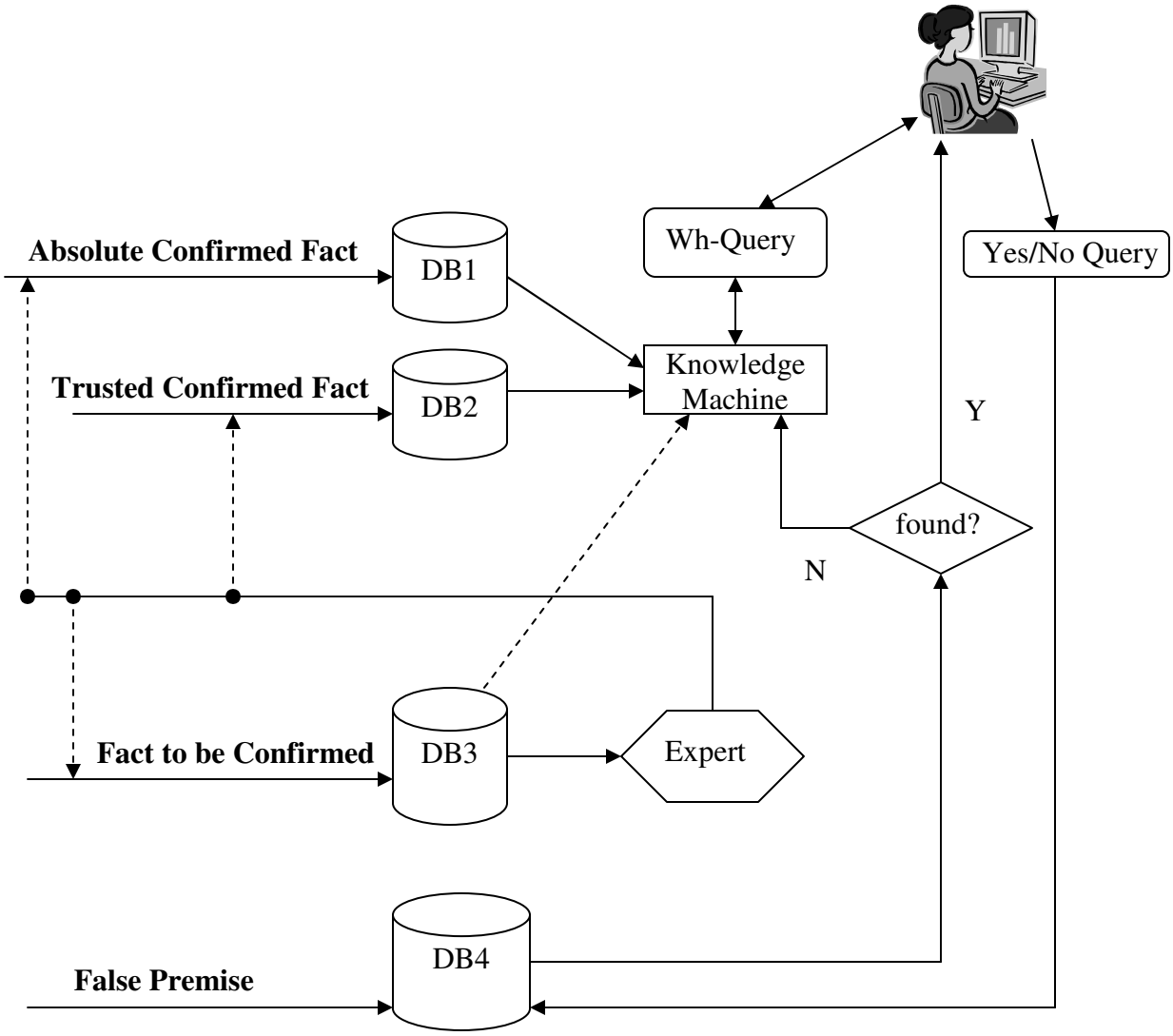


Figure 5- 8 Knowledge Collection of Natural Language for Users' Query

Time-varied facts must be considered with the confirmation of facts when the search engine acquires them for knowledge collection. The facts that are not varied by time, called constant facts, are stored in a usual two-dimensional knowledge database. The facts that are varied by time, called instant facts, are either stored in temporal knowledge database, or, instead of storing them, referred to the site that maintains records.

Temporal, Static and Snapshot Data

Time must be considered an integral part of the data value being time-stamped. TQuel [Snodgrass] uses START and END as attributes of time in a tuple. The query works by using a valid clause to specify the START and END values of a retrieved tuple that causes a problem of integrity of temporal information such as joining non-temporal data (that does not maintain START and END attributes) with START and END attributes.

Static data is defined as a constant regardless of time. Temporal data does not become static by merely erasing timestamps from it, but removing timestamp from a value actually increases its period of validity which may not be present in the database. In another words, static data is one case of temporal data. However, for simplicity of the search engine, the knowledge databases must be separated into two databases, constant-fact database and instant-fact database, or temporal database when dealing with the time-related facts. For a snapshot value, such as an attribute in non-temporal relation, is also another case of temporal data whose period of validity is a single instant of time. In work of [Gadia], the difference in the functionality of temporal, static, and snapshot data simply disappears by the degeneration of his model when all data is valid at a single instant of time.

Keyed Temporal Relations

Union of relations R and S is a set. Each pair of tuples of relation R and S is collapsed and must agree on all key attributes into a single tuple. Note that if the two tuples being collapsed together have different values in some non-key attributes at the same instant of time, then this may give a run time error. Therefore, the relations R and S

should be changed the key before R union S is computed. The run time error will not rise if all attributes in scheme is the key.

Difference and Intersection operations work the same manner as non-temporal relation but tuple τ must agree on the key attribute.

5.5.3 Questioning Knowledge Machine

A question, or query, is a statement expecting to receive information in return. There are two main types of questions:

- Yes/No questions: Was George Washington born on 1732? A Yes/No question is similar to an inform statement, except that it starts with an auxiliary verb followed by a subject. It is an inversion of subject and auxiliary verb, except for the Do/Does/Did auxiliary verb, to converse a question to an inform statement and vice versa. For example the inversion of the auxiliary verb, “Was” and the subject “George Washington” of the question “Was George Washington born on 1732?” is the conversion from an inform statement “George Washington was born on 1732?”. If a question starts with the Do/Does verb, there is no need for inversion between the subject and the auxiliary verb to converse to an inform statement. Ignoring the Do/Does verb, the question becomes an inform statement. For Did verb, the question becomes an inform statement by taking out the Did verb and changing the verb located after the subject into the past tense.

- Wh-questions: When was George Washington born? A Wh-question is a sentence starting with an interrogative pro-form -what, where, when, who, why, and how- that expects a noun phase as an answer. The inversion of an auxiliary verb, except

Do/Does/Did verb, and the subject transforms a wh-question into an inform statement with one or more have-to-find value x. For example, a question “When was George Washington born?” can transform into an inform statement “George Washington was born x.” where x is a have-to-find variable. Similar to Yes/No question, the Do/Does/Did verb does not need the inversion but the tense conversion must be considered.

After the conversion of wh-question into an inform statement, the semantic of the interrogative pro-form must be combined to find the answer, fact, of the question. This is explained in the next section.

Besides starting with interrogative pro-form of question statements, a query may present in a command, or request, statement such as “Show me the year George Washington was born”. Other examples of these verb phases are “to find”, “to tell me”, “to give me”, etc. Some command queries include the wh-word such as “Show me when George Washington was born”. For this case, the knowledge machine will ignore the “Show me” phase and its command phase will be left with an inform statement with the interrogative pro-form, “when”. For the case of no wh-word “Show me the year George Washington was born”, the semantics of the “year” must be performed at the knowledge processing level.

Another form of query contains a wh-word but the semantic of the query does not intend at the wh-word, “What time and place was George Washington born?” The user does not intend for the semantics of “What”, rather “What time” that is “When” and “What place” that is “Where” instead. Therefore, the knowledge database must maintain synonyms of wh-word for the knowledge processing unit to process such queries.

5.5.4 Knowledge Processing of Intelligent Search Engine

Regarding the semantics of the question, the knowledge processing unit retrieves answers from the knowledge databases, either constant or temporal databases if the semantics of the question is clear. Otherwise mandatory retrieval on both constant and temporal databases is needed. In the second case, the results from both types of databases are combined and the process may take longer if there is no parallel execution. Therefore, it is important that the semantics of query must be clarified to reduce the overhead of the knowledge processing. Re-phasing the query into one of two main forms of question and confirming from the user may be necessary. However, repeating confirmation from the user may be unpleasant to the user.

Query Processing on Temporal Databases

The output relation of a query is produced by the following steps.

1. The constants and free variables appearing in the window match the value in tuple components of each corresponding attribute of relation instances.

2. Fixed variables are bound and matched to the corresponding free variable.

3. The expressions in condition boxes are evaluated.

4. A tuple that satisfies the condition is retrieved from the database to the output items.

If a simple-valued variable appears in the column of a set-valued attribute, the variable matches any member of any set value of that attribute. If a variable occurs at several columns in a window, all occurrences match the same value.

When a variable and an attribute in which the variable appears as an example element have the same type, then we say that the variable is compatible with the attribute. Otherwise, it is incompatible. [Tansel]

If query is invoked by citing its name in the output box of another subquery, the result of the query is placed as a set-valued (set-triplet-valued) attribute value in that box. In this case, the query should have a single-column output. Such subqueries are used in forming relations with set-valued or set-triplet-valued attributes.

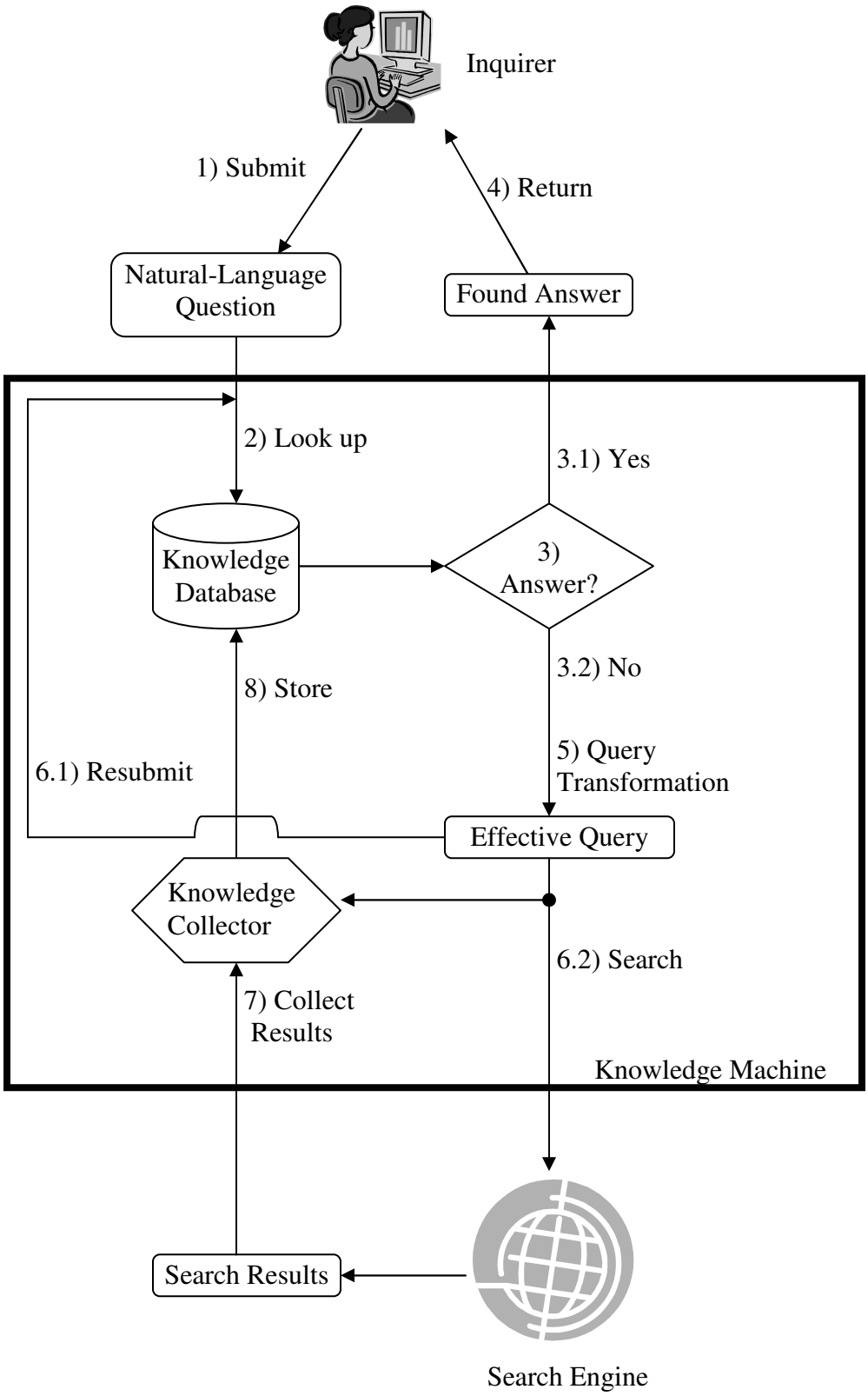


Figure 5- 9 Query Processing at Knowledge Machine

Starting when an inquirer submits a question in natural language, the question is converted into an inform statement that will be queried from the knowledge database categorizing information by the subjects. Depending on the type of wh-question, and other main phase, such as subject, object or verb, the knowledge database is queried by categories of the area under discussion. Assuming the expertise of the knowledge database, we called this step a look-up process because most information has probably already been stored in the knowledge database. If the answer(s) is (are) found, it (they) will be sent to the inquirer. Otherwise, the natural-language query is transformed into effective query [Agichtein].

Query transformation sends the effective query to two sources:

- search engines to search with terms and phases expected to appear in documents containing the answer of the question. The search engine for natural language applications [Cafarella] can be applied to the system.
- knowledge database to look it up again with a better query than a natural-language question. If there is no answer found at this time, the knowledge machine will wait and then process again after the knowledge collector returns new knowledge from the search engine.

The search results returned by search engine(s) are whole document(s) of related terms in the effective query but not yet the answer of inquirer's question. The knowledge collector analyzes the document with the effective query to generate inform statement(s) of related terms to the effective query. These inform statements are sent to the knowledge

database to permanently store for finding the current answer, and later use for coming question that is the same.

5.5.5 Intelligent Search Engine for Medical Network

Intelligent search engine should apply to any field or all fields together in one search engine. The problem of information retrieval of the knowledge processing in medicine is not infeasible and is explained in the next section. Information in medicine is tremendous but countable and grows at a slow pace. The amount of knowledge is acceptable to store and computable to process. Many researchers and businesses implemented databases of diseases and medicines as documents, but not for answering a specific question. It is quite challenge to store information that can directly answer a specific question even though only about medicine, because the same inform statement can be asked by different forms of questions. Consider an inform statement “A symptom of more advanced lung cancer is a new cough or a cough that does not go away.” Many questions can be answered from it, for example, “What is a symptom of lung cancer?”, “What is a symptom of advanced lung cancer?”, “What are diseases of cough symptom?”, “Is a chronic cough symptom of lung cancer?” and etc. Therefore, the knowledge of medicine and natural-language parsing must be considered together when designing the knowledge database.

The knowledge of medicine is classified into such categories as definition, symptoms, and methods of cure of diseases. The knowledge database can be designed according to categories and the type of questions, such as where, and who. Therefore, the databases for the knowledge machine must be set into two groups, the medical

knowledge databases and natural-language parsing knowledge databases [Agichtein]. A user question is parsed by using the second group of databases to generate an effective query for the outer search engine or SQL query for the medical databases. To accelerate information retrieval for particular questions, information may be partially redundant if necessary. Although redundancy is not acceptable in database design, the knowledge in medicine is likely unchanged. So the likelihood of the need to update and to delete anomaly is very small.

5.5.6 Social Impact of Intelligent Medical Search Engine

Applying the intelligent search engine to medical network is suitable and serves a public benefit. There are many places in the world that still need to improve the health standard. The Internet is already available to the general public worldwide, but medical knowledge is usually limited to individuals with medical expertise. An intelligent medical search engine by computer scientists can become a bridge between them.

Nevertheless the drawback of this intelligent search engine is that a user does not have a chance to read through the whole document to find out the answer as implemented by most search engines. So the user will not grow his/her knowledge about the topic of the query, and it may not be helpful for advanced medical research. However, the short and precise answer would be more desirable for:

- patients who want to find out about the disease as soon as possible for their personal understanding and do not intend to become an expert in medical field.
- medical experts who already understand most of the material of their query but need only a small missing piece of information.

- emergency personnel who need an immediate answer when faced with an urgent life and death decision.

Because of those needs, the intelligent medical search engine should be adopted for research and, later, implemented for more general use to advance medical knowledge in the general public.

5.6 Conclusion

Applying the intelligent search engine as answer-the-question into medical network is reasonable to implement. Inform statements in medical and related documents are collected into four levels of facts, absolute confirmed fact, trusted confirmed facts, unconfirmed facts, and false premises for levels of confidence into knowledge databases that later queried and evaluated the results by knowledge machine. Temporal databases could be deployed for the time-stamped data. Parsing a user question in natural language into an inform statement is done by using the natural-language parsing knowledge databases with a parser before retrieval from the medical databases or sending to search engines in case of unfound answer in the current medical databases. The medical knowledge that used to be limited to a group of medical experts can be widely queried and distributed to a more general audience and can improve knowledge of health issues worldwide.

Applying Deductive Distributed System to Other Fields

After the long study in the deductive distributed system while many events occurred during the time, such as SARS and Bird Flu epidemic, September/11 and chaos

in middle east, Katrina and Tsunami disasters, and other events, we can apply the deductive distributed system, *not only to medical system, but also in other areas such as disaster relief, domestic investigation, military strategy, e-commerce, e-travel and others*.

***Disaster Relief*:** The Katrina efforts has shown the results of unprepared and poor coordination of US organizations in both pre-event such as the readiness of warning system and meteorologist and post-event such as medical and disaster release teams, and rebuilding efforts. The knowledge of these organizations can be shared and jointly analyzed using the concept of MKT network.

***Domestic Investigation*:** After the 9/11 attacks, the computer technology has been expanded to almost every local and federal security institute. The domestic security departments and law enforcements have been linked together and administered under the department of homeland security that includes coast guard, secret service, military liaison, emergency management agency, transportation administration, customs services, immigration services, etc. [12] whose data are linked together. Each department still maintains local information that can be shared by retrieval but not for intelligent deduction. The deductive processes among agencies and departments can utilize these resources for faster investigation. With the confident level, discussed in section 5.3, the accuracy of deduction will provide better decision before action is taken.

***Military Strategy*:** Similar to the domestic investigation, military services can cooperate their comprehensive combat information among military units to plan and adjust tactic and strategy before, during and after a combat. The chance of injuries or losses would be reduced if the decision is made based on the high confident level of success of cooperation, not on the risk of individual unit. Facts gathered among units can

be linked on distributed architecture and retrieved to other units by simple query processing method on distributed databases.

***E-Travel*:** It is not to travel on the internet but to join travel agency together. The current webpage already provides the complete reservation of hotel, flight, rental car, and cruise together but not to the main destination of the trip such as tour at Paris or inside the Versailles palace. So the travelers can choose an arbitrary plan from one place to another with multiple agencies. It could reduce cost for the travel agencies and increase their revenues.

***E-Commerce*:** Similar to the E-Travel, commercial companies can share their information and use the similar method of medical analysis to analyze the demand of customers of the coordinated organizations. Then, the company can improve its strategies. However, due to sharing effort, the security of companies' information and customer privacy must be considered.

***Others*:** Organizations can share and distribute their knowledge among their peer organizations, so their users can directly retrieve information from many organizations and benefit from the use of the coalesced information. For the organizations, satisfying their user is the goal.

Chapter 6 Simulation Results of New Methodology for Deductive Medical Databases

6.1. Program Structure

The novel methodology of this research is divided into four sets of programmability; one local knowledge node, two near-proximity knowledge nodes, two far-proximity knowledge nodes, and three combinations of near and far proximity knowledge nodes. Each of these sets of programs is classified into four steps according to medical deduction analysis; Patient Given Symptoms, Mark Symptoms, Mark Causes, and Disease Conclusion, and two steps to randomize the simulation; Test Preparation and Result Collection. The methodology was simulated by four geographical zone; local, regional, national and global testing.

6.1.1. Steps of Medical Deductive Analysis

Four steps of medical deductive analysis are explained in the following as for the general procedure comparing to medical practice. These four steps will be further discussed in the next section as for the matter of programmability of the test.

- Patient Given Symptoms The first reason for a patient to visit a doctor is the suspiciousness of symptoms occurring to his/her health. At first, the physician asks about the symptoms known by the patient. So, the first step of programmability allows patients to give out the known symptoms to the system. The program uses these symptoms to create a query to the knowledge node to find likely diseases that usually showing all these symptoms. These first-step diseases will be shown to the patient with the first layer of

confidence level[1]. Using these likely diseases, the Patient Given Symptoms Step creates another query to the knowledge node to retrieve their other symptoms to show to the patient in the second step.

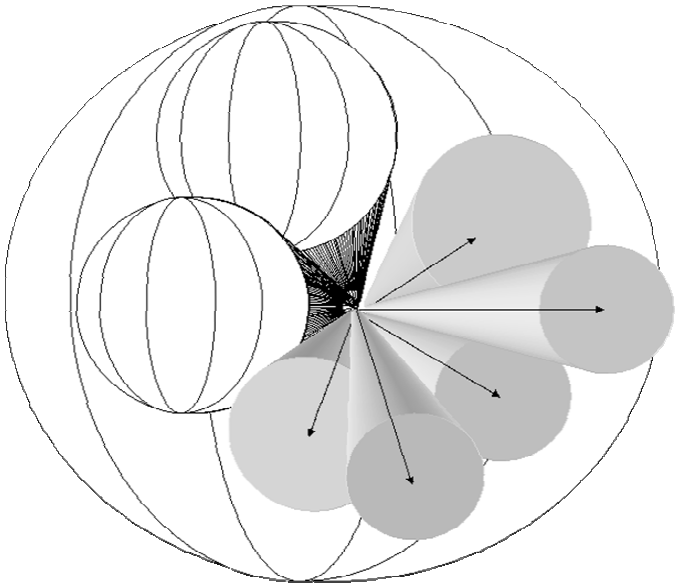


Figure 6- 1 Sphere at Given Symptom Step

Figure 6- 1 demonstrates the medical analysis by using a sphere that represents scope of illnesses. Inside the outer sphere, it contains diseases, symptoms, causes, method of treatment, and other matters related to illnesses. Each inner sphere represents a disease. In Figure 6- 1, there are seven diseases matching the patient symptoms. Each arrow represents confidence level of its disease.

- Mark Symptoms The program shows the list of other symptoms retrieving at the Patient Given Symptoms Step to the patient into two sets; the common symptoms that are likely to occur to all members of the likely disease list, and the uncommon symptoms that are not likely to occur and are shown separately. The patient will mark these new symptoms occurring to him/her. The program uses the combination of symptoms from

[1] The confidence levels of the first, second, and third layer are explain in Chapter 5 section 5.3.3

both steps; Patient Given Symptoms and Mark Symptoms, to generate a query to find the likely diseases[1]. The likely diseases are shown to the patient with second layer of confidence level explained in chapter 5. Using these new likely diseases, the program creates a query to the knowledge node to retrieve their causes to provide to the patient in the next step.

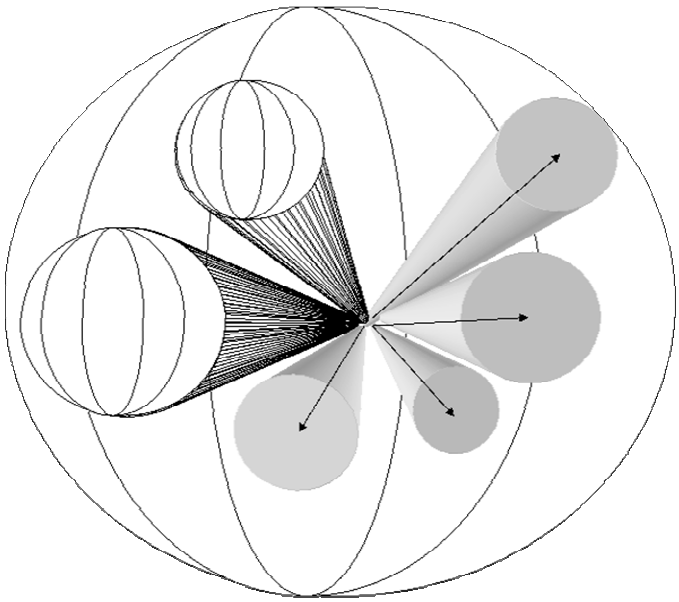


Figure 6- 2 Sphere at Mark Symptom Step

After user marks symptoms, some disease may be eliminated from the list of likely diseases. So the number of inner spheres in Figure 6- 2 is reduced from Figure 6- 1; one sphere disappears. The size of most spheres may be reduced because unmatched symptoms of the disease are eliminated. Some sphere may remain unchanged in size. When the confidence level of a disease is higher, the arrow, representing the confidence level, from the center to the inner sphere of the disease is longer. The arrows of some diseases in Figure 6- 2 may remain the same as in Figure 6- 1, or may become shorter, depending on the change of confidence level at the mark symptom step.

[1] The number of likely diseases at the second step should probably be less than the number of likely diseases at the first step because more symptoms were provided to the system.

- Mark Causes The causes of new likely diseases are shown at this step. The patient will mark the causes that he/she recognizes from the list[1]. The causes and their likely diseases of the patient are submitted as a query to knowledge node to find the concluding diseases with the confidence level.

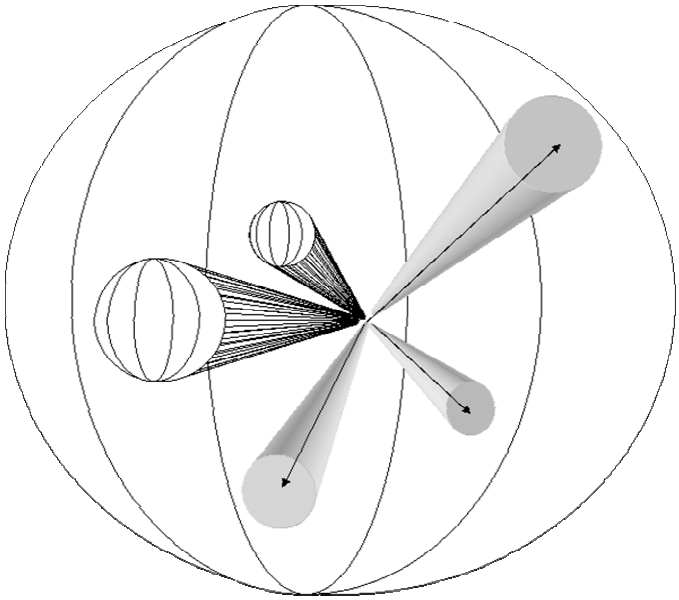


Figure 6- 3 Sphere at Mark Cause Step

The causes that are not related to the patient are eliminated and so the size of the inner spheres representing likely diseases in Figure 6- 3. The length of the arrow denotes the level of confidence; the longer the arrow, higher the confidence level.

- **Disease Conclusion** Not only the concluding diseases are queried from the knowledge node to summarize to the patient, but also the method of treatment which the patient can do by himself/herself or through further contact with physicians. The recommendation or prescription of medicine can also provide to the patient[2].

[1] At this step, the patient may not recognize any cause occurring to him/her. The number of mark causes can be zero.

[2] The medicine recommendation and prescription are not provided in this research due to the regulation of medical practice.

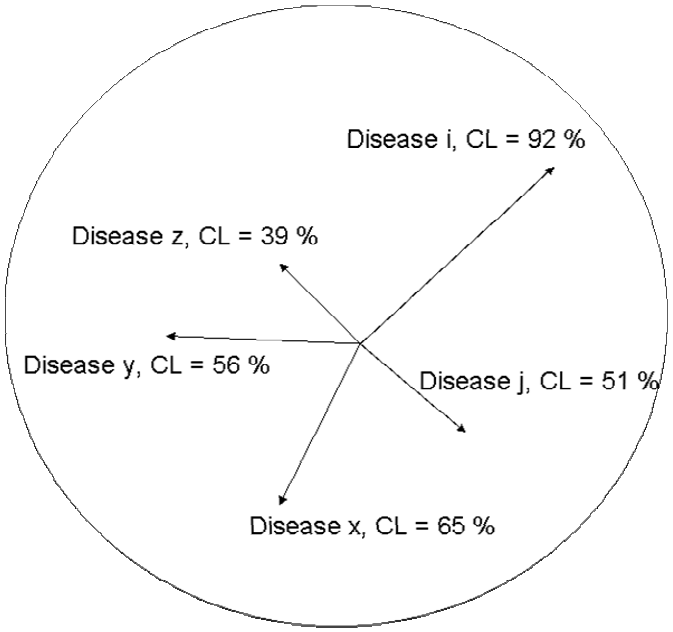


Figure 6- 4 Sphere at Disease Conclusion Step

At the last step, the user (patients or physicians) of the medical analysis program will be given the conclusion about which diseases are likely to occur to the patient with a confidence level as show in the conclusion sphere of Figure 6- 4

6.1.2. Six Steps of Simulation of Medical Deductive Analysis

Extending from the general four steps of medical deductive analysis, this section discusses about the programmability of the simulation. Two additional steps are needed; Test Preparation and Result Collection.

- **Test Preparation** Because the simulation is automated and not required a test runner who is ill, the program generates a randomized disease for each time that test

runner starts the test. The random disease is from the knowledge node[1]. The program was written in C program on the server at Brooklyn College to:

- randomize a disease
- retrieve all symptoms of the random disease
- send these symptoms to the user's computer
- generate a dynamic JavaScript program that can run on the computer of the user (or runner) to random symptoms.

- Patient Given Symptoms The JavaScript program randomizes patient symptoms whose number of randomized symptoms is also random between one to three symptoms. The execution is started at this point. The test preparation is not a part of execution time for the medical deductive analysis. The JavaScript program will send the randomized symptom as "Patient Given Symptoms" to the knowledge node to analyze the symptoms and query the knowledge node to find a list of likely diseases and their symptoms. The analyzing program was written in C program on the server side at Brooklyn College to:

- find diseases having the patient given symptoms
- retrieve all symptoms of the found diseases
- show the diseases with confidence level of the first layer and their symptoms into two sets, common and uncommon symptoms to the user in the HTML format presenting to the browser of the user

[1] Multiple knowledge nodes are further discussed in the next section.

- generate a dynamic JavaScript program with these new symptoms to random on the user computer in the next step.

- Mark Symptoms The JavaScript program randomizes these new symptoms in the same as a patient marking symptoms from the list. The number of randomized marked symptoms is also random depending on the number of symptoms of the original random disease. But the marking symptoms are not included in the patient-given-symptoms. The new random symptoms are sent to the server with all previous information; the random disease, likely diseases, random patient-given-symptoms, and starting time. A C program on the server analyzes these new symptoms with the patient-given-symptoms and likely diseases to:

- refine the likely diseases from two list of symptoms; patient symptoms and marked symptoms
- retrieve all cause of the new refine likely diseases
- show the new likely disease with confidence level of the second layer[1] in HTML format
- give choices of causes of these new likely diseases in HTML format
- generate a dynamic JavaScript program with these causes

- Mark Causes This JavaScript program is similar to the JavaScript program at the Mark Symptoms step. Instead of randomizing symptoms, the program randomizes causes from the list and sends them to the server. The program on the server receives the

[1] Explained in the section 5.3.3

starting time, the random disease, new likely diseases, random patient-given-symptoms, random mark-symptoms, and random causes. The sending information is used in the program to:

- analyze and conclude the final diseases
- retrieve the methods of treatment of each concluding disease
- show the concluding disease with confidence level of the third layer and their methods of treatment in HTML format
- generate a JavaScript program

- **Disease Conclusion** At this state, the patient receives the conclusion and the simulation for the deductive analysis is finished. The JavaScript at this step determines the finishing time that will be subtracted by the starting time for finding the total execution time. All running data will be sent to the final step for collection. By the same time, the JavaScript called a PHP program located in a server to determine the IP address of the computer of the test runner. This IP address determines the location of the test runner for the analysis in section 6.2.

- **Result Collection** All results are collected by a program that creates an insertion query to submit into a database server at Brooklyn College. All collected results are:

- starting time from the Test Preparation step,
- ending time from the Disease Conclusion step,
- simulation date from the Disease Conclusion step,
- random disease ID from the Test Preparation step,

- a list of final likely diseases analyzing at the final conclusion step,
- a list of patient-given-symptoms from the Patient-Given-Symptoms step,
- a list of marked symptoms from the Mark Symptoms steps,
- a list of marked causes from the Mark Causes steps,
- IP address of the test runner

Other data such as city, country, and computer specification of the test runner, and comment are recorded manually after the runner completes the tests.

6.1.3. Sets of Programmability on Multiple Knowledge Nodes

Programmability is defined into four sets regarding to the number of knowledge nodes; one, two near-proximity, two far-proximity, and three knowledge nodes. Each of these sets contains all six steps of simulation of medical deductive analysis explained in the previous section. The major difference among these four sets is the medical analysis on multiple medical knowledge nodes.

- **Single medical knowledge node** The medical knowledge node locates in a database server at Brooklyn College, Brooklyn, NY. The medical knowledge database is shown in the following figure using the Entity-Relationship model explained in chapter 2.

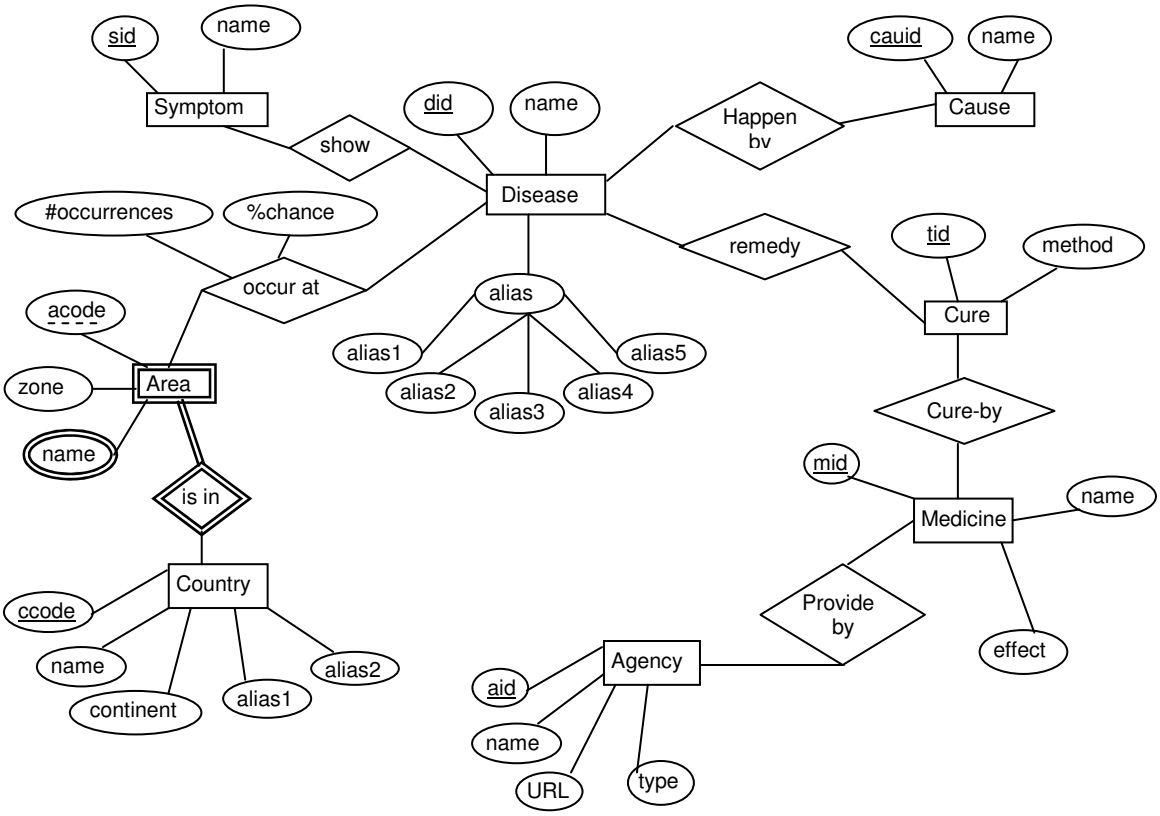


Figure 6- 5 The ER diagram of Medical Knowledge Database at Brooklyn Nodes; sid is a symptom identification number, did is a disease identification number, cauid is a cause identification number, tid is an identification number of the method of cure, mid is a medicine identification number, aid is an identification number of an agency providing the medicine, acode is an area code according to the public telecommunication system, and ccode is a country code defined for global telecommunication system.

The main entity of the medical knowledge structure in this research is the diseases. One disease may show many symptoms, happen by many causes, remedy by many methods of treatment, and occur at many places worldwide. For simplicity, we limit the number of different names of a disease to only five aliases. All this information is used for the medical analysis in our research. However, as shown in the ER diagram, two sets of entity are not deployed and used in the programmability of our research, the medicine and the occurrence of diseases.

- Due to academic research, the medicines and their providers require the involvement of pharmaceutical companies with their business issues and copyrights.
- The occurrence of diseases is a ready effort for the future research that will study about the global and quick contagious diseases such as SARS, and bird flu. For this research, the occurrence of diseases can be shown at the disease conclusion step to the patients as details, adding to the method of treatment, of the final concluded disease.

For one knowledge node, the medical analysis follows the six steps of simulation of medical deductive analysis. The Brooklyn knowledge node provides the medical knowledge of diseases, related symptoms, causes, and methods of treatment. The random values such as disease identification numbers, symptoms, and causes are retrieved only from the Brooklyn node. After the conclusion is made, the result is entered into a database in Brooklyn node. Because the medical analysis was finish at the disease conclusion step, the time of result collection is not added to the execution time.

The results of the tests on single medical knowledge node include the transaction numbers, date and hour of executions, the length of execution times, random disease identification numbers, and a list of likely diseases at the conclusion. The result of the test on single medical knowledge node, that is a stand alone as central knowledge node, is used to compare with two and three medical knowledge nodes that are distributed. The raw results of single medical knowledge nodes are in Appendix A2, B1, C1, and D1. The

comparisons of the results of single medical knowledge node are shown in Network Geography Simulation sections 6.2, 6.3, 6.4, and 6.5.

- **Two Medical Knowledge Nodes** The tests are separated into two sets: Near-Proximity and Far-Proximity Medical Knowledge Nodes. For Near-Proximity test, the medical knowledge nodes are located in a database server at Brooklyn College, Brooklyn, NY and at BizHostNet.com, Islandia, Long Island, NY whose distance between the two nodes is forty two miles. The medical knowledge nodes of Far-Proximity test are at Brooklyn College, Brooklyn, NY and at FreeSQL.org, San Francisco, CA whose distance between the two nodes is twenty five hundred and seventy two miles[1]. The structure of medical knowledge database at BizHostNet, Long Island, and FreeSQL.org, San Francisco are similar to the structure of medical knowledge node at Brooklyn College in Figure 6- 5, but, without the occurrence records of diseases. For future research, the medical analysis will include the occurrences of disease for the analysis.

The tests of two medical knowledge nodes follow the six steps of Simulation of Medical Deductive Analysis, as in the tests of single medical knowledge node. On the contrary, two nodes provide medical knowledge for the medical analysis. To randomize disease, the program at the test preparation step randomizes which node either Brooklyn or Long Island nodes for near-proximity tests (and either Brooklyn or San Francisco nodes for far-proximity tests) contains the disease of the patient. Using that randomized node, the JavaScript program will randomize the random disease in the test preparation step, the random symptoms in the Patient-Given-Symptoms and Mark-Symptoms steps, and the random causes in the Mark-Causes steps. However, during the medical analysis,

[1] The distance between two locations is according to www.findlocalweather.com

the random node will be hidden from the program of medical analysis. The program must follow the medical analysis to find out the random disease as the situation that physicians do not know in advance which illness(es) the patient has.

Same as the result collection, the time for randomization is not included in the medical analysis. After finishing the tests, the results are collected in a database at the Brooklyn node. The results of the tests on two medical knowledge nodes include the transaction numbers, date and hour of executions, the length of execution times, random disease identification numbers, a list of likely diseases at the conclusion, and whether the conclusion is from two knowledge nodes (or just found the conclusion from only one knowledge node). The last result, number of nodes found the disease, and its execution time are used to compare between near-proximity and far-proximity tests and between single and two medical knowledge nodes. The raw results of near-proximity tests are in the Appendix A3, B2, C2, and D2 whereas the raw results of far-proximity tests are in the Appendix A4, B3, C3, and D3.

The comparisons between single, two and three knowledge nodes are shown in the following section at three knowledge nodes. Whereas, the comparisons of the results of two medical knowledge nodes are also shown in Network Geography Simulation sections 6.2, 6.3, 6.4, and 6.5.

- Three Medical Knowledge Nodes The last synchronized tests combine all three knowledge nodes together; a database server at Brooklyn College, Brooklyn, NY, at BizHostNet.com, Islandia, Long Island, NY and at FreeSQL.org, San Francisco, CA. The

structures of medical databases of three knowledge nodes are the same as in the tests of two medical knowledge nodes.

The simulation randomizes the nodes containing the patient disease to choose from one node of three nodes. Same as the tests on two medical knowledge nodes, the randomized node will be used to random the symptoms, and causes. But three medical knowledge nodes analyze the illness from their medical knowledge to find the conclusion. The confidence level of multiple medical agencies is provided according to the number of nodes, symptoms, and causes found for the random disease that is hidden from the program during the medical analysis.

The results of the tests on three medical knowledge nodes include the common values as in single and two medical knowledge nodes; the transaction numbers, date and hour of executions, the length of execution times, random disease identification numbers, and a list of likely diseases at the conclusion. Since the test is associated more than two nodes other values are included in the results; the number of knowledge nodes found the disease matching to the patient circumstances, list of knowledge nodes found symptoms of patient, and list of knowledge nodes showing the final disease(s) at the conclusion. The raw results of the tests on three medical knowledge nodes are in the Appendix A5, B4, C4, and D4.

6.1.4. Geography of Simulation

Geographically, there are four categories of zones of simulation; local, regional, national and global for location of clients running the real-time simulation. The zones are distinguished by the distance from the main knowledge node and testing programs at

Brooklyn College. Clients on each zone used different sets of hardware specification such as CPU and memory of computers running the test, and Internet connection.

The local geography includes client nodes that are located a very short distance, less than one thousand yards (about one kilometer). For our research, all client nodes in the local geography are located in Brooklyn College where all programs were implemented. The tests were simulated in five different Local Area Networks (LANs); Atrium, Sci, Library, Library Café, and Faculty Lab. The detail of LANs at Brooklyn College is in the section 6.2.1, Topology of Local Network.

The regional geography is composed of clients that are in the same city. For this research, the client nodes for the regional geography are located in New York City, NY whose radiuses from our main knowledge node and testing program are less than 20 miles. The client nodes of our regional geography are located in Woodside Queens, Flushing Queens, midtown Manhattan, and downtown Manhattan. The detail of each client node is explained in the section 6.3.1, Topology of Regional Network.

The national geography includes clients located only inside the United States. The client nodes are located in three different cities, Whippany (Morristown) New Jersey, Houston Texas, and Durham North Carolina whose details are explained in the section 6.4.1. Even though the client nodes in New York City and Brooklyn College are inside the US, we do not demonstrate them in the national geography because they are categorized for the regional and local geography. However, we compare the results among different geography in section 6.7.

The last geography of comparison is global outside the United States. We include four different countries in two continents; Germany and Austria in Europe and Thailand

and Japan in Asia. Whereas our client nodes in Europe is less than five thousand miles away from the United States (at the main knowledge node, and programs in the New York City), two Asian countries are located on the opposite (or near-opposite) side of the earth from the United States.

6.2.Real-Time Local Simulation

The tests at local network are inside Brooklyn College where the main knowledge node is located. We ran the tests on various Local Area Networks (LANs) located in different locations whose details are explained in the next section, Topology of Local Simulation. The results are compared between single and multiple knowledge nodes in section 6.2.2 whereas the comparisons among nodes in local network are shown in section 6.2.3.

6.2.1. Topology on Local Simulation

The topology of local simulation is among different LANs inside Brooklyn College. This study includes five LANs located in four different buildings;

- Atrium LAN at West End Building where the main computer lab for students is located. This LAN includes about three hundred Windows PCs, one hundred UNIX SUN computers, and twenty Macintosh Apple computers (MACs) to serve students as publicity and classroom. Wireless connection is allowed for students to connect their laptops to the LAN to serve the internet and use facilities of the LAN. The main medical knowledge base is also located in this LAN.

- Science LAN at Ingersoll Hall where Computer Information Science Department is located. This network serves only for faculty and staff of CIS department. There are both types of computers, PCs and SUN computers, in this LAN.

- Library LAN is located in the Brooklyn Library building where students can use the computers for researches. 48 PCs are available in the Reference Area and 20 PCs in the Reserve Reading Room 98 PCs, and 12 MACs are available on the 2nd floor and 80 PCs in the Lower Level to all library users. In addition to the Internet and online Library resources, access is provided to Microsoft Office and other software such as statistical and graphics packages.

- Faculty Lab LAN inside the Faculty Development Lab to serve only the faculty of Brooklyn College. 11 PCs and 4 MACs are available for use by faculty and staff in the Faculty Development Lab and 32 PCs and 27 MACs in the Multimedia Classrooms connecting to this network. All computers include access to the Internet, Microsoft Office and additional software such as statistical and graphics packages. This LAN is also located in Brooklyn Library building.

- Library Café LAN operating 24 hours a day and 7 days a week for Brooklyn College students. It also allows non-Brooklyn College students to use with limited hours. There are about 80 new PCs and 12 MACs. Wireless connection is also allowed for students' laptops. Unlike library LAN, this network serves students both research and class work with various types of software. The library café is located in Whitehead Hall.

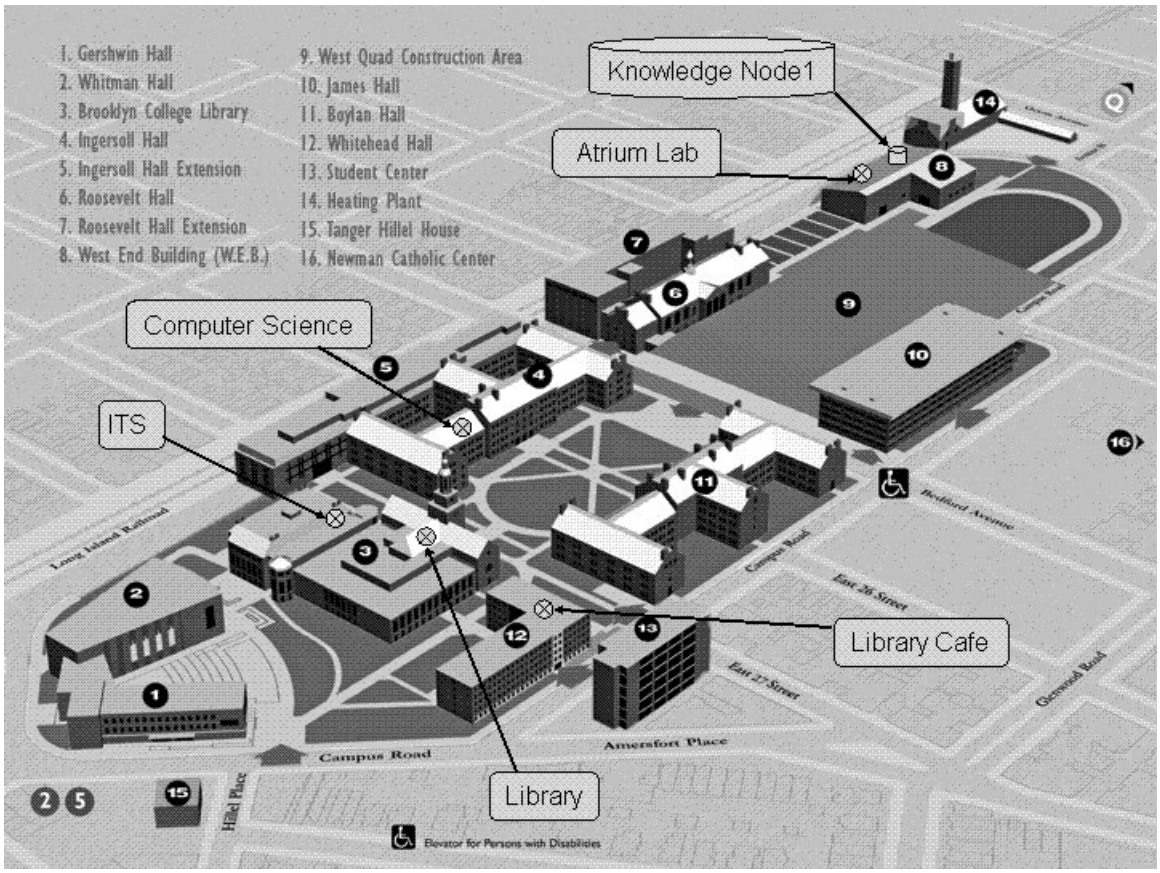


Figure 6- 6 Brooklyn College Map. Client Nodes are located in Atrium Lab, Computer Science, Library, Library Café, and Faculty Lab. The main knowledge node is located in Atrium Lab.

Various types of computers were used to run the simulation on local network such as personal computers using Windows XP operating system and SUN computers using UNIX Solaris operating system. The test at Library, Library Café, and Faculty Lab used PCs whereas the test at Science LANs used SUN computer. Because the server computer of medical knowledge is located at Atrium Lab, we ran the test on both PC and SUN computer in Atrium Lab. The characteristics of each client node are explained in the next section. Because all clients in local network are located in Brooklyn College, the distances from client nodes to the medical knowledge nodes is almost irrelevant when comparing among the local nodes. For example, the distances from any local nodes to the

medical knowledge nodes 1 are less 400 yards whereas the distances from any local nodes to medical knowledge nodes 2, and 3 are 42 miles and 2572 miles sequentially. However the distance from clients to medical knowledge nodes will change when clients are not in the same locations such as the comparison for national and global networks.

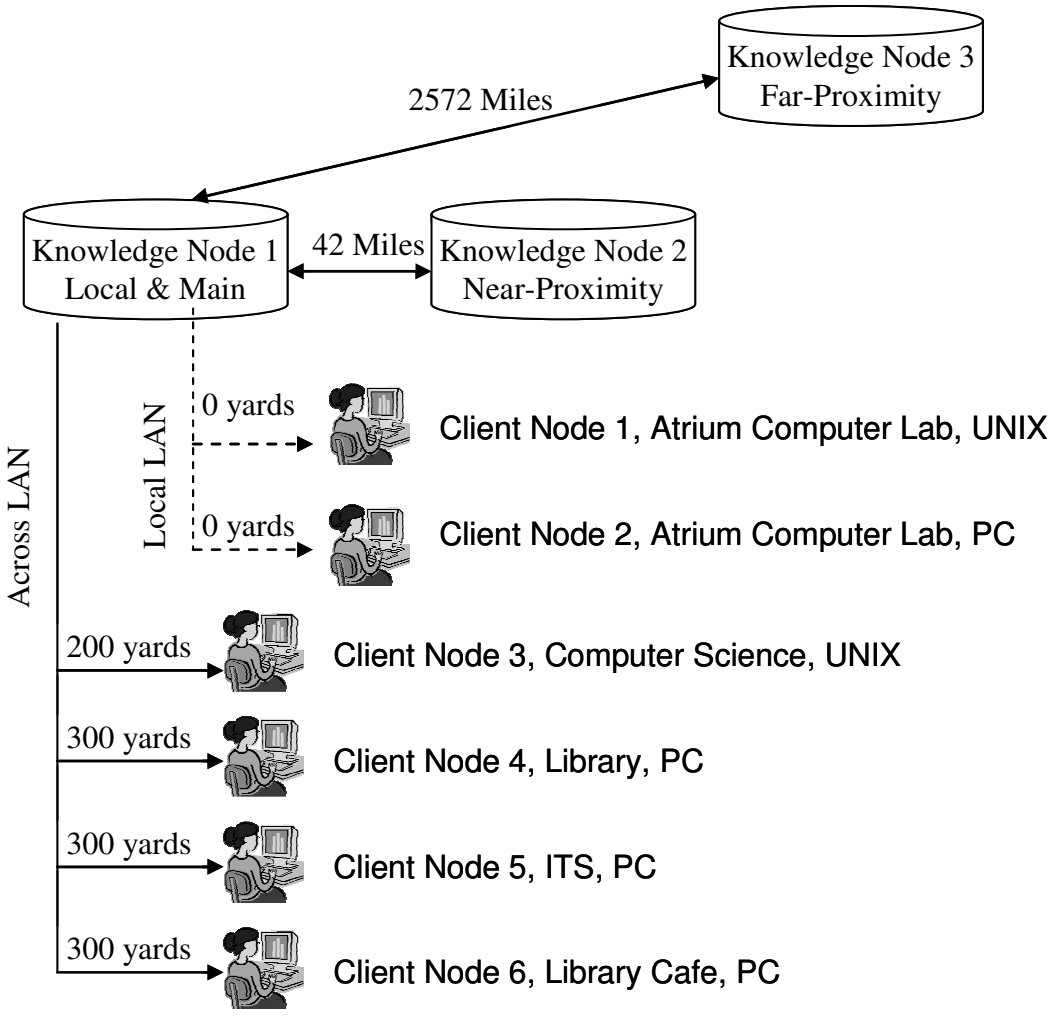


Figure 6- 7 Topology of Local Simulation at Brooklyn College

6.2.2. Results and Comparison between Single and Multiple Knowledge Nodes on Local Simulation

At the atrium lab, there are two types of computer running the test. The first set of results ran on SUN, UltraSPARC-IIe Sparcv9 processor operating at 650 MHz with 512 Mbytes of memory. The following graph shows the results of the four sets of tests on different number of knowledge nodes. It was run between 3 pm to 5pm on Sunday February 26, 2006 when network usage is low because of weekend.

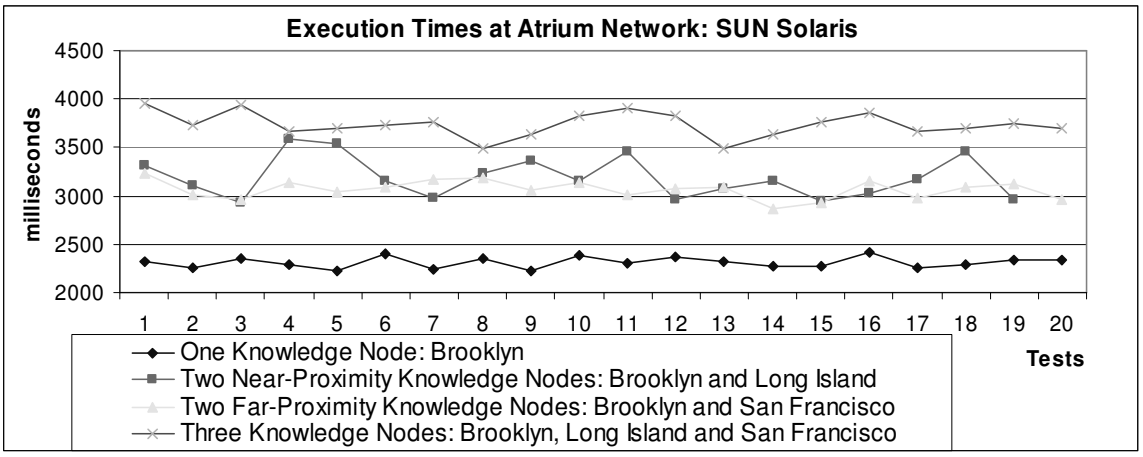


Figure 6- 8 Execution Times at Atrium Network on SUN computer of Local Simulation

Due to the low traffic of network on weekend, the result shows that almost all 20 times of tests for different numbers of nodes have similar variances, such as, for one knowledge node, the execution times are between 2,200 and 2,500 milliseconds and its variance is 102.03 ms whereas, for three knowledge nodes, the execution times are between 3,500 and 4,000 milliseconds and its variance is 119.43 ms. For two knowledge nodes, both near-proximity and far-proximity tests show the execution time in the same range, between 2,800 and 3,600 milliseconds. Their averages and variances are very close;

3,187 and 154.63 milliseconds sequentially for the near-proximity and 3,064 and 113.12 milliseconds sequentially for the far-proximity.

The second set of result running at Atrium Lab used PC Pentium IV processor at 1.7 GHz with 256 Mbytes of memory. The test was run on the same day, Sunday February 26, 2006, of the first set of the test at Atrium lab between 2 pm to 4 pm.

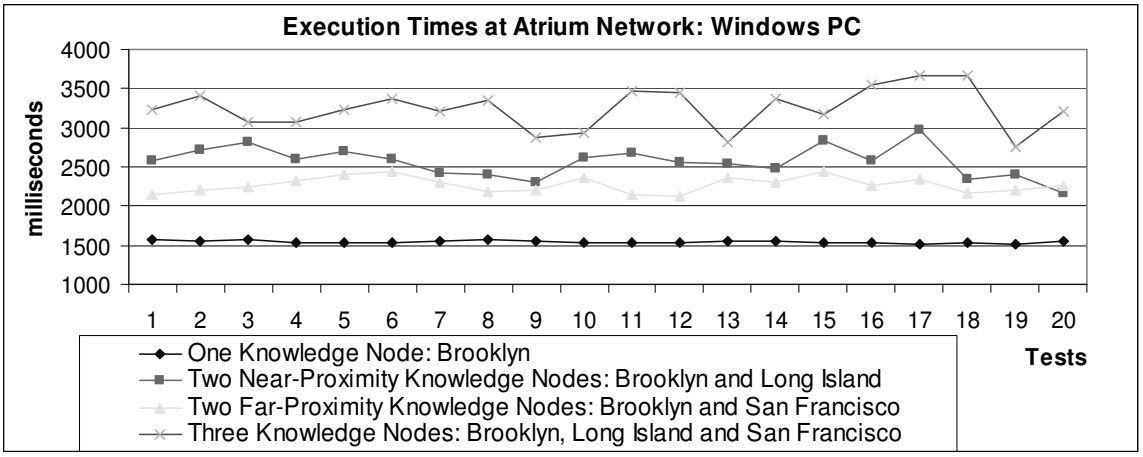


Figure 6- 9 Execution Times at Atrium Network on PC of Local Simulation

Comparing between the results of two near-proximity nodes and the results of two far-proximity nodes, the average execution times of the two far-proximity knowledge nodes whose average is 2,271 milliseconds is less than of two near-proximity knowledge nodes whose average is 2,565 milliseconds.

Twenty results of testing on one knowledge node are more stable (very close to other) for running on PC than on SUN computer in the previous graph. The memory of PC is less than that of SUN computer but, reversely, the CPU speed of the PC is almost three times faster in the number of instructions/second. Comparison of different clients on the local simulation will be discussed further in section 6.2.3

The next test was run at Science LAN of Computer Science on SUN computer whose processor is Ultra SPARC II Sparc operates at 270 MHz and memory is 128

Mbytes. It was run on the same day after the test at Atrium Lab, Sunday February 26, 2006 between 5 pm – 6 pm.

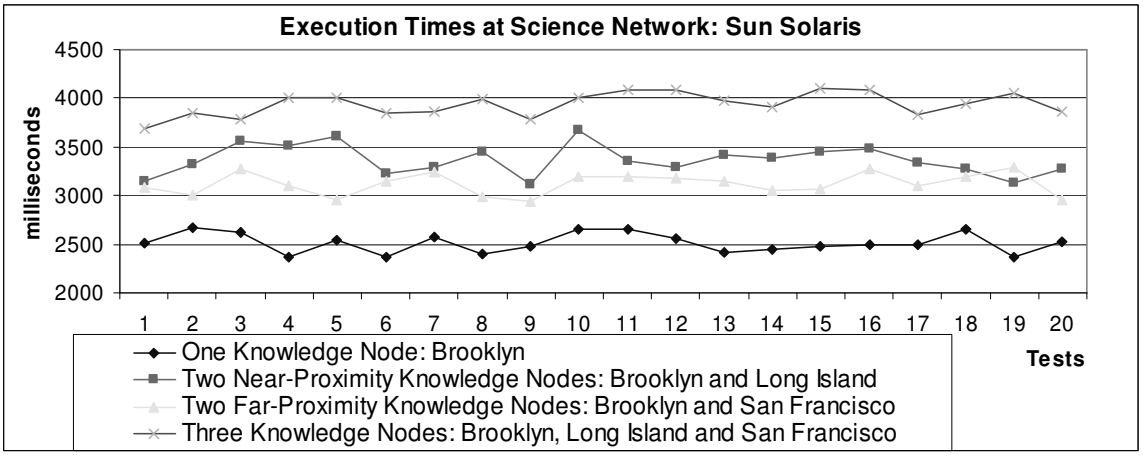


Figure 6- 10 Execution Times at Science Network on SUN computer of Local Simulation

The comparison of four sets of different number of nodes when running at Science is similar to the comparison among four sets when running at Atrium by PC. None of one-node run is higher than two-node runs and most of the execution times of two-far-proximity-node run are less than of two-near-proximity-node run. But, the result of one-node run may not be quite as stable as running by PC at Atrium Lab.

The next graph shows the results of running at Library LAN of Brooklyn College by using PC whose processor is Pentium IV 1.8 GHz and memory is 384 Mbytes. However, this test was run at weeknight between 8 pm – 9 pm of Wednesday, February, 22, 2006 when there are some students using the computer in the network.

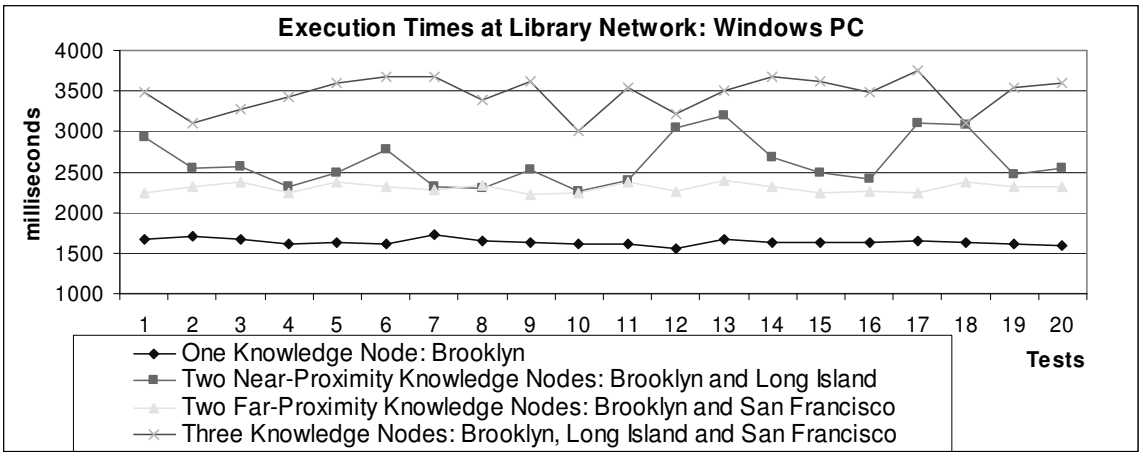


Figure 6- 11 Execution Times at Library Network on PC of Local Simulation

Results of two-near-proximity-node run are not less stable than results of two-far-proximity-node and one-node runs. However, the three-knowledge-node run always fluctuated by 500 milliseconds.

The test at library café in Figure 6- 12 used the fastest computer comparing to other tests of local simulation. The speed of the client computer at library café is 3.2 GHz of Pentium IV processor and its memory is 384 Mbytes, the same as of the client computer at the Library. This test was also run on weekend and on the same day, Sunday February 26, 2006, of the tests at both sets of the test at Atrium lab and the test at Science network. It ran between 5 pm to 6 pm.

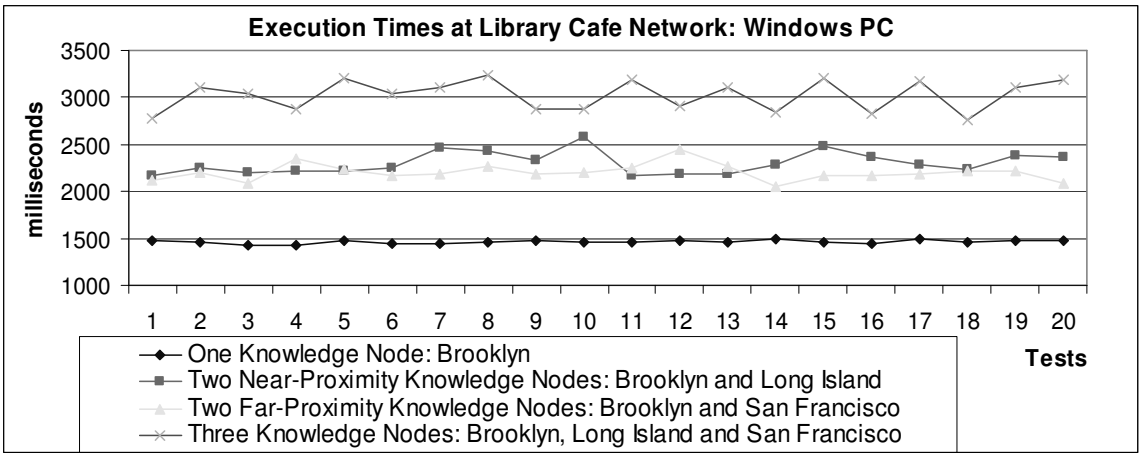


Figure 6- 12 Execution Times at Library café Network on PC of Local Simulation

The results of one-node run are very stable. Also the both results of two-knowledge-node are close of which the average execution times are 2301 and 2200 milliseconds for near-proximity and far-proximity sequentially.

The last of tests at local simulation is at Faculty Lab using PC whose processor is Pentium IV at 1.9 GHz with 256 Mbytes of memory. The test was run on weekday Thursday February, 23 2006 in the evening between 4 pm – 5 pm.

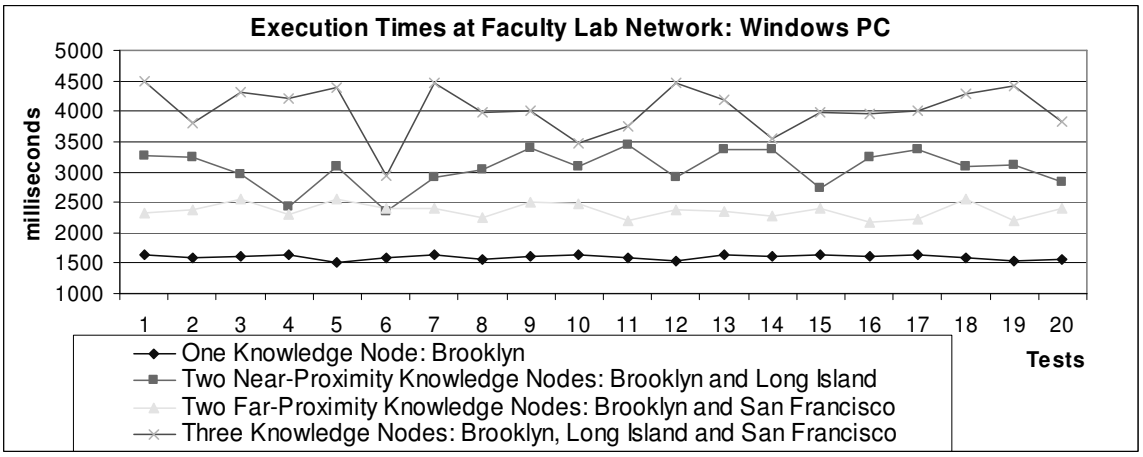


Figure 6- 13 Execution Times at Faculty Lab Network on PC of Local Simulation

Although, the results of one-node and two-far-proximity-node runs are quite stable because the lab is reserved for only faculty, the results of two-near-proximity-node

and three-node runs are broadening due to the time of execution during the weekday. The range of two-near-proximity-node runs are from 2,343 to 3,437 milliseconds, which is more than 1,000 milliseconds of difference and its variance is 304.26 ms. It is even worse in the case of three-nodes whose range of execution times is from 2,937 to 4,484 milliseconds and its variance is 396.40 ms.

The following graph in Figure 6- 14 shows the comparison of average execution time of all clients in local network together categorized by the number of knowledge nodes.

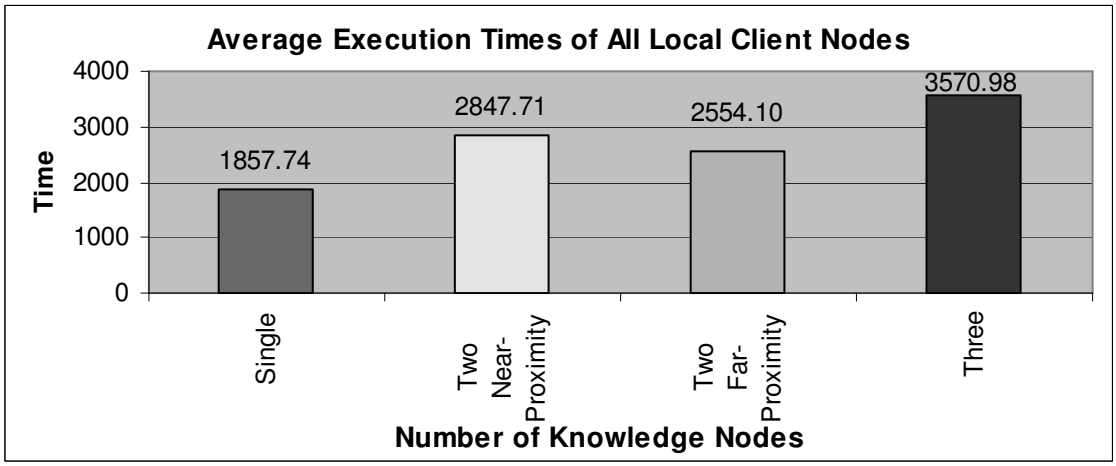


Figure 6- 14 Average Execution Times of all Clients for each Number of Knowledge Nodes at Local Simulation

The averages of all clients at the local network among different number of knowledge nodes show us that the difference between the single node and three-node is 3570.98 – 1857.74 = 1713.24 ms. But the averages between two average values of two-knowledge-node is (2847.71 + 2554.10)/2 = 2700.905 which is close to the

$$1857.74 + (1713.24)/2 = 1857.74 + 852.62 = 2714.36$$

and $$3570.98 - (1713.24)/2 = 3570.98 - 852.62 = 2714.36$$

That accumulates approximately 850 milliseconds of execution time when increasing the number of knowledge nodes by one (from one node to two nodes and from two nodes to three nodes.)

If 852.62 milliseconds is the difference of execution times between n and n+1 knowledge nodes and the average of execution time of the single node is 1857.74 milliseconds, then 1857.74 – 852.62 = 1005.12 that is approximately 1000 milliseconds should be the set up time when **not** considering the execution time at a knowledge node.

6.2.3. Comparison among Client Nodes on Local Simulation

In this section, we show the comparison among nodes in the local network considering separately each number of knowledge nodes.

The next graph shows the comparison among different nodes in the local network on the execution of one knowledge node.

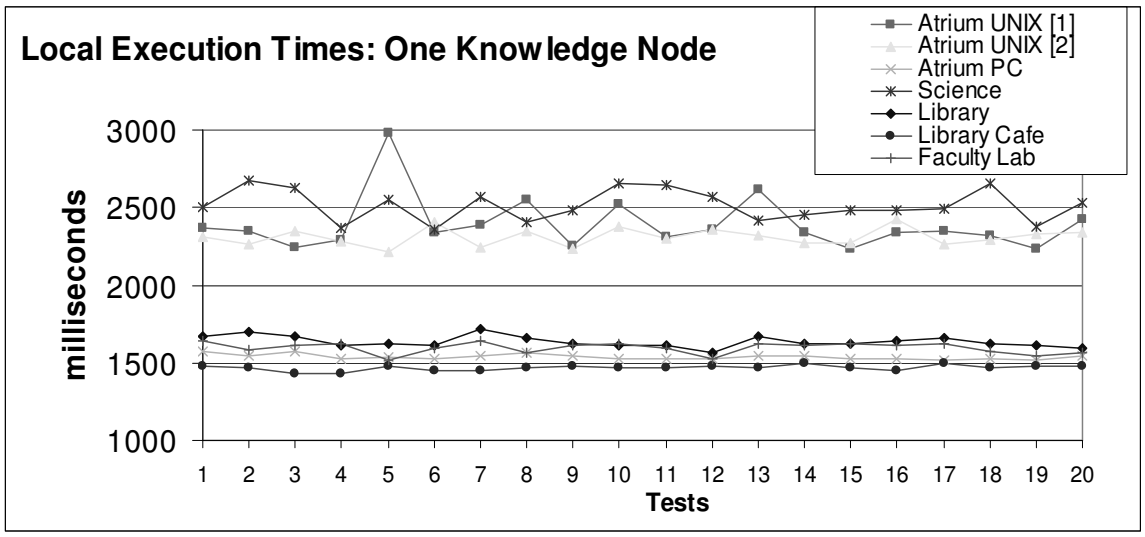


Figure 6- 15 Comparison of Tests among Nodes at Local Simulation on One Knowledge Node

For one knowledge node, twenty tests at Library Café had the least execution time compared to other client nodes. The most effective parameters of the results comparing to other client nodes of the client at the Library Café is the processor speed which is 3.2 GHz. However, the time of execution, which is in the evening of weekend, likely affects the results.

The results are divided into two groups whose execution times are divided by the 2,000th milliseconds line. However, these two groups should not be compared to one another due to the major differences in architecture; processor types and Operating Systems. All client nodes below the 2,000th ms line of execution time run on Window XP PCs and use Pentium IV whose CPU speed is at least 1.7 GHz and run on Windows XP. While all the client nodes above the 2,000th ms line run on UNIX Solaris computer whose processor is Ultra SPARC II with speed less than 700 MHz.

One set of results; Atrium UNIX [1] is shown only in this section, comparing two sets of tests on the same computer, SUN UltraSPARC-IIe sparcv9 processor operated at 650 MHz with 512 Mbytes of memory, but different time of execution. Atrium UNIX [1] in pink was run on weekday in the evening where Atrium UNIX [2] in yellow was run on weekend in the evening. The comparison is almost obvious that the execution of the weekend tests run faster than of the weekday tests. Their averages are shown in the graph of Figure 6- 16.

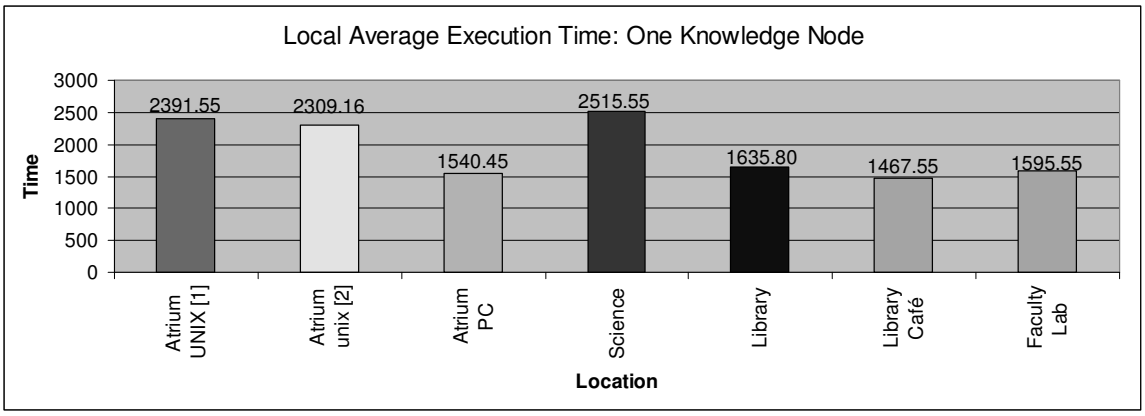


Figure 6- 16 Average Execution Times among Client Nodes at Local Simulation on One Medical Knowledge Node

Due to the slowest processor of the client node, SUN SPARC II sparc operates at 270 MHz and memory is 128 Mbytes, at Science network, it shows the longest average execution time. All PCs at Atrium, Library and Faculty Lab are similar specification on both processor and memory. However, the Atrium PC shows the lowest average execution time for two reasons; the client in the Atrium PC is located in the same LAN of the medical knowledge node, and it was run on weekend.

The Table 6- 1 on the next page shows the summary of execution of each node including average execution times, hardware specifications, hours of execution, and the distances from the client to the knowledge nodes[1].

[1] Tables of the same format but in difference network simulation will be shown again in section 6.3, 6.4, and 6.5 with more variety of comparison.

Location	Network	Atrium UNIX [1]	Atrium UNIX [2]	Atrium PC	Science	Library	Library Café	Faculty Lab
	Localize with Brooklyn Knowledge Server	Local LAN	Local LAN	Local LAN	Across LAN	Across LAN	Across LAN	Across LAN
Execution Time	Average	2391.55	2309.16	1540.45	2515.55	1635.80	1467.55	1595.55
	Max	2978.00	2424.00	1578.00	2676.00	1718.00	1496.00	1641.00
	Min	2234.00	2220.00	1515.00	2363.00	1562.00	1434.00	1516.00
	Variance	172.46	57.90	17.23	102.03	38.03	17.61	36.15
Distance from Knowledge Node to Client (miles)	Brooklyn Node	0	0	0	0.15	0.2	0.25	0.2
Client Connection	Speed (kbps)	45,000	45000	45000	45,000	45,000	45000	45,000
	Media Type	T3	T3	T3	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	512	512	256	128	384	384	256
	CPU	SUN Ultra SPARC IIe 650 MHz	SUN Ultra SPARC IIe 650 MHz	Pentium IV 1.7 GHz	SUN Ultra SPARC II 270 MHz	Pentium IV 1.8 GHz	Pentium IV 3.2 GHz	Pentium IV 1.9 GHz
Execution Hour	Client	5 PM - 7 PM	4 PM - 5 PM	2 PM - 3 PM	5 PM - 6 PM	8 PM - 9 PM	4 PM - 5 PM	4 PM - 6 PM
	Brooklyn Node	5 PM - 7 PM	4 PM - 5 PM	2 PM - 3 PM	5 PM - 6 PM	8 PM - 9 PM	4 PM - 5 PM	4 PM - 6 PM
	Date	Wed/12/14/05	Sun/02/26/06	Sun/02/26/06	Sun/02/26/06	Wed/02/22/06	Sun/02/26/06	Thu/02/23/06

Table 6- 1 Summary of Local Simulation on Single Medical Knowledge Node

The Figure 6- 17 shows two graphs of execution times on two-near-proximity-knowledge-node and two-far-proximity-knowledge-node for which we compare among local nodes.

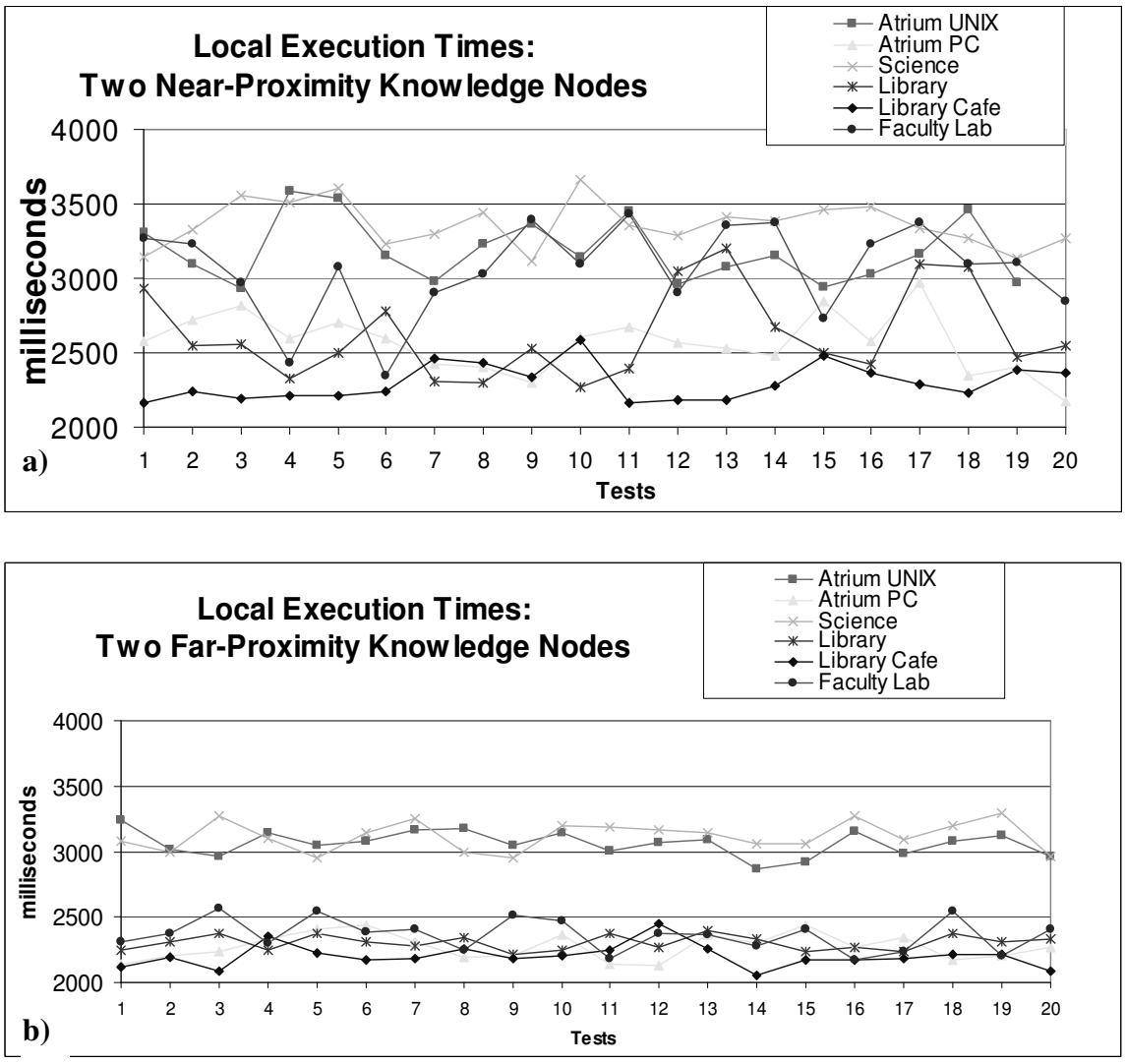


Figure 6- 17 Comparison of Tests among Nodes at Local Simulation on Two Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

Considering graphs a) and b) of the Figure 6- 17, the graph line of tests run on two far-proximity knowledge nodes are more stable than those run on two near-proximity nodes especially for the tests at Faculty Lab, Library, and Atrium PC whose variances

differ widely, 304.26, 295.71, and 193.93 ms sequentially, whereas the variances of any test on two far-proximity nodes are less than 125 ms..

Two following graphs in Figure 6- 18 show the comparison of average execution times among client nodes running on two knowledge nodes.

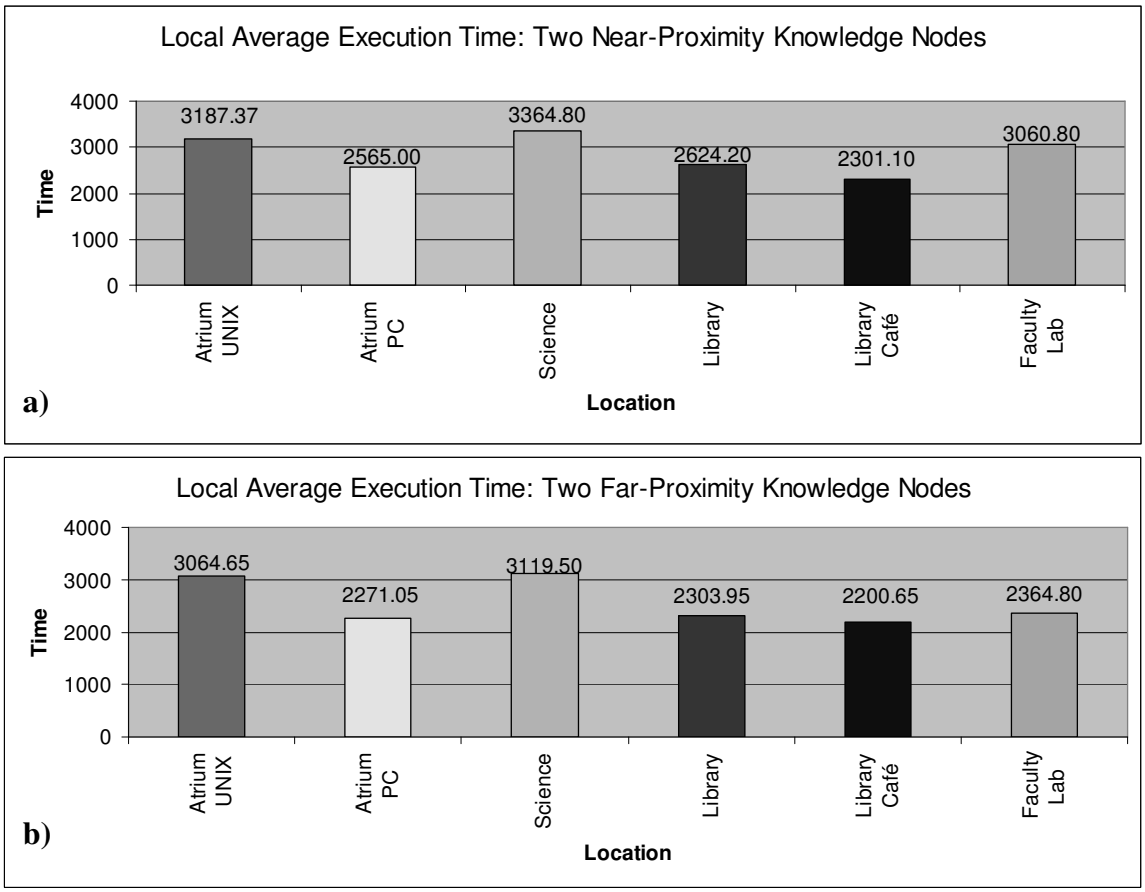


Figure 6- 18 Average Execution Times among Client Nodes at Local Simulation on Two Medical Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

Average execution times of client at Library Café in both graph a) and b) in Figure 6- 18 show the lowest comparing to other client nodes including the client node at the Atrium PC where the main knowledge node located. Conversely, in Figure 6- 16, the average time of client node at Atrium PC is the lowest because it ran only on one knowledge node located in Atrium Lab. In reality, most patients are not in the same

location of the knowledge node. Therefore, when a patient accesses multiple medical knowledge nodes, the distance between the locations of patient and knowledge may be affected. We shall see more examples later in this chapter.

The following two tables, Table 6- 2 and Table 6- 3 show the summary of the simulation on local network where client nodes are located and run on two medical knowledge nodes.

Location	Network	Atrium UNIX	Atrium PC	Science	Library	Library Café	Faculty Lab
	Localize with Brooklyn Knowledge Server	Local LAN	Local LAN	Across LAN	Across LAN	Across LAN	Across LAN
Execution Time	Average	3187.37	2565.00	3364.80	2624.20	2301.10	3060.80
	Max	3589.00	2969.00	3668.00	3203.00	2587.00	3437.00
	Min	2936.00	2172.00	3113.00	2266.00	2166.00	2343.00
	Variance	210.48	193.93	154.63	295.71	120.72	304.26
Distance from Knowledge Node to Client (miles)	Brooklyn Node	0	0	0.15	0.2	0.25	0.2
	Islandia Long Island Node	42	42	42	42	42	42
Client Connection	Speed (kbps)	45,000	45,000	45,000	45,000	45,000	45,000
	Media Type	T3	T3	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	512	256	128	384	384	256
	CPU	SUN Ultra SPARC IIe 650 MHz	Pentium IV 1.7 GHz	SUN Ultra SPARC II 270 MHz	Pentium IV 1.8 GHz	Pentium IV 3.2 GHz	Pentium IV 1.9 GHz
Execution Hour	Client	4 PM - 5 PM	2 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	Brooklyn Node	4 PM - 5 PM	2 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	Long Island Node	4 PM - 5 PM	2 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	Date	Sun/02/26/06	Sun/02/26/06	Sun/02/26/06	Wed/02/22/06	Sun/02/26/06	Thu/02/23/06

Table 6- 2 Summary of Local Simulation on Two Near-Proximity Medical Knowledge Nodes

Location	Network	Atrium UNIX	Atrium PC	Science	Library	Library Café	Faculty Lab
	Localize with Brooklyn Knowledge Server	Local LAN	Local LAN	Across LAN	Across LAN	Across LAN	Across LAN
Execution Time	Average	3064.65	2271.05	3119.50	2303.95	2200.65	2364.80
	Max	3238.00	2437.00	3294.00	2391.00	2447.00	2563.00
	Min	2870.00	2125.00	2947.00	2219.00	2057.00	2172.00
	Variance	94.60	99.23	113.12	55.99	89.12	122.46
Distance from Knowledge Node to Client (miles)	Brooklyn Node	0	0	0.15	0.2	0.25	0.2
	San Francisco Node	2572	2572	2572	2572	2572	2572
Client Connection	Speed (kbps)	45,000	45,000	45,000	45,000	45,000	45,000
	Media Type	T3	T3	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	512	256	128	384	384	256
	CPU	SUN Ultra SPARC IIe 650 MHz	Pentium IV 1.7 GHz	SUN Ultra SPARC IIi 270 MHz	Pentium IV 1.8 GHz	Pentium IV 3.2 GHz	Pentium IV 1.9 GHz
Execution Hour	Client	3 PM - 4 PM	3 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	Brooklyn Node	3 PM - 4 PM	3 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	San Francisco Node	12 PM - 1 PM	12 PM - 1 PM	3 PM - 4 PM	5 PM - 6 PM	2 PM - 3 PM	1 PM - 3 PM
	Date	Sun/02/26/06	Sun/02/26/06	Sun/02/26/06	Wed/02/22/06	Sun/02/26/06	Thu/02/23/06

Table 6- 3 Summary of Local Simulation on Two Far-Proximity Medical Knowledge Node

The next graph shows the comparison among client nodes at local network when running on three knowledge nodes located at Brooklyn, Long Island, and San Francisco.

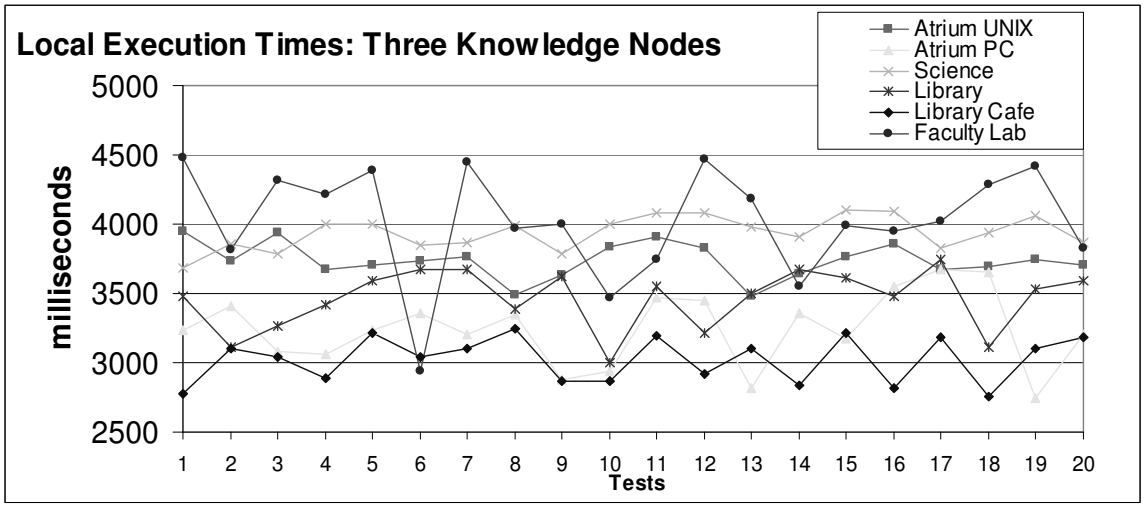


Figure 6- 19 Comparison of Tests among Nodes at Local Simulation on Three Knowledge Nodes

Almost every node running PCs show a wide variance of execution times on three knowledge nodes except the node at Library Café whose processor is much faster than processor speed of other PC's. The variances of execution times of client nodes on three knowledge nodes running at Faculty Lab, Atrium PC, and Library are 396.40, 264.60, and 215.08 milliseconds sequentially, whereas the variance of execution times of client node at Library Café is 163.72 milliseconds. Surprisingly, the variances of execution times of client node on three knowledge nodes when running on SUN computer are less than 130 milliseconds. The ranges of execution times of client nodes on three knowledge nodes running on SUN computer at Atrium Lab, and Science Network are 127.40 and 119.43 milliseconds sequentially. This seems to have similar affect on the next graph when we are considering the averages.

The next graph shows the averages of execution times among client nodes running at local network on three distributed medical knowledge nodes.

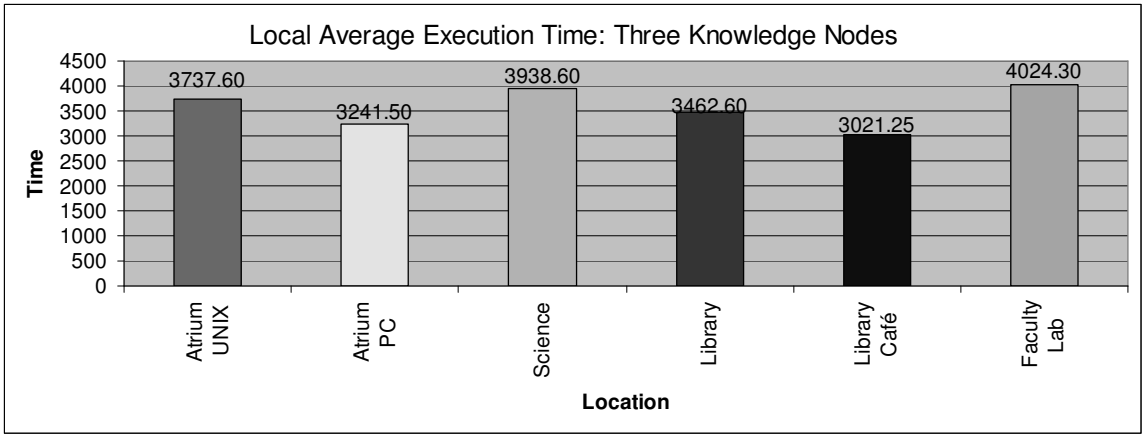


Figure 6- 20 Average Execution Times among Client Nodes at Local Simulation on Three Medical Knowledge Nodes

The local average execution times among client nodes are very close when running on three knowledge nodes. Considering the local average execution time when running on one knowledge node of Figure 6- 16, the average times of clients with low performance computer (Atrium UNIX and Science) are almost twice compared to clients with high performance computer (Atrium PC, Library, Library Café and Faculty Lab). In case of two knowledge nodes in Figure 6- 18, the average times do not differ as much as the two groups of clients using low and high performance computers. The results of Figure 6- 20 confirm this. Most of average execution times of client with high performance computer are very close to the average execution times of client with low performance computer. Especially the client at Faculty Lab, its average execution time run during weekday with the high volume of network traffic, is higher than the average execution time of the client with low performance computer.

Therefore, the heavy traffic of network has a greater effect than the performance of the computer on the execution time on the tests on local clients when the execution involves more medical knowledge nodes.

Location	Network	Atrium UNIX	Atrium PC	Science	Library	Library Café	Faculty Lab
	Localize with Brooklyn Knowledge Server	Local LAN	Local LAN	Across LAN	Across LAN	Across LAN	Across LAN
Execution Time	Average	3737.60	3241.50	3938.60	3462.60	3021.25	4024.30
	Max	3949.00	3671.00	4105.00	3750.00	3242.00	4484.00
	Min	3484.00	2750.00	3688.00	3000.00	2759.00	2937.00
	Variance	127.40	264.60	119.43	215.08	163.72	396.40
Distance from Knowledge Node to Client (miles)	Brooklyn Node	0	0	0.15	0.2	0.25	0.2
	Long Island Node	42	42	42	42	42	42
	San Francisco Node	2572	2572	2572	2572	2572	2572
Client Connection	Speed (kbps)	45,000	45,000	45,000	45,000	45,000	45,000
	Media Type	T3	T3	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	512	256	128	384	384	256
	CPU	SUN Ultra SPARC IIe 650 MHz	Pentium IV 1.7 GHz	SUN Ultra SPARC II 270 MHz	Pentium IV 1.8 GHz	Pentium IV 3.2 GHz	Pentium IV 1.9 GHz
Execution Hour	Client	4 PM - 5 PM	3 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	Brooklyn Node	4 PM - 5 PM	3 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	Long Island Node	4 PM - 5 PM	3 PM - 4 PM	6 PM - 7 PM	8 PM - 9 PM	5 PM - 6 PM	4 PM - 6 PM
	San Francisco Node	1 PM - 2 PM	12 PM - 1 PM	3 PM - 4 PM	5 PM - 6 PM	2 PM - 3 PM	1 PM - 3 PM
	Date	Sun/02/26/06	Sun/02/26/06	Sun/02/26/06	Wed/02/22/06	Sun/02/26/06	Thu/02/23/06

Table 6- 4 Summary of Local Simulation on Three Medical Knowledge Node

6.3.Real-Time Regional Simulation

The tests at regional network are in New York City area where the main knowledge node is located in one of its borough, Brooklyn. We ran the tests on various boroughs of New York City. Most of them are using the CUNYNET network of educational City University of New York. However, there are some tests located a residential area. The details are explained in the next section, Topology of Regional Simulation. In section 6.3.2, the results are compared between single and multiple knowledge nodes of each node in regional network, whereas the comparisons among nodes in regional network are shown in section 6.3.3.

6.3.1. Topology on Regional Simulate

The topology of regional simulation is among different locations inside New York metropolitan area. This regional study includes five different locations. Some locations ran the tests on different settings:

- In Woodside Queens, a medium-density resident area, we ran the tests in three different settings; PC set 1 with Digital Subscriber Line (DSL), PC set 2 with 56K modem, and PC set2 with (DSL). PC set 2, which is a desktop, runs on Pentium IV 1.8 GHz and 1 Gbytes of memory whereas the PC set 1, which is a laptop, runs on Celeron 2.8 GHz and 768 Mbytes of memory. So PC set 1 uses wireless connection to internet via DSL, whereas PC set 2 uses hardwire to connect to internet via DSL and 56K modem.

- In Flushing Queens, a high-density resident area, we ran the tests at the New York Public Library, a government-funded facility. When we ran the tests, the library was crowded serving about 100 computer stations, All computers were occupied by a

user who mostly spent time searching the internet. There were lots of Queens residents waiting in line to access the computers. The library limits each user to 30 minutes of usage. Beside computer usage, the library, at that time, was also serving more than 500 people for book, music tape and CD, DVD, and other library services.

The next three locations are at colleges of City University of New York (CUNY). Because all CUNY colleges are under the same administration of the university, the internet access of all CUNY colleges are using CUNYNET access.

- In Midtown Manhattan, a high commercial area, the tests were run at the Graduate Center, CUNY inside the main computer lab at the lower level of the building. This computer lab supports several computing platforms providing access information resources to a variety of available software, including systems dedicated to large-scale empirical research. The Graduate Center network design is a CISCO based Ethernet network with a distributed star topology. This network provides over 1500 campus computers with high-speed local communications and a gateway to the Internet via CUNYNET. We ran the tests on two different setting; on weekday and weekend when there were few accesses to computer network at the Graduate Center.

- In Downtown Manhattan, also a high commercial area, the tests were run at the Borough Manhattan Community College, BMCC, CUNY inside the open-access lab located on the first floor of the main building. It provides a comprehensive variety of services supporting the use of information technology in the educational process. We ran the tests at BMCC on weekend when there is a small traffic on the computer network. We can compare these results with the results of weekend setting at The Graduate Center, CUNY.

- In Staten Island, where is a medium resident area, the tests were run at the College of Staten Island, CSI, CUNY which is one of the CUNY largest open-area colleges with more than 200 acres. Buildings are far apart to another. We ran the tests at a computer station inside the college library which enables librarians and students to retrieve non-print sources of information from beyond the walls of the building through online, satellite, and fiber-optic technology. The tests were run during weekday. So the results can be compared with the results of one setting at The Graduate Center, CUNY which is also during a weekday.

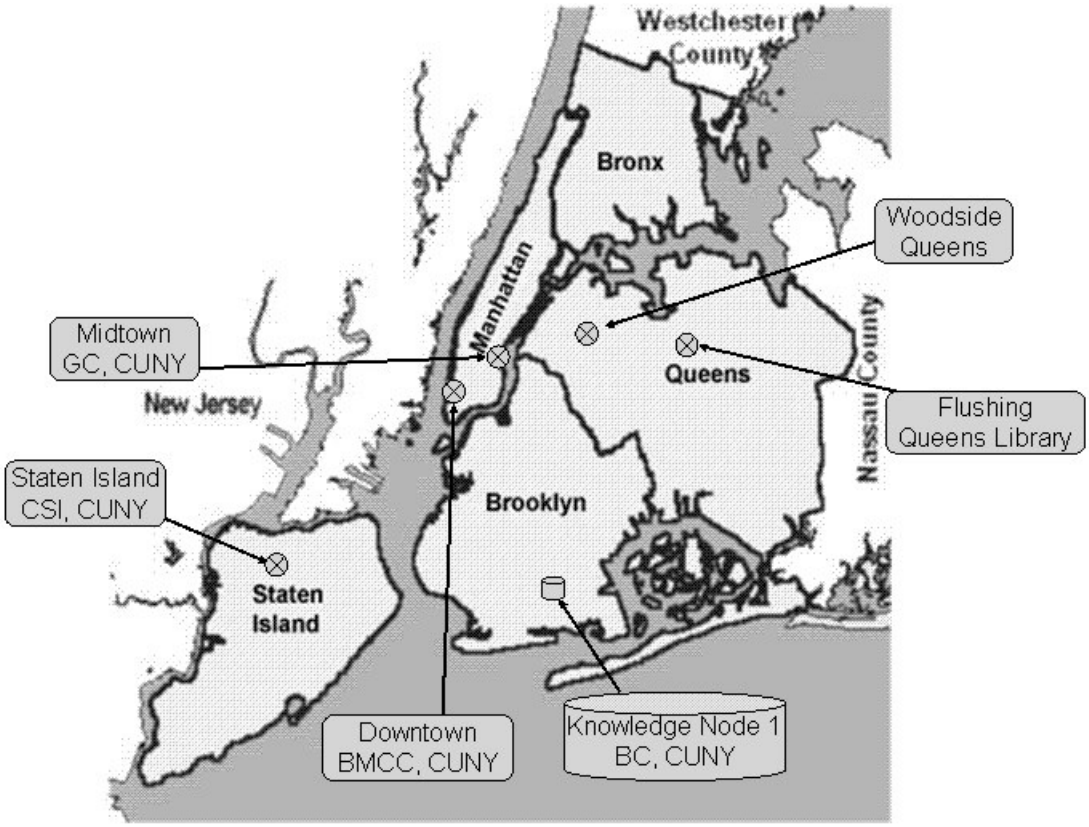


Figure 6- 21 Regional Map. Client Nodes in New York City are located at Woodside in Queens, Flushing Queens Library in Queens, The Graduate Center CUNY in midtown Manhattan, Borough of Manhattan Community College CUNY in downtown Manhattan, and College of Staten Island CUNY at Staten Island. The main knowledge node is located at Brooklyn College in Brooklyn.

All computers run on regional simulation are Windows-based. All of them run on Windows XP operating system. Most of these clients use Pentium IV processors whose speeds are between 1.6 GHz to 2.8 GHz and memory sizes between 256 to 1024 Mbytes except the client at Staten Island whose processor is Pentium IV 3.0 GHz which is quite fast but only 128 Mbytes of memory which is quite small because the main purpose of this client is only to research on the internet.

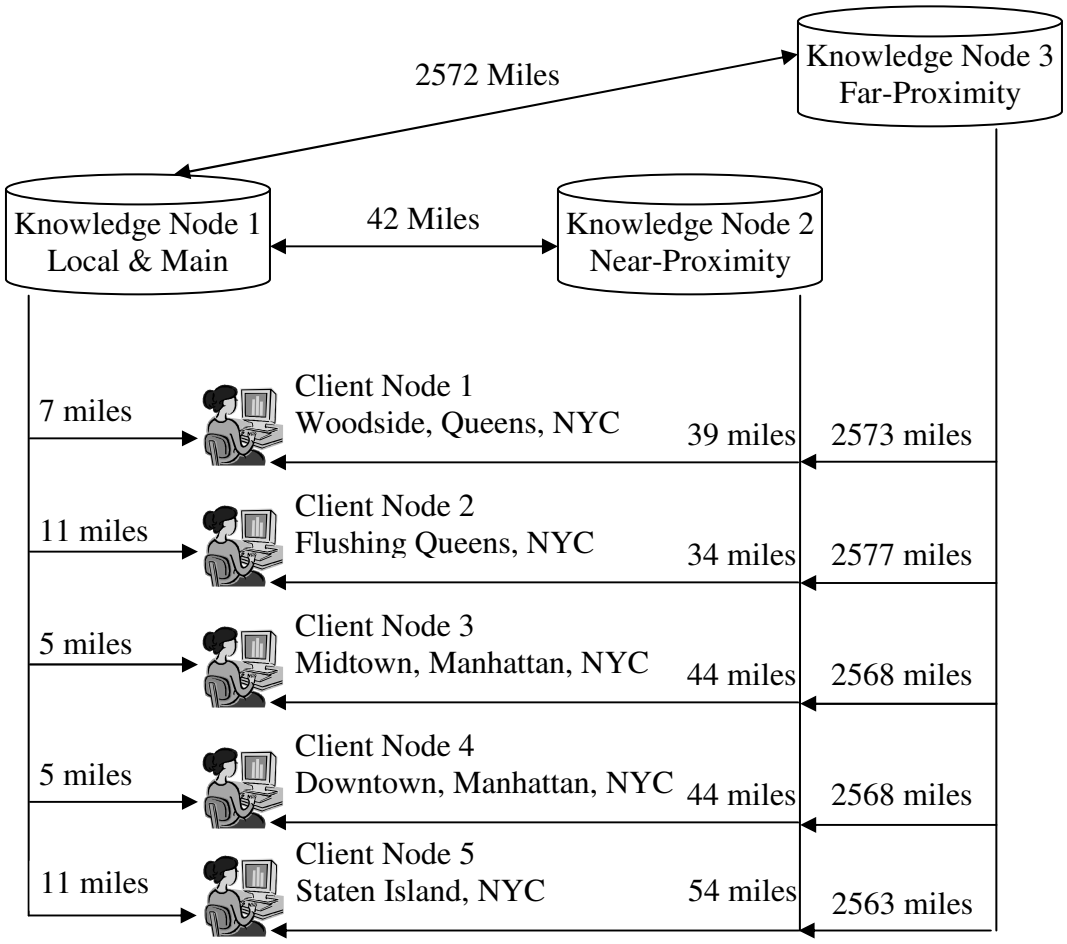


Figure 6- 22 Topology of Regional Simulation in New York City

Also, the distances from each regional client to the knowledge nodes are not very different; less than 12 miles from a regional client to the main knowledge node at Brooklyn, between 30 and 55 miles from a regional client to the near-proximity medical

knowledge node at Long Island, and between 2560 and 2580 miles from a regional client to the far-proximity medical knowledge node at San Francisco. However when we consider the national and global network, the distance from a client to medical knowledge node will be very different.

6.3.2. Results and Comparison between Single and Multiple Knowledge Nodes on Regional Simulate

The first test of regional simulation is at Woodside. The Figure 6- 23 shows the result of running at Woodside in Queens, New York City, by using PC set 1: Celeron 2.8 GHz with 768 Mbytes of memory and wireless connection. These tests were run at different times.

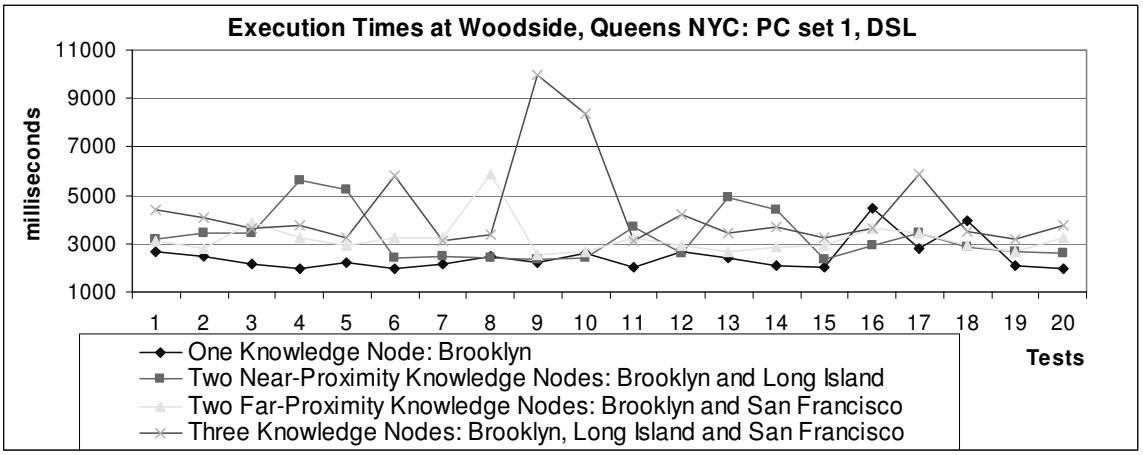


Figure 6- 23 Execution Times at Woodside on PC set 1 of Regional Simulation

For one knowledge node, the tests were run at night. The graph is not smooth as other graphs of the tests on one knowledge node. Its variance is as big as 649.32 milliseconds. However, the tests on two far-proximity knowledge nodes were also run at night and its variance is 709.96 ms. The tests on two near-proximity knowledge nodes and three knowledge nodes were run during the day and their variances are 1017.77 and

1837.84 ms sequentially. However these big variances are comparable with results of PC set 2 in Figure 6- 24.

The next graphs in Figure 6- 24 show two sets of results running from the same computer and location but different internet connections; DSL and 56k-Modem. It was run on PC; Pentium IV 1.8 GZ, and 1024 Mbytes of RAM, at Woodside.

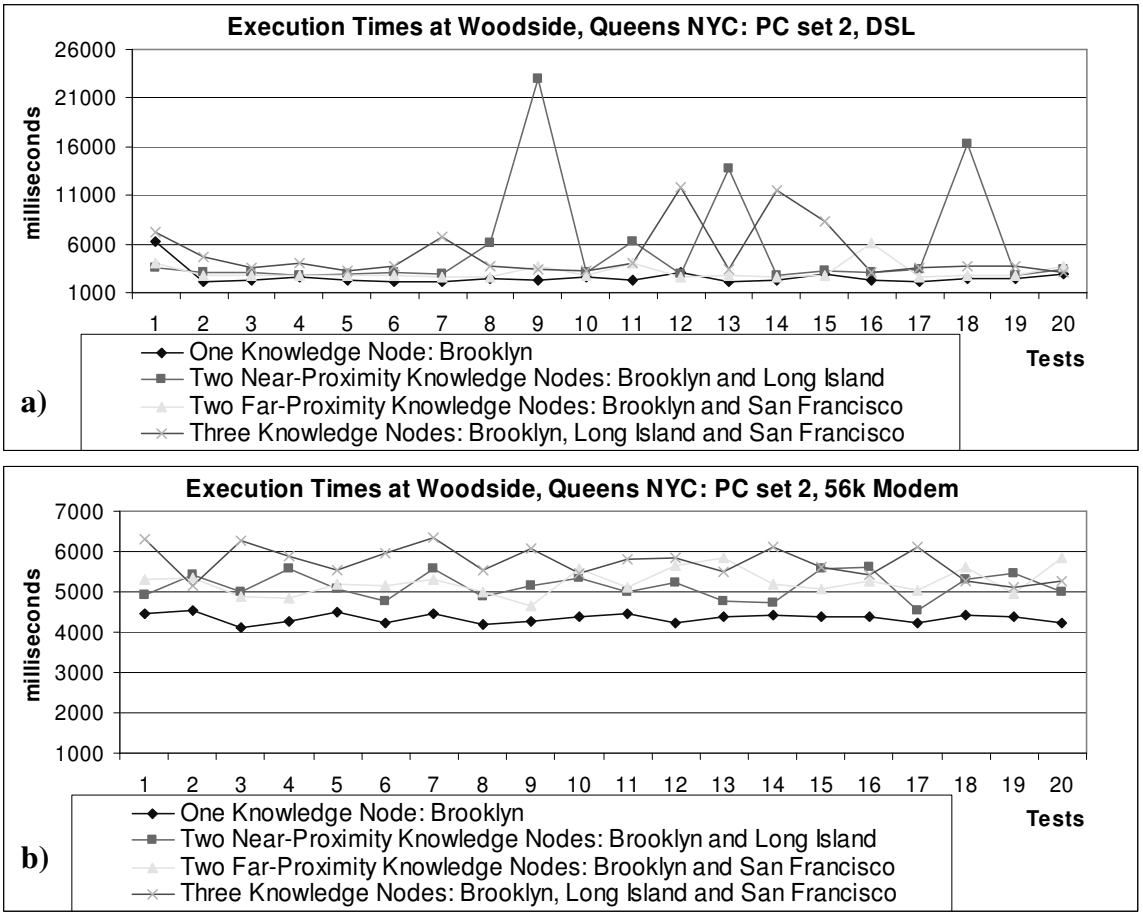


Figure 6- 24 Execution Times at Woodside on PC set 2 of Regional Simulation a) Connect to Internet via DSL b) Connect to Internet via 56k Modem

Both sets of tests in Figure 6- 24 were run at night. Because the connection is 56k modem, all execution times (on any number of nodes) of part b) are high. However, all of them show variances less than 400 ms. In comparison, the variances of tests on the same set of computer, but, with higher speed of internet connection (DSL) in the part a) are much higher. The variances of the tests on DSL of on one knowledge node, two near-

proximity and two far-proximity knowledge nodes and three knowledge nodes are 911.46, 5520.64, 854.13, and 2708.87 ms.

Considering their averages on the case of two near-proximity and three knowledge nodes, both types of internet connection shows a close value of averages between 5000 to 6000 ms. The averages of two near-proximity knowledge nodes are 5147 ms. for 56k-modem connection and 5556 ms. for DSL. The averages of three knowledge nodes are 5731 ms. for 56k-modem connection and 4968 ms for DSL. However in the case of one knowledge node, the averages are much different between two different internet connections; 4349 ms. for 56k-modem and 2607 ms. for DSL.

This shows us that the time spending from the client to Internet Service Provider (ISP) (56k or DSL) highly causes the high execution time when there is only one medical knowledge node. In contrast, when more medical knowledge nodes are added to the medical diagnostic system, the time spending from the client to the ISP does not highly affected the execution time but the time of processing among different knowledge nodes becomes the major cause of the execution time.

The Figure 6- 25 shows the results of the tests at Flushing Queens public library using PC Pentium IV 1.6 GHz with 256 Mbytes of memory. This computer is not considered slow. However, at the time of testing, the library was crowded with people. Therefore, the result shows more variability.

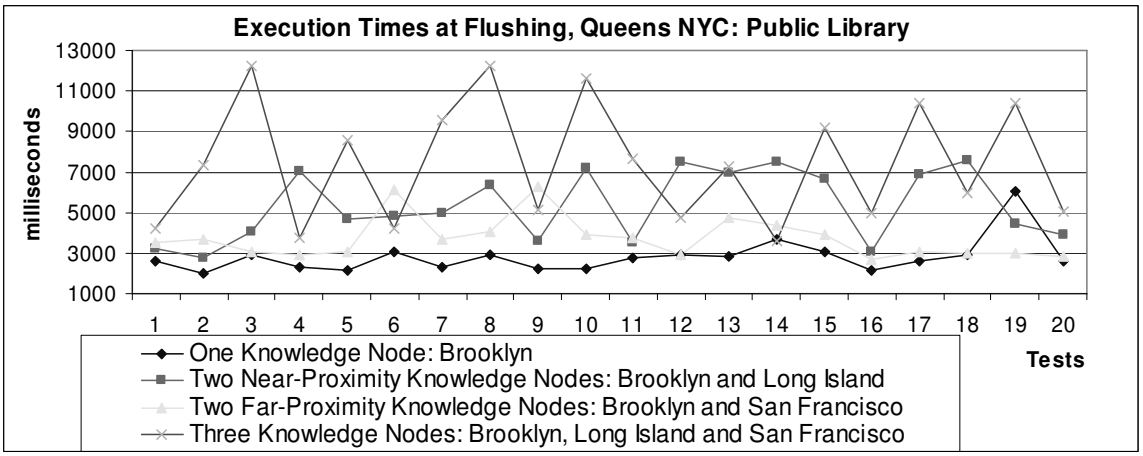


Figure 6- 25 Execution Times at Flushing Public Library of Regional Simulation

The tests were run during extremely high volume of internet access. Not including low speed 56k-modem, the results at Flushing library are very high compared to of other clients for each number of nodes. The variances of each number of nodes are large, especially for three knowledge nodes and even though the results of only one knowledge node. This shows similar result to the next graph in Figure 6- 26 a) when running in midtown Manhattan during the weekday.

The result of running in midtown Manhattan at The Graduate Center, CUNY are showed in Figure 6- 26 into two sets; the tests during weekday and the tests during weekend. On both sets, the test was run on the same computer; Pentium IV 1.9 GHz and 256 Mbytes of memory.

Figure 6- 26 a) shows the result of testing during the weekday. The graph is similar to the results when running at Flushing library when there was an extremely high volume of internet access. Beside one knowledge node, the averages in any number of knowledge nodes are very high compared to the Figure 6- 26 b) which is the results of running during the weekend when fewer users were connecting to internet.

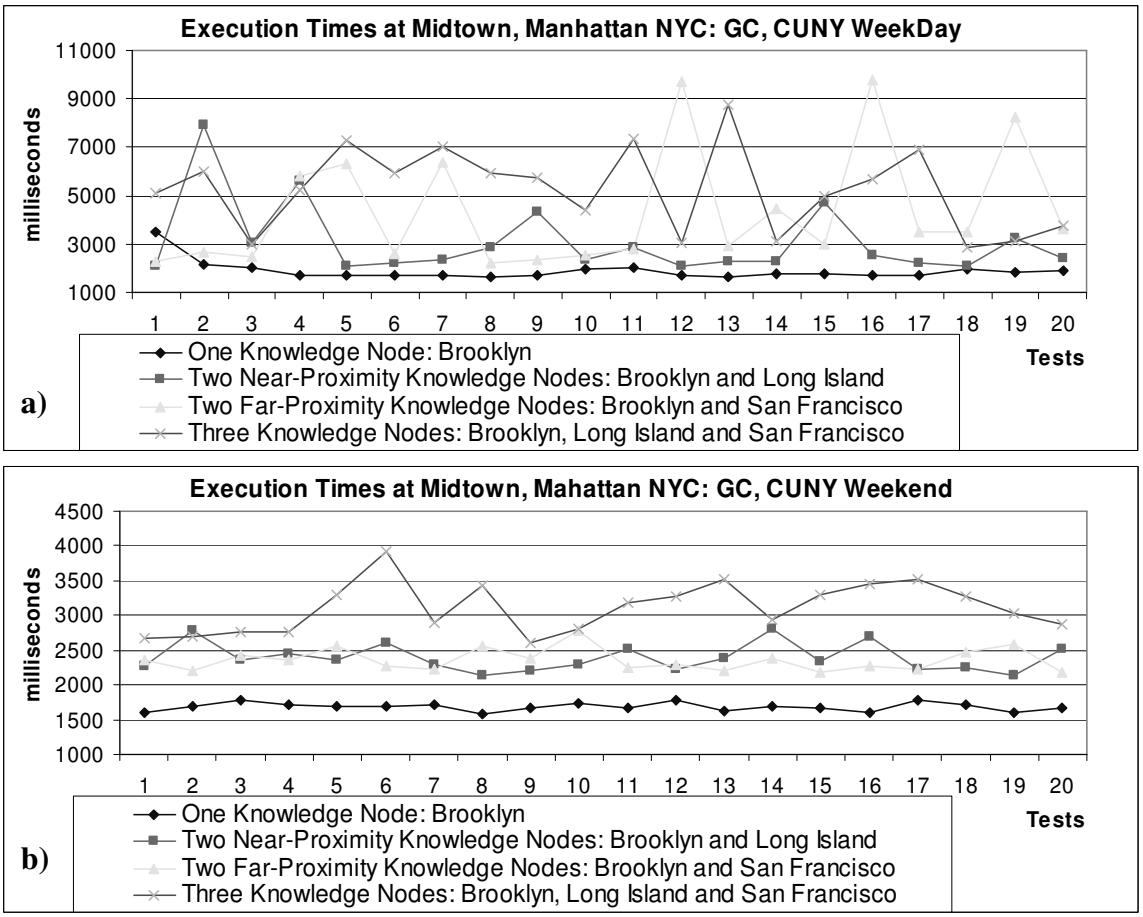


Figure 6- 26 Execution Times at GC CUNY, Midtown Manhattan, of Regional Simulation a) during Weekday b) during Weekend

Figure 6- 26 b) also shows very close averages between the results of two near-proximity and far-proximity knowledge nodes; 2391 ms. and 2358 ms. sequentially. The figure also shows a wider variance when additional knowledge nodes are added. These averages and variances also show similar results to the next graph in Figure 6- 27 where the test was run on a different location but the same factor.

Figure 6- 27 shows the results of tests running at the Borough of Manhattan Community College, BMCC CUNY, downtown Manhattan. The tests were run on weekend when there are few students accessing the internet. We ran on a computer using Pentium IV 2.8 GHz and 512 Mbytes of memory which is better than most computers of any clients in regional simulation.

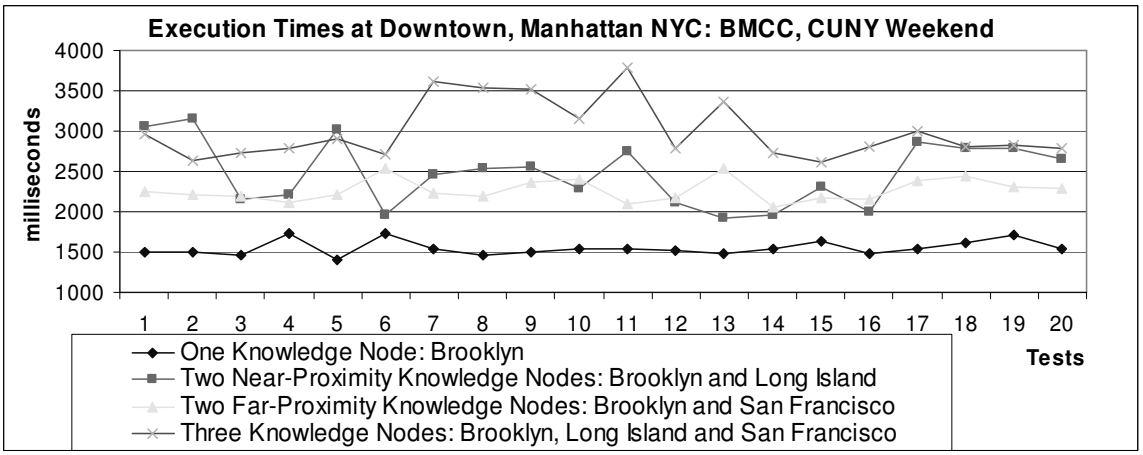


Figure 6- 27 Execution Times at BMCC CUNY, Downtown Manhattan, of Regional Simulation

As mention earlier, the results of running at downtown Manhattan on weekend is almost the same as of midtown Manhattan on the same period of time. The close averages of running at downtown Manhattan on both two near-proximity and far-proximity knowledge nodes are 2479 ms. and 2264 ms. sequentially whereas their variances are affected by the number of nodes similar to the result midtown Manhattan during the weekend in Figure 6- 26 b)

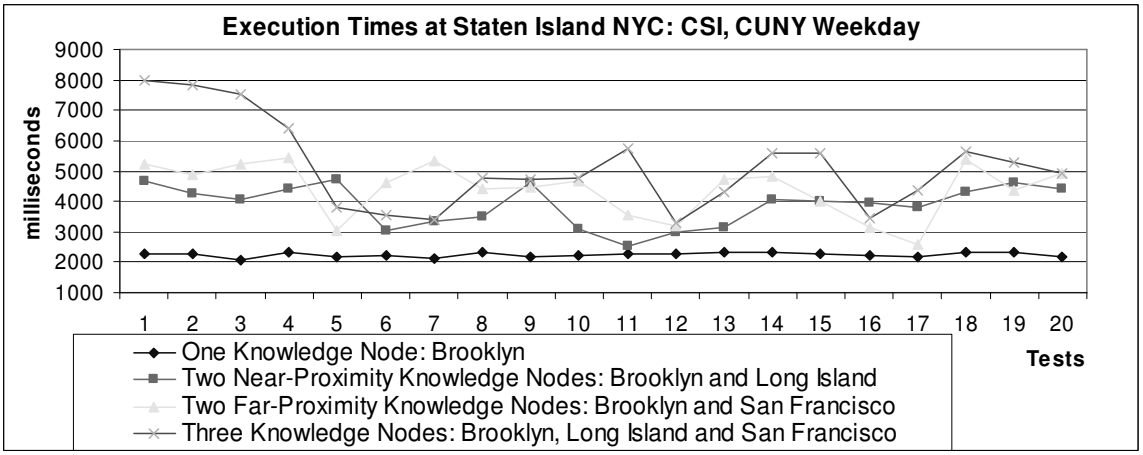


Figure 6- 28 Execution Times at CSI CUNY, Staten Island, of Regional Simulation

The last set of results of regional simulation was run at College of Staten Island, CSI CUNY by using a computer at its library. The test was also run on a computer with

high performance processor; Pentium IV 3.0 GHz but only 128 Mbytes of memory which is for internet purpose only.

The results are also similar to results running at Flushing and midtown Manhattan during the weekday because the test at Staten Island was also run during the weekday. The averages are high in every number of knowledge nodes and the variances are wide in almost every number of knowledge nodes except for one knowledge node.

The following graph in Figure 6- 29 shows the comparison of average execution time of all clients in regional simulation together categorized by the number of knowledge nodes.

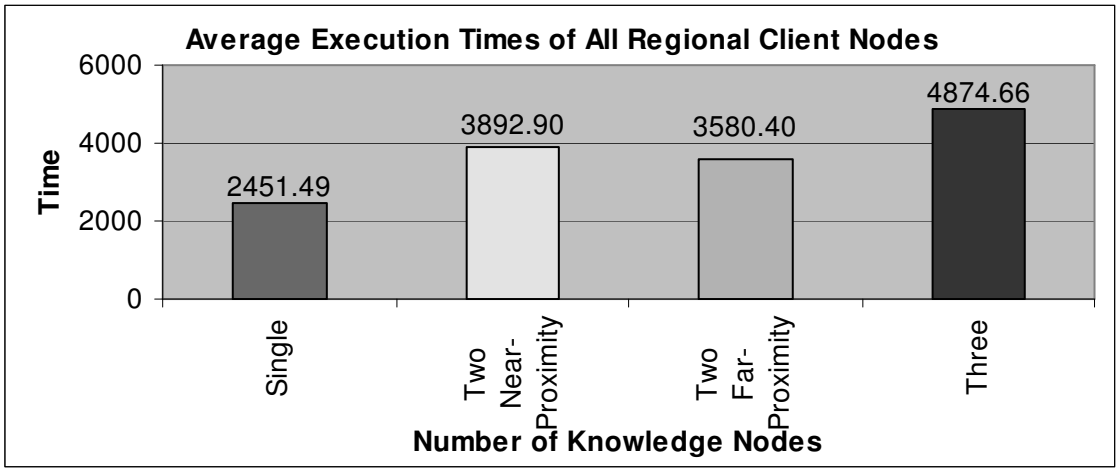


Figure 6- 29 Average Execution Times of all Clients for each Number of Knowledge Nodes at Regional Simulation

The mean between the average execution times of two near-proximity and of two far-proximity knowledge nodes from Figure 6- 29 is (3892.90 + 3580.40) / 2 = 3736.65 ms. The mean between the average of execution time of one and of three knowledge nodes is (2451.49 + 4874.66)/2 = 3663.075 ms. Therefore, the means of the two types of two knowledge nodes are similar.

Using the means, we can find the average execution time when the number of knowledge is increased:

3663.075 – 2451.49 = 1211.585

from one knowledge node –to- the mean between one and three knowledge nodes

4874.66 – 3663.075 = 1211.585

from the mean between one and three knowledge nodes –to- three knowledge nodes

3736.65 – 2451.49 = 1285.16

from one knowledge node –to- the mean of two knowledge nodes

4874.66 – 3736.65 = 1138.01

from the mean of two knowledge nodes to three knowledge nodes

The average is (1211.585 + 1211.585 + 1285.16 + 1138.01)/4 = 1211.585

That accumulates approximately 1200 milliseconds of execution time when increasing the number of knowledge nodes by one (from one node to two nodes and from two nodes to three nodes.) on the regional execution.

If 1211.585 milliseconds is the difference of execution times between n and n+1 knowledge nodes and the average execution time of the single node is 2451.49 milliseconds, then 2451.49 – 1211.585 = 1239.905, that is approximately 1200 milliseconds should be the set up time when **not** considering the execution time at a knowledge node.

6.3.3. Comparison among Client Nodes on Regional Simulate

In this section, we show the comparison among nodes in regional simulation by considering separately each number of knowledge nodes.

The next graph in Figure 6- 30 shows the comparison among different nodes in the regional simulation on the execution of one knowledge node.

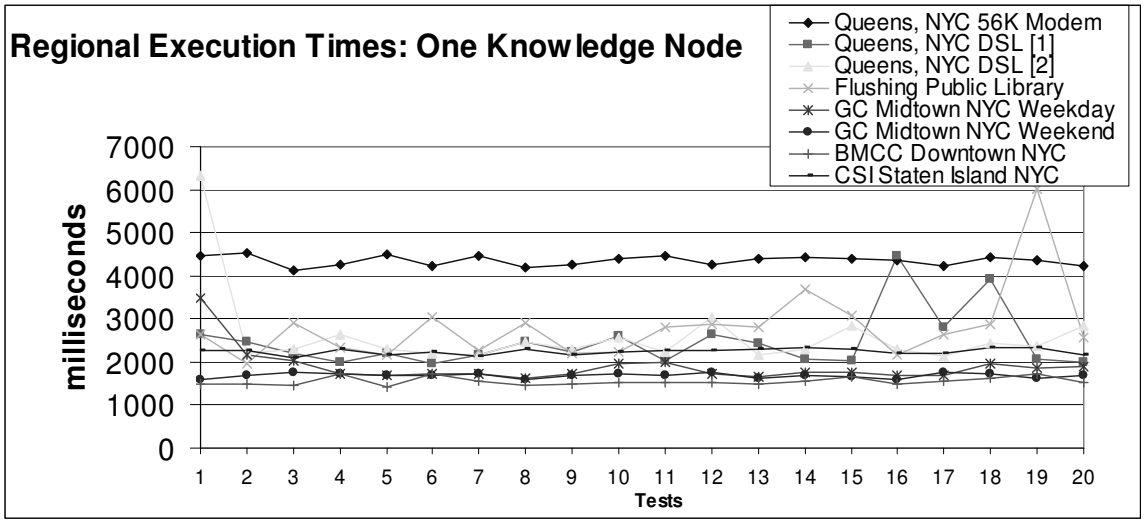


Figure 6- 30 Comparison of Tests among Nodes at Regional Simulation on One Knowledge Node

The regional graphs of BMCC downtown, and GC midtown weekend show the stability of fundamental time when it is executed low network traffic on weekend. Their averages are only 1548.35 ms. for downtown and 1683.00 ms for midtown. While the graphs of GC midtown weekday and CSI Staten Island were executed during the weekday, they both are also quite stable, not including the first execution of midtown during the weekday (Most of the first time executions were affected by the setup time of a browser.) Their average times are 1895.60 ms. for midtown and 2246.60 ms. for Staten Island which is higher than the average execution times of weekend tests because they were executed during the weekday. The comparison of average execution times of each regional client is showed in Figure 6- 31.

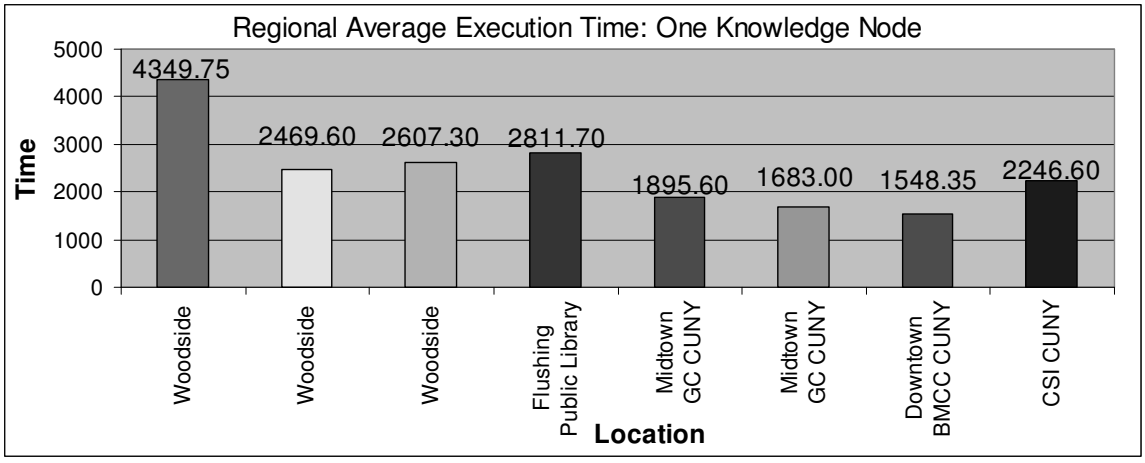


Figure 6- 31 Average Execution Times among Client Nodes at Regional Simulation on One Medical Knowledge Node

The graph in Figure 6- 30, of the execution at Flushing fluctuates more than other nodes[1] because of the highest internet usage and low internet speed connection. Three graphs of execution at Woodside Queens are stable but only the 56k-modem shows very high average, 4349.75 ms whereas the averages of any nodes in regional tests are less than 3000 ms.

Table 6- 5 shows the summary of the execution of all client nodes in the regional area of New York City.

[1] similar results later on for more knowledge nodes

Location	City	Woodside	Woodside	Woodside	Flushing Public Library	Midtown GC CUNY	Midtown GC CUNY	Downtown BMCC CUNY	CSI CUNY
	Borough	Queens	Queens	Queens	Queens	Manhattan	Manhattan	Mahattan	Staten Island
	Type	Resident	Resident	Resident	Government	Education	Education	Education	Education
Execution Time	Average	4349.75	2469.60	2607.30	2811.70	1895.60	1683.00	1548.35	2246.60
	Max	4547.00	4467.00	6329.00	6029.00	3483.00	1771.00	1735.00	2344.00
	Min	4116.00	1972.00	2143.00	1968.00	1640.00	1583.00	1406.00	2094.00
	Variance	117.83	649.32	911.46	865.53	399.73	58.12	93.47	75.77
Distance from Client To Knowledge Node (miles)		7	7	7	11	5	5	5	11
Client Connection	Speed (kbps)	56	3,000	3,000	10,000	45,000	45,000	45,000	45,000
	Media Type	56K Modem	DSL	DSL	Ethernet	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	1024	768	1024	256	256	256	512	128
	CPU (GHz)	PentiumIV 1.8	Celeron 2.8	PentiumIV 1.8	Pentium IV 1.6	Pentium IV 1.9	Pentium IV 1.9	Pentium IV 2.8	Pentium IV 3.0
Execution Hour	Client	1AM-2AM	12AM-1AM	5PM-6PM	12PM-1PM	11AM-12PM	6PM-7PM	4PM-5PM	2PM-4PM
	Date	Wed 03/15/06	Tue 12/20/05	Tue 11/29/05	Sun 02/19/06	Tue 02/07/06	Sun 02/19/06	Sun 02/19/06	Tue 03/21/06

Table 6- 5 Summary of Regional Simulation on Single Medical Knowledge Node

The Figure 6- 32 shows two graphs of execution times on two-near-proximity-knowledge-node and two-far-proximity-knowledge-node for which we compare among regional nodes.

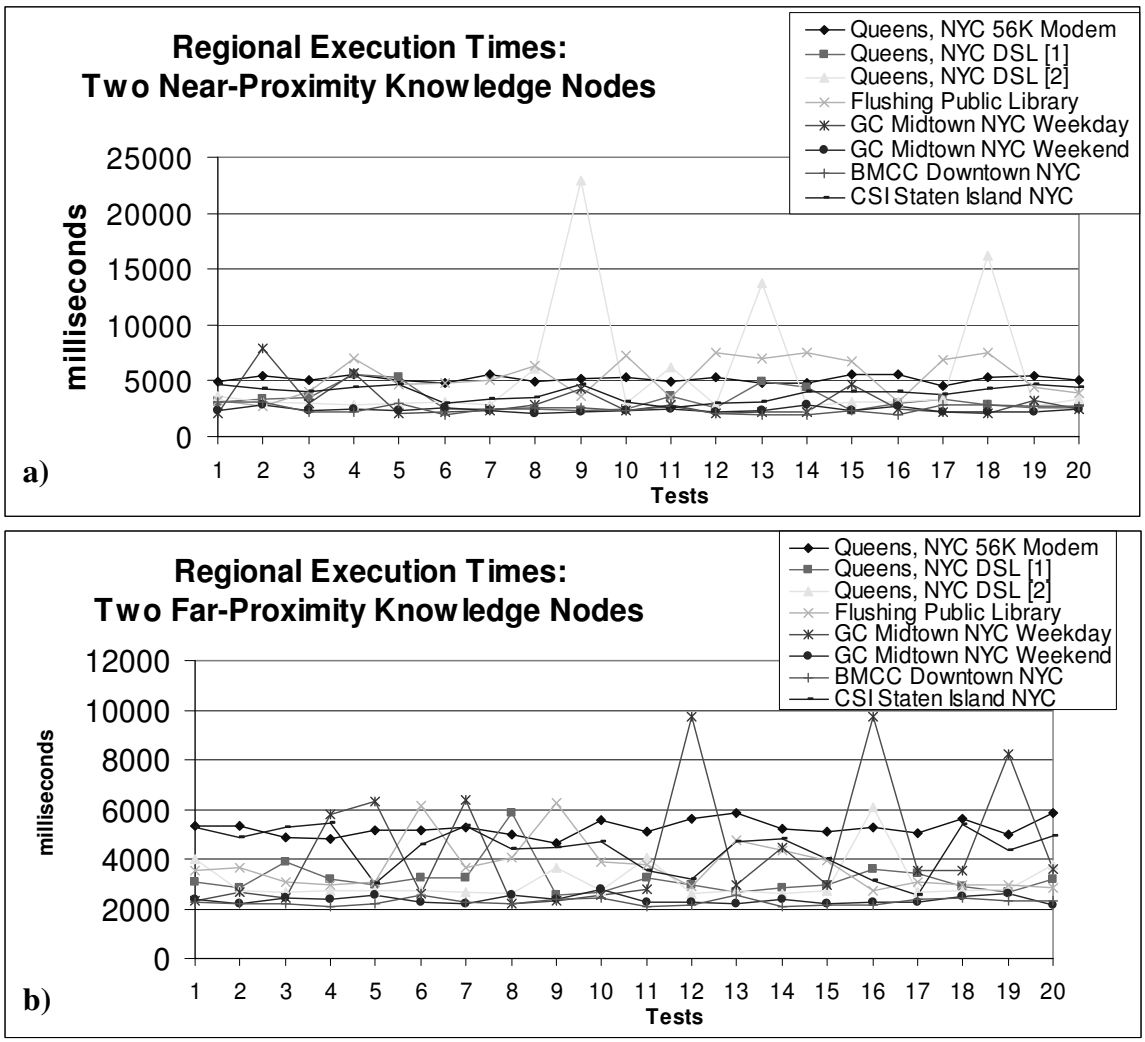


Figure 6- 32 Comparison of Tests among Nodes at Regional Simulation on Two Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

When running on two knowledge nodes on weekday, processing speed in high capacity networks is affected during the weekday, when network traffic is high, compared to the weekend when traffic is low. Graphs in Figure 6- 32 confirm this when running the tests during the weekday at the midtown, Staten Island, Flushing and

Woodside on PC set 2 with DSL. Even for sometimes, their execution times are higher than of the lower speed 56k-modem tests in Woodside.

The average times of each client for the execution of two knowledge nodes are shown in Figure 6- 33

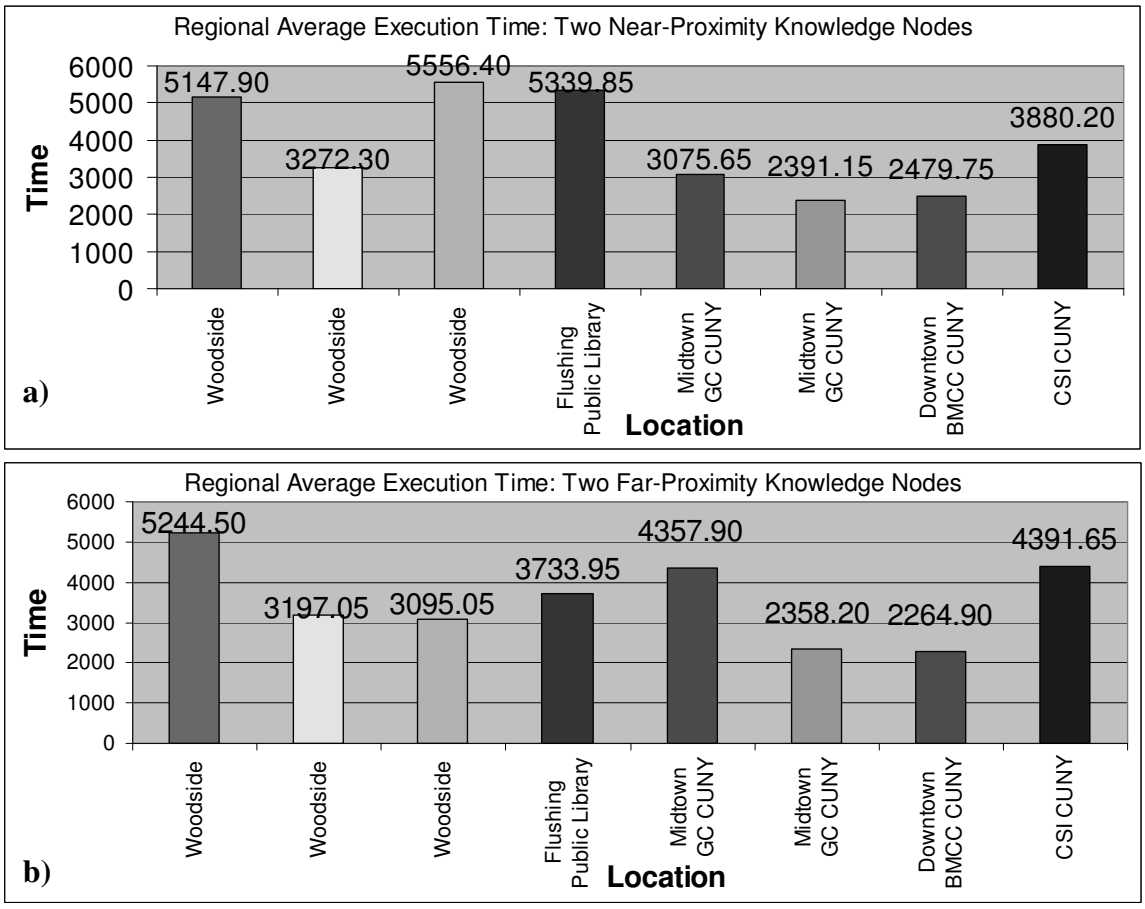


Figure 6- 33 Average Execution Times among Client Nodes at Regional Simulation on Two Medical Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

The average times of the tests on weekend such as downtown, the second bar of midtown, and the second bar of Woodside show no difference between two near-proximity and far-proximity knowledge nodes. The average times of the tests on weekday such as CSI, the first bar of midtown, and the third bar of Woodside show inconsistency of the networks during the tests. However, the tests at Flushing were also on the weekend

but they show the inconsistency due to the very high usage of internet by people at the public library. The results of the first bar at Woodside on both a) and b) of Figure 6- 33, of which are the tests on low-speed 56K-modem, also show no difference on testing between near-proximity and far-proximity.

The following two tables; Table 6- 6 and Table 6- 7, show the summary of executions of regional clients on two knowledge nodes. Table 6- 6 shows the details of execution on two near-proximity medical knowledge nodes while Table 6- 7 shows on two far-proximity medical knowledge nodes.

Location	City	Woodside	Woodside	Woodside	Flushing Public Library	Midtown GC CUNY	Midtown GC CUNY	Downtown BMCC CUNY	CSI CUNY
	Borough	Queens	Queens	Queens	Queens	Manhattan	Manhattan	Mahattan	Staten Island
	Type	Resident	Resident	Resident	Government	Education	Education	Education	Education
Execution Time	Average	5147.90	3272.30	5556.40	5339.85	3075.65	2391.15	2479.75	3880.20
	Max	5628.00	5618.00	22923.00	7563.00	7904.00	2804.00	3157.00	4703.00
	Min	4556.00	2354.00	2673.00	2736.00	2078.00	2131.00	1922.00	2531.00
	Variance	323.76	1017.77	5520.64	1727.54	1499.32	202.97	397.62	664.57
Distance from Knowledge Node to Client (miles)	Brooklyn Node	7	7	7	11	5	5	5	11
	Islandia Long Island Node	39	39	39	34	44	44	44	54
Client Connection	Speed (kbps)	56	3,000	3,000	10,000	45,000	45,000	45,000	45,000
	Media Type	56K Modem	DSL	DSL	Ethernet	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	1024	768	1024	256	256	256	512	128
	CPU (GHz)	Pentium IV 1.8	Celeron 2.8	Pentium IV 1.8	Pentium IV 1.6	Pentium IV 1.9	Pentium IV 1.9	Pentium IV 2.8	Pentium IV 3.0
Execution Hour	Client	1AM-2AM	9AM-10AM	11PM-12AM	12PM-1PM	11AM-1PM	6PM-7PM	4PM-5PM	3PM-5PM
	Brooklyn Node	1AM-2AM	9AM-10AM	11PM-12AM	12PM-1PM	11AM-1PM	6PM-7PM	4PM-5PM	3PM-5PM
	Long Island Node	1AM-2AM	9AM-10AM	11PM-12AM	12PM-1PM	11AM-1PM	6PM-7PM	4PM-5PM	3PM-5PM
	Date	Wed 03/15/06	Sat 12/24/05	Fri 12/23/05	Sun 02/19/06	Tue 02/07/06	Sun 02/19/06	Sun 02/19/06	Tue 03/21/06

Table 6- 6 Summary of Regional Simulation on Two Near-Proximity Medical Knowledge Nodes

Location	City	Woodside	Woodside	Woodside	Flushing Public Library	Midtown GC CUNY	Midtown GC CUNY	Downtown BMCC CUNY	CSI CUNY
	Borough	Queens	Queens	Queens	Queens	Manhattan	Manhattan	Mahattan	Staten Island
	Type	Resident	Resident	Resident	Government	Education	Education	Education	Education
Execution Time	Average	5244.50	3197.05	3095.05	3733.95	4357.90	2358.20	2264.90	4391.65
	Max	5848.00	5848.00	6069.00	6281.00	9761.00	2788.00	2531.00	5437.00
	Min	4657.00	2524.00	2584.00	2702.00	2196.00	2173.00	2063.00	2562.00
	Variance	326.54	709.96	854.13	1015.70	2487.08	166.41	138.36	873.06
Distance from Knowledge Node to Client (miles)	Brooklyn Node	7	7	7	11	5	5	5	11
	San Francisco Node	2573	2573	2573	2577	2568	2568	2568	2563
Client Connection	Speed (kbps)	56	3,000	3,000	10,000	45,000	45,000	45,000	45,000
	Media Type	Modem	DSL	DSL	Ethernet	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	1024	768	1024	256	256	256	512	128
	CPU (GHz)	Pentium IV 1.8	Celeron 2.8	Pentium IV 1.8	Pentium IV 1.6	Pentium IV 1.9	Pentium IV 1.9	Pentium IV 2.8	Pentium IV 3.0
Execution Hour	Client	1AM-2AM	11PM-12AM	12AM-1AM	1PM-2PM	12PM-1PM	6PM-7PM	4PM-5PM	3PM-5PM
	Brooklyn Node	1AM-2AM	11PM-12AM	12AM-1AM	1PM-2PM	12PM -1PM	6PM-7PM	4PM-5PM	3PM-5PM
	San Francisco Node	10PM-11PM	8PM-9PM	9PM-10PM	10AM-11AM	9AM-10AM	3PM-4PM	1PM-2PM	12PM-2PM
	Date	Wed 03/15/06	Sun 01/15/06	Thu 01/05/06	Sun 02/19/06	Tue 02/07/06	Sun 02/19/06	Sun 02/19/06	Tue 03/21/06

Table 6- 7 Summary of Regional Simulation on Two Far-Proximity Medical Knowledge Nodes

The next graph in Figure 6- 34 shows the comparison among client nodes at regional area of New York City when running on three knowledge nodes located at Brooklyn, Long Island, and San Francisco.

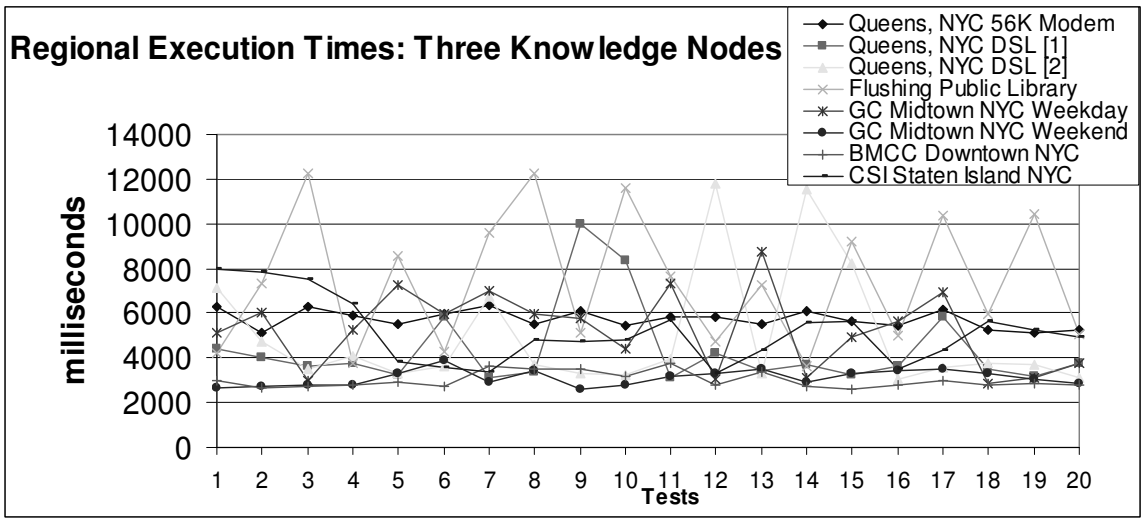


Figure 6- 34 Comparison of Tests among Nodes at Regional Simulation on Three Knowledge Nodes

The graph of three knowledge nodes is quite fluctuating to all clients in regional simulations including the tests on weekend at midtown and downtown Manhattan. When compared to others, the variances of the tests on weekend at midtown and downtown are very low, 357 and 359 ms. whereas the variances of others are more than 1400 ms. However, the variances of the tests on weekend at midtown and downtown on three knowledge nodes are greater than the variances of the results on one and two knowledge nodes at the same location. The most fluctuation of the tests is the results from the high volume network at Flushing public library whose variance on the tests of three knowledge nodes is 2951 ms. Its average, 7404 ms., is also the highest compared to the other clients in the same category. The average execution times of all regional clients running on three medical knowledge nodes are shown in Figure 6- 35.

Regardless the low speed connection of 56K-modem at the first bar of Woodside, the results show high average but very low variance, 396 ms. which is very close to the variance of high speed Ethernet and weekend tests at midtown and downtown Manhattan. This also is similar to the results from the same client on one and two knowledge nodes which show high average but low variance.

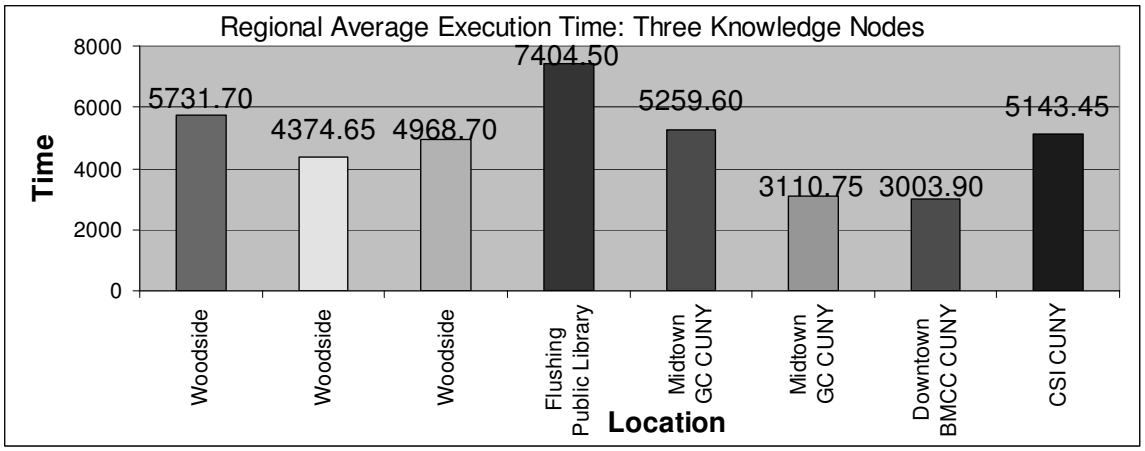


Figure 6- 35 Average Execution Times among Client Nodes at Regional Simulation on Three Medical Knowledge Nodes

The average execution times on three knowledge nodes in Figure 6- 35 show many different sets of results: low average times for the tests on weekend at the second bar of midtown and the bar of downtown, high average times for the tests on weekday at the first bar of midtown and the bar of Staten Island, CSI, no difference between two sets of hardware from the same location the second and third bar of Woodside, and high average times for the tests on high volume of internet usage at the bar of Flushing and the low speed internet of the first bar of Woodside

The followingTable 6- 8 shows the summary of the simulation on three medical knowledge nodes among clients in New York City region.

Location	City	Woodside	Woodside	Woodside	Flushing Public Library	Midtown GC CUNY	Midtown GC CUNY	Downtown BMCC CUNY	CSI CUNY
	Borough	Queens	Queens	Queens	Queens	Manhattan	Manhattan	Mahattan	Staten Island
	Type	Resident	Resident	Resident	Government	Education	Education	Education	Education
Execution Time	Average	5731.70	4374.65	4968.70	7404.50	5259.60	3110.75	3003.90	5143.45
	Max	6349.00	9994.00	11807.00	12258.00	8735.00	3915.00	3781.00	7985.00
	Min	5107.00	3134.00	3025.00	3608.00	2828.00	2605.00	2625.00	3296.00
	Variance	396.16	1837.84	2708.87	2951.86	1725.63	357.39	359.53	1435.03
Distance from Knowledge Node to Client (miles)	Brooklyn Node	7	7	7	11	5	5	5	11
	Long Island Node	39	39	39	34	44	44	44	54
	San Francisco Node	2573	2573	2573	2577	2568	2568	2568	2563
Client Connection	Speed (kbps)	56	3,000	3,000	10,000	45,000	45,000	45,000	45,000
	Media Type	Modem	DSL	DSL	Ethernet	T3	T3	T3	T3
Client Computer	Memory (Mbyte)	1024	768	1024	256	256	256	512	128
	CPU (GHz)	Pentium IV 1.8	Celeron 2.8	Pentium IV 1.8	Pentium IV 1.6	Pentium IV 1.9	Pentium IV 1.9	Pentium IV 2.8	Pentium IV 3.0
Execution Hour	Client	1AM-2AM	11PM-1AM	9PM-11PM	1PM-3PM	12PM-1PM	6PM-7PM	4PM–5PM	3PM-5PM
	Brooklyn Node	1AM-2AM	11PM-1AM	9PM-11PM	1PM-3PM	12PM-1PM	6PM-7PM	4PM–5PM	3PM-5PM
	Long Island Node	1AM-2AM	11PM-1AM	9PM-11PM	1PM-3PM	12PM-1PM	6PM-7PM	4PM-5PM	3PM-5PM
	San Francisco Node	10PM-11PM	8PM-10PM	6PM-8PM	10AM-12PM	9AM-10AM	3PM-5PM	1PM-2PM	12PM-2PM
	Date	Wed 03/15/06	Fri 01/13/06	Sun 01/15/06	Sun 02/19/06	Tue 02/07/06	Sun 02/19/06	Sun 02/19/06	Tue 03/21/06

Table 6- 8 Summary of Regional Simulation on Three Medical Knowledge Nodes

6.4.Real-Time National Simulation

This section presents the results of tests that compare testing done in various states of United States. Most of the tests are in residential areas, but one test is located in the government facility and one test is located in the business area. The details are explained in the next section, Topology of National Simulation. In section 6.4.2, the results are compared between single and multiple knowledge nodes of each node in national network whereas the comparisons among nodes in national network are shown in section 6.4.3.

6.4.1. Topology on National Simulation

The simulations under the national topology are among different locations in United States. This national study includes five different locations. One location ran the tests on two different settings:

- Whippany, Morristown New Jersey, a light-density resident area. At this location, we ran the tests in two different settings of computers. One of the computers is slower on speed of processor and lower capacity of memory than the other. Both PC sets connect to internet via high speed cable modem using fiber optics.

- Houston, Texas, also a resident area. The tests at Houston ran on a big complex apartment that shares internet connection among residents of the building.

- Durham, North Carolina where is inside a government facility using a very high speed internet but also sharing simultaneously the internet connection with hundreds of users. The tests were run during the weekday.

- Chapel Hill, North Carolina, a light-resident area not far from the previous location at Durham. In contrast with the tests at Durham, NC, these simulations ran via low speed 56k-modem to connect to the internet. Some tests were run during weekend and some during the weekday.

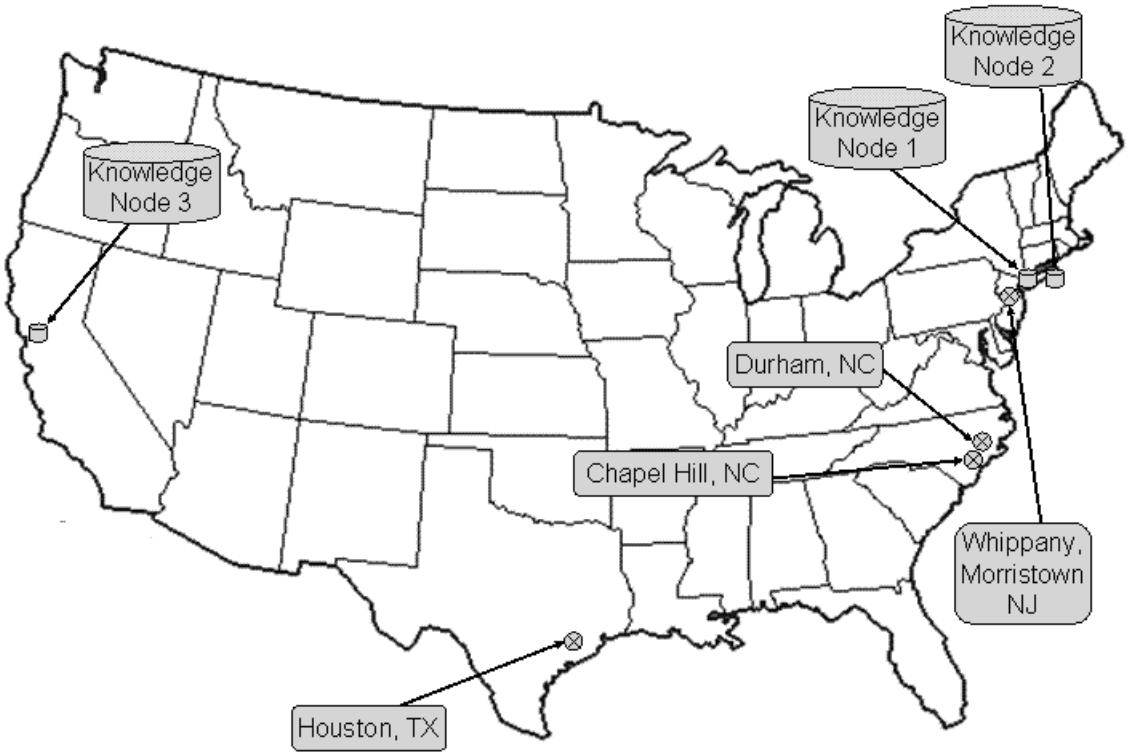


Figure 6- 36 National Map. Client Nodes are located at Whippany Morristown NJ, Houston TX, Durham NC, and Chapel Hill NC. Knowledge Nodes are located at Brooklyn NY, Islandia Long Island NY, and San Francisco CA.

All computers ran on national simulation are Windows-based. Most of them run on Windows XP operating system except the client at Chapel Hill which runs on Windows 98. Figure 6- 36 shows the national map and location of clients and knowledge nodes whereas Figure 6- 37 shows the topology of the national simulation.

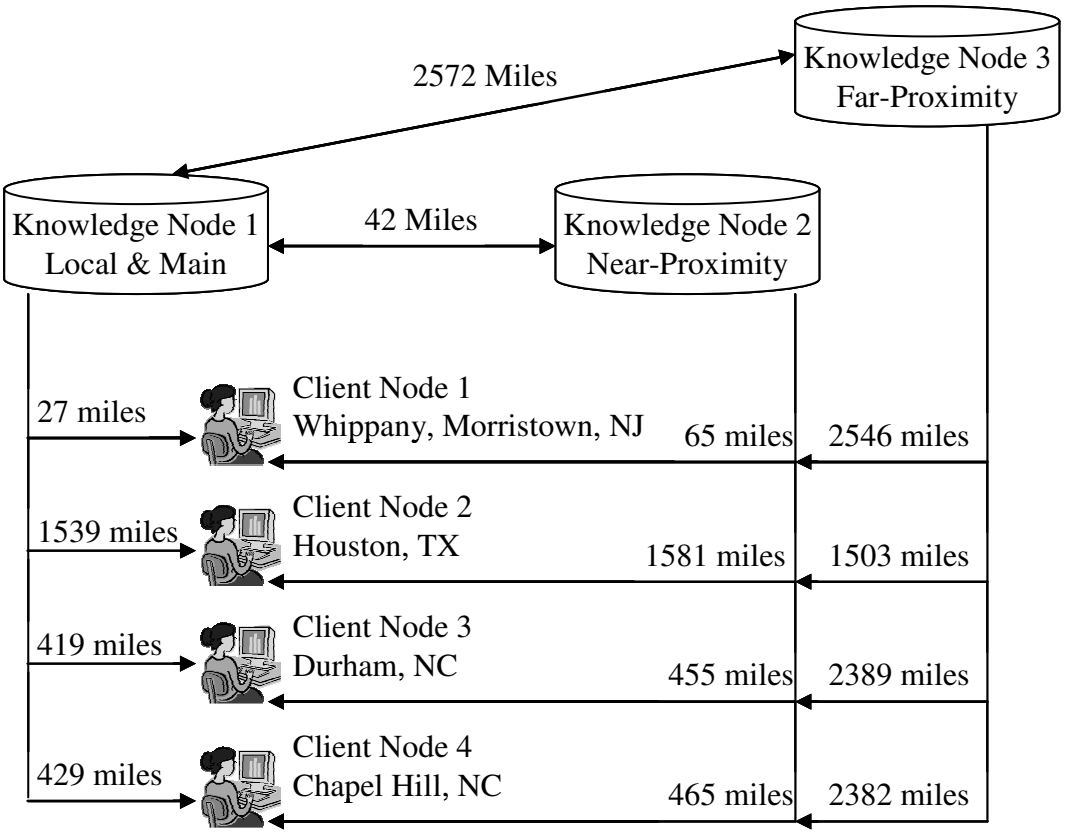


Figure 6- 37 Topology of National Simulation in United States

In the national simulation, the distances from client nodes to medical knowledge nodes are far apart. The closest client to the main medical knowledge node in Brooklyn NY is at Whippany in New Jersey whose distance is only 27 miles from the knowledge node at Brooklyn and 65 miles from the medical knowledge node at Long Island but 2546 miles from the medical knowledge node at San Francisco. Whereas, the client nodes at North Carolina are also about 400 miles away from the knowledge nodes at Brooklyn and Long Island and more than 2000 miles away from the knowledge node at San Francisco.

The only node that is in the middle among all knowledge nodes is at Houston where its distances from any medical knowledge nodes are almost the same; 1539, 1581, and 1503 miles away from the knowledge nodes at Brooklyn, Long Island, and San Francisco sequentially.

6.4.2. Results and Comparison between Single and Multiple Knowledge Nodes on National Simulation

The first set of national simulations was run at Whippany, Morristown NJ. At this location, we ran the tests in two different settings; PC set 1 using Pentium III 333 MHz processor with 512 Mbytes of memory and PC set 2 running on Celeron 2.8 GHz processor and 768 Mbytes of memory. Both PC sets connect to internet via high speed cable modem using fiber optics.

In the Figure 6- 38, graphs of the PC set 1, which is lower performance than PC set 2, is more stable than the graph of PC set 2. Only the tests of two far-proximity knowledge nodes of PC set 1 shows a big variance, 1634 ms while variances of other tests by PC set 1 is less than 1000 ms. However the variance, 1634 ms., of the PC set 1 is still less than the variance of PC set 2 on the same test for two far-proximity knowledge nodes which is 2579 ms.

Not only the tests on two far-proximity knowledge nodes of the PC set 2 show a big variance but also its tests on two near-proximity knowledge nodes which is 1183 ms.

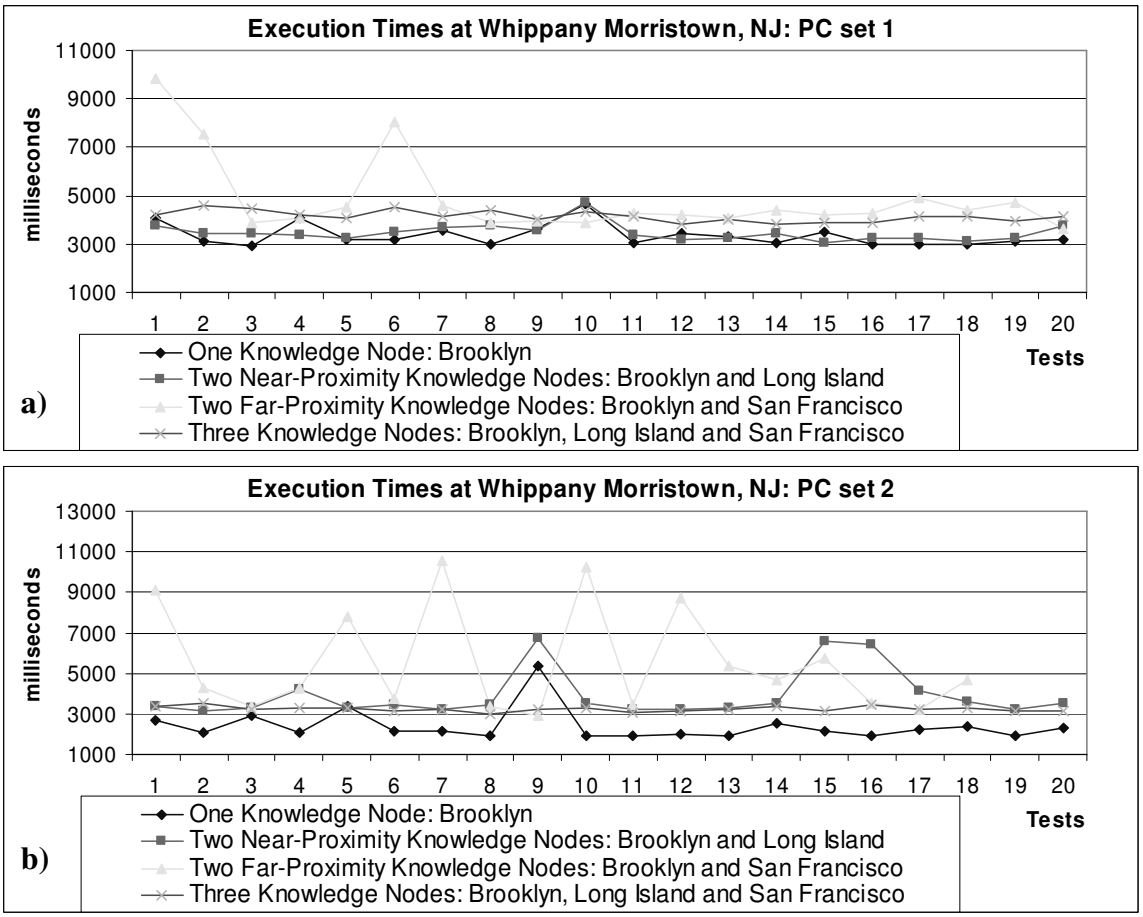


Figure 6- 38 Execution Times at Whippany, Morristown NJ of National Simulation a) Running 333 MHz Processor b) Running 2.8 GHz Processor.

Surprisingly, the variances of both PCs on three knowledge node are very small; 216 and 130 ms for PC set 1 and set 2 sequentially whereas their average is not as high as the averages of both two-near-proximity and two-far-proximity knowledge nodes of PC set 2. The reason is that the tests of both PCs on three knowledge nodes were on weekend and late night, when the internet traffic on nation wide was expected to be very light.

The second set of national tests was at Houston, Texas. The computer used on the tests is a laptop using Pentium IV 1.8 GHz processor with 1.5 Gbytes of memory. As mention earlier, the internet connection for this client is shared with other residents in the same building. So, the results in Figure 6- 39 are quite wide range at the tests of multiple knowledge nodes.

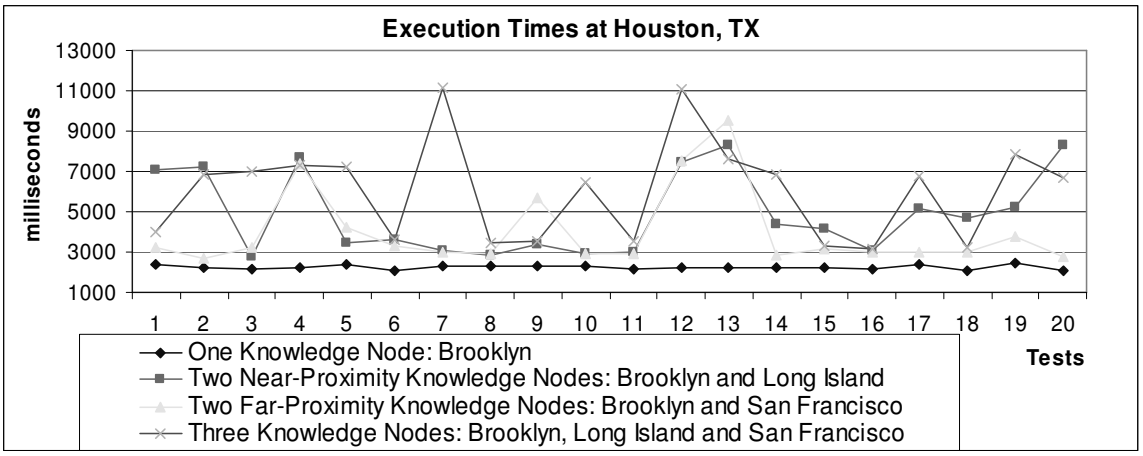


Figure 6- 39 Execution Times at Houston, TX of National Simulation

The variances of the simulation on two near-proximity knowledge nodes of the client at Houston are as big as 2022 ms which is much bigger than of the simulation at Whippany, NJ. The variance of the tests on two far-proximity knowledge nodes of client at Houston is also high as 1950 which is the similar to the test at Whippany on the same category. However, different from the tests at Whippany NJ, the variance of the tests on three knowledge nodes at Houston node is as big as 2475 ms. Considering the distance from Houston to all knowledge nodes, Houston is not closed to any knowledge nodes, more than 1500 miles. That should explain the big variances when executed on the multiple medical knowledge nodes.

The third set of national simulation was run at Durham, North Carolina. The tests ran on a computer using Pentium IV 2.4 GHz processor with 512 Mbytes of memory. However the tests were run during the weekday from a government building where many users are sharing the internet connection. Figure 6- 40 shows the results of testing at Durham, NC.

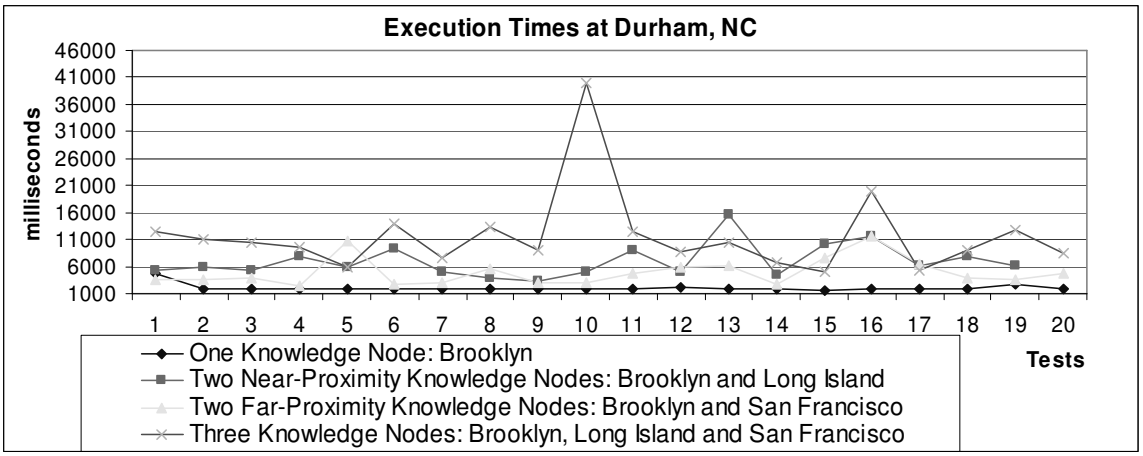


Figure 6- 40 Execution Times at Durham, NC of National Simulation

Because of high speed internet connection, regardless the number of user simultaneously connecting to the internet, the graph of the tests in Figure 6- 40 on one knowledge node shows the lowest average compared to any other clients of national simulation. But when testing on the multiple knowledge nodes, the large number of internet users at the facility at Durham affects their results. The variances and averages are very high compared to other nodes of national simulation (not including the client at Chapel Hill using 56k-modem), especially the tests of three knowledge nodes which is 7557 ms. for variance and 11627 ms. for the average. However, the tenth result of testing on three knowledge nodes draws its average and variance. Without the tenth result, its average and variance are 10129, and 3591 ms sequentially which is still high compared to other national client node in the same category.

North Carolina was also run on the fourth set of national simulations, not at Durham but at Chapel Hill, instead. These simulations ran on a laptop using the Pentium III 500 MHz with 256 Mbytes of memory and connecting to the internet via 56k-modem. Some tests were run during weekend and some during the weekday. Their results are shown in Figure 6- 41.

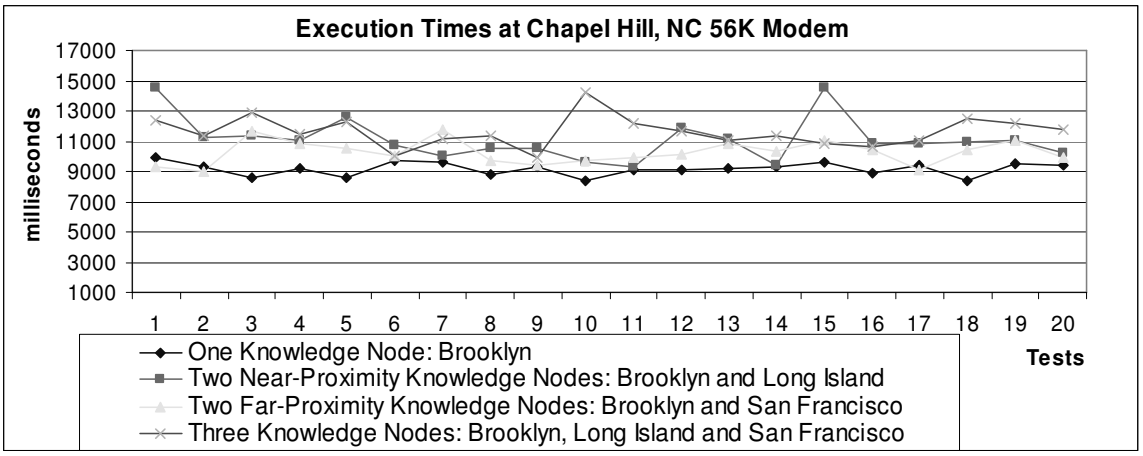


Figure 6- 41 Execution Times at Chapel Hill, NC of National Simulation

The results at Chapel Hill on national simulation using 56k-modem show high averages in Figure 6- 41 as expect for low speed connection on the internet. Similar to other tests using 56k-modem, the variances of the tests at Chapel Hill are not the highest comparing to other national client nodes on the same number of knowledge nodes because the execution times are greatly affected by the connection between ISP and the client.

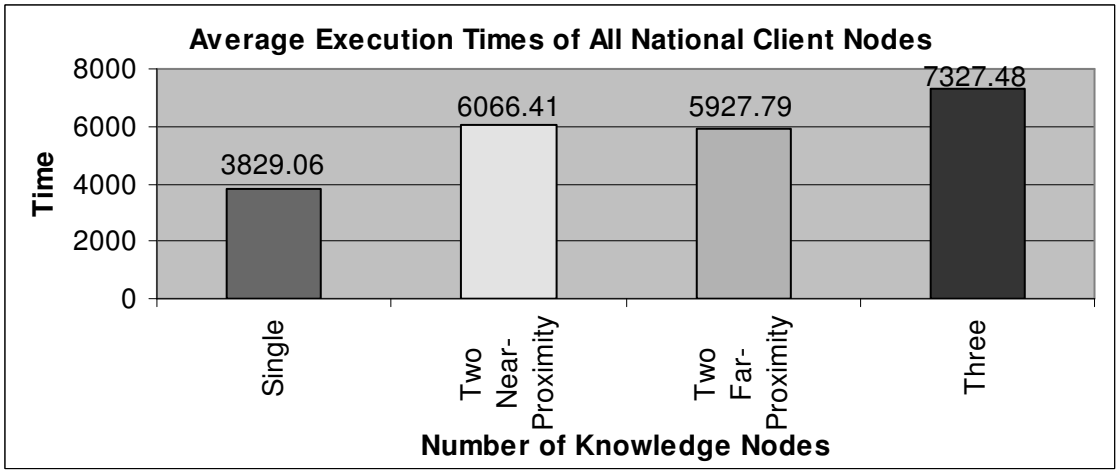


Figure 6- 42 Average Execution Times of all Clients for each Number of Knowledge Nodes at National Simulation

The mean between the average execution times of two near-proximity and of two far-proximity knowledge nodes from Figure 6- 42 is (6066.41 + 5927.79) / 2 = 5997.1 ms. But the mean between the average of execution time of one and of three knowledge nodes is (3829.06 + 7327.48)/2 = 5578.27 ms., which is different from the mean of two types of two knowledge nodes.

Using the means, we can find the average execution time when the number of knowledge is increased:

5578.27 – 3829.06 = 1749.21

from one knowledge node –to- the mean between one and three knowledge nodes

7327.48 – 5578.27 = 1749.21

from the mean between one and three knowledge nodes –to- three knowledge nodes

5997.1 – 3829.06 = 2168.04

from one knowledge node –to- the mean of two knowledge nodes

7327.48 – 5997.1 = 1330.38

from the mean of two knowledge nodes to three knowledge nodes

The average is (1749.21 +1749.21 + 2168.04 + 1330.38)/4 = 1749.21

That accumulates approximately 1749.21 milliseconds of execution time when increasing the number of knowledge nodes by one (from one node to two nodes and from two nodes to three nodes.) on the national execution.

If 1749.21 milliseconds is the difference of execution times between n and n+1 knowledge nodes and the average execution time of the single node is 3829.06 milliseconds, then 3829.06– 1749.21 = 2079.85 ms. that is the average set up time of national simulation when **not** considering the execution time at a knowledge node.

6.4.3. Comparison among Client Nodes on National Simulation

In this section, we show the comparison among client nodes in national simulation by considering separately each number of knowledge nodes.

The next graph in Figure 6- 43 shows the comparison among different nodes in the national simulation on the execution of one knowledge node.

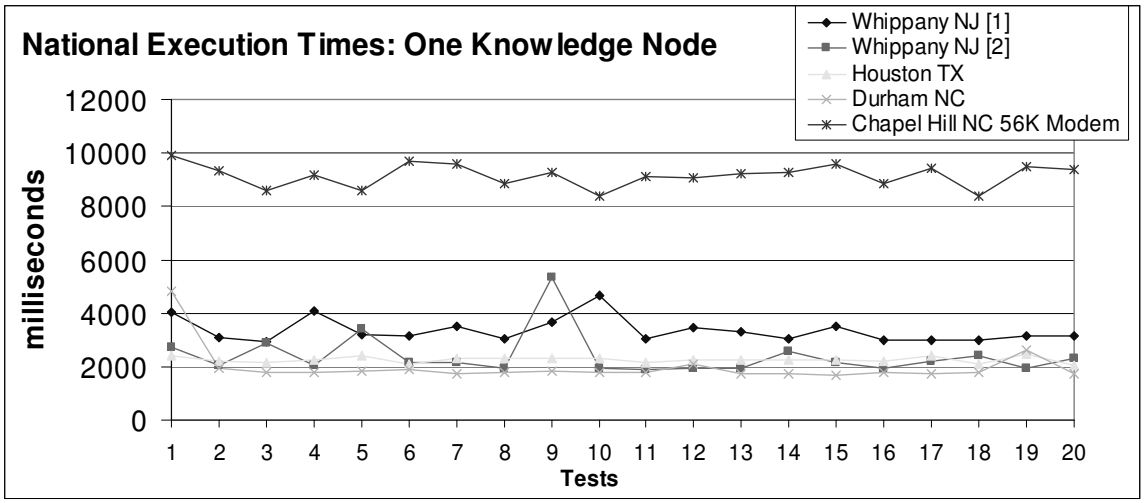


Figure 6- 43 Comparison of Tests among Nodes at National Simulation on One Knowledge Node

The graphs of simulations on one knowledge node are smooth for almost all national clients whose variances are less than 500 ms except for the Whippany set 2, and Durham because one of their results deviates from others. However, overall the variances for national clients are higher compared to the local and regional simulations.

The next graph in Figure 6- 44 shows the average execution times of all national clients running on one medical knowledge node.

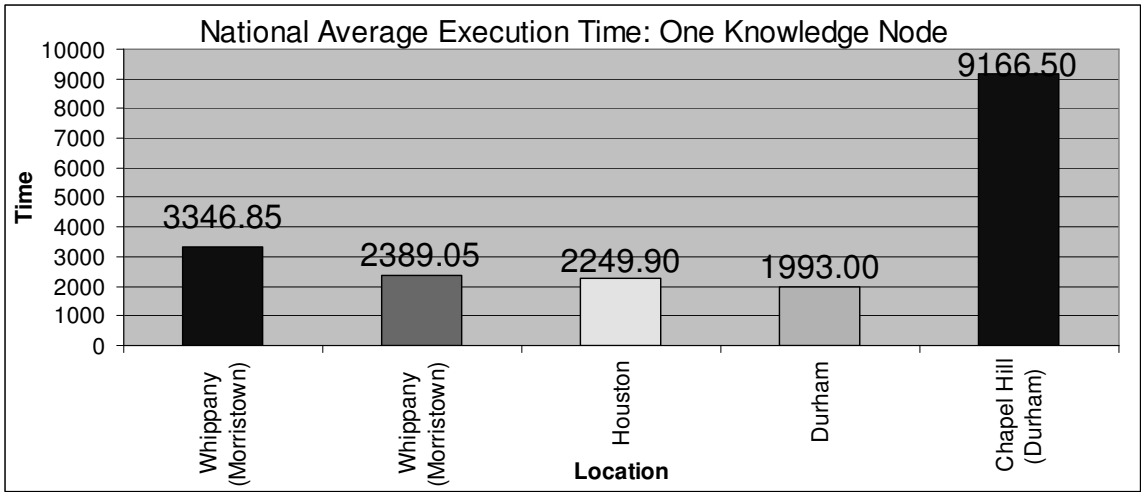


Figure 6- 44 Average Execution Times among Client Nodes at National Simulation on One Medical Knowledge Nodes

The average of the tests at Chapel Hill is very high, compared to the others, due to the low speed internet connection of 56K-modem. That is normal. In Figure 6- 44, two sets of tests ran on the same location at Whippany but their average are much different because the first set of the tests at Whippany used a low performance computer, Pentium III 333 MHz processor, while the second set of the tests at Whippany ran on the same condition but using Celeron 2.8 GHz processor.

The average times of two sets of tests at Houston, and Durham are small because the tests ran from organizations using high speed internet. So the tests on the high speed internet with typical speed of computer do not affect the average of execution time on single knowledge node.

The summary of the tests on single knowledge node running by clients at national location is shown in the following Table 6- 9.

Client Location	City	Whippany (Morristown)	Whippany (Morristown)	Houston	Durham	Chapel Hill (Durham)
	State	New Jersey	New Jersey	Texas	North Carolina	North Carolina
	Type	Resident	Resident	Resident	Government	Resident
Execution Time	Average	3346.85	2389.05	2249.90	1993.00	9166.50
	Max	4657.00	5348.00	2453.00	4828.00	9890.00
	Min	2934.00	1883.00	2047.00	1672.00	8400.00
	Variance	460.46	799.27	112.97	697.69	433.41
Distance from Knowledge Node to Client (miles)	Brooklyn Node	27	27	1539	419	429
Client Connection	Speed (kbps)	30,000	30,000	512	45,000	56
	Media Type	Cable Modem	Cable Modem	DSL	DS3	Modem
Client Computer	Memory (Mbyte)	512	768	1536	512	256
	CPU (GHz)	Pentium III 0.33	Celeron 2.8	Pentium IV 1.8	Pentium IV 2.4	Pentium III 0.5
Execution Hour	Client	7 AM - 9 AM	12 AM - 1 AM	10 PM - 11 PM	10 AM - 11 AM	4 PM - 5 PM
	Brooklyn Node	7 AM - 9 AM	12 AM - 1 AM	11 PM - 12 AM	10 AM - 11 AM	4 PM - 5 PM
	Date	Sun/12/04/05	Sun/12/18/05	Multiple Days	Thu/02/23/06	Sat/02/25/06

Table 6- 9 Summary of National Simulation on Single Medical Knowledge Node

Two following charts compare the results of tests on two knowledge nodes by national clients. Figure 6- 45 a) shows on two near-proximity medical knowledge nodes and Figure 6- 45 b) shows on two far-proximity medical knowledge nodes.

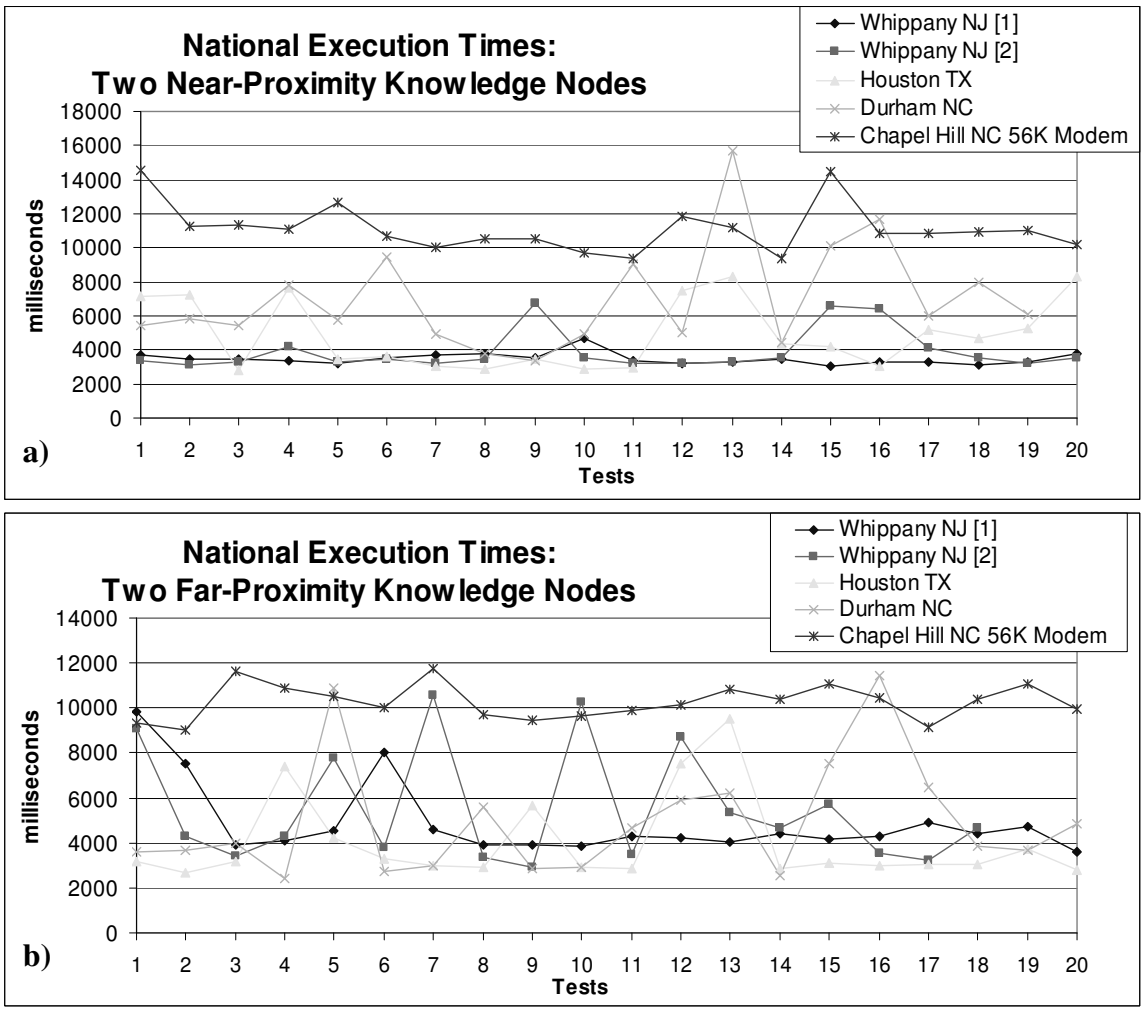


Figure 6- 45 Comparison of Tests among Nodes at National Simulation on Two Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

The graphs in Figure 6- 45 fluctuate a lot because of the real time statistics. Most of the tests on two knowledge nodes at national client were run on weekday. The tests at Durham on both near and far proximity knowledge nodes show high variances, 3092 and 2574 milliseconds sequentially because which they were run inside office building. In addition, the tests at Houston running inside a big resident complex sharing internet

connection show high variances as well, 2022 ms. for near-proximity and 1950 ms. for far-proximity. Because the tests at Houston were run at night on weekday, their variances are smaller than of Durham.

The results of two sets of computers run at Whippany show that different sets of computer do not affect the result on two knowledge nodes. Both computers at Whippany show low variance for the tests on two near-proximity knowledge nodes; 361 ms for PC set 1 which is low performance computer and 1183 ms for PC set 2 using faster processor than PC set 1. Both computers at Whippany show higher variance for the tests on two far-proximity knowledge nodes than on two near-proximity knowledge nodes. The variances for the tests on two far-proximity knowledge nodes are 1634 ms for PC set 1 and 2579 ms for PC set 2.

Two charts in Figure 6- 46 compare the average execution times among clients at national simulation. Figure 6- 46 a) compares clients running the tests on two near-proximity knowledge nodes while Figure 6- 46 b) compares on two far-proximity knowledge nodes.

Both charts show high averages for the tests at Chapel Hill because of the low speed internet but in Figure 6- 45 the tests at Chapel Hill show the lowest variance on two far-proximity knowledge nodes and medium variance on two near-proximity knowledge nodes comparing to other clients in national simulation. So the fluctuation is not affected by the low speed internet connection of the national tests, but not the case on the average execution time which is always high in any scenarios.

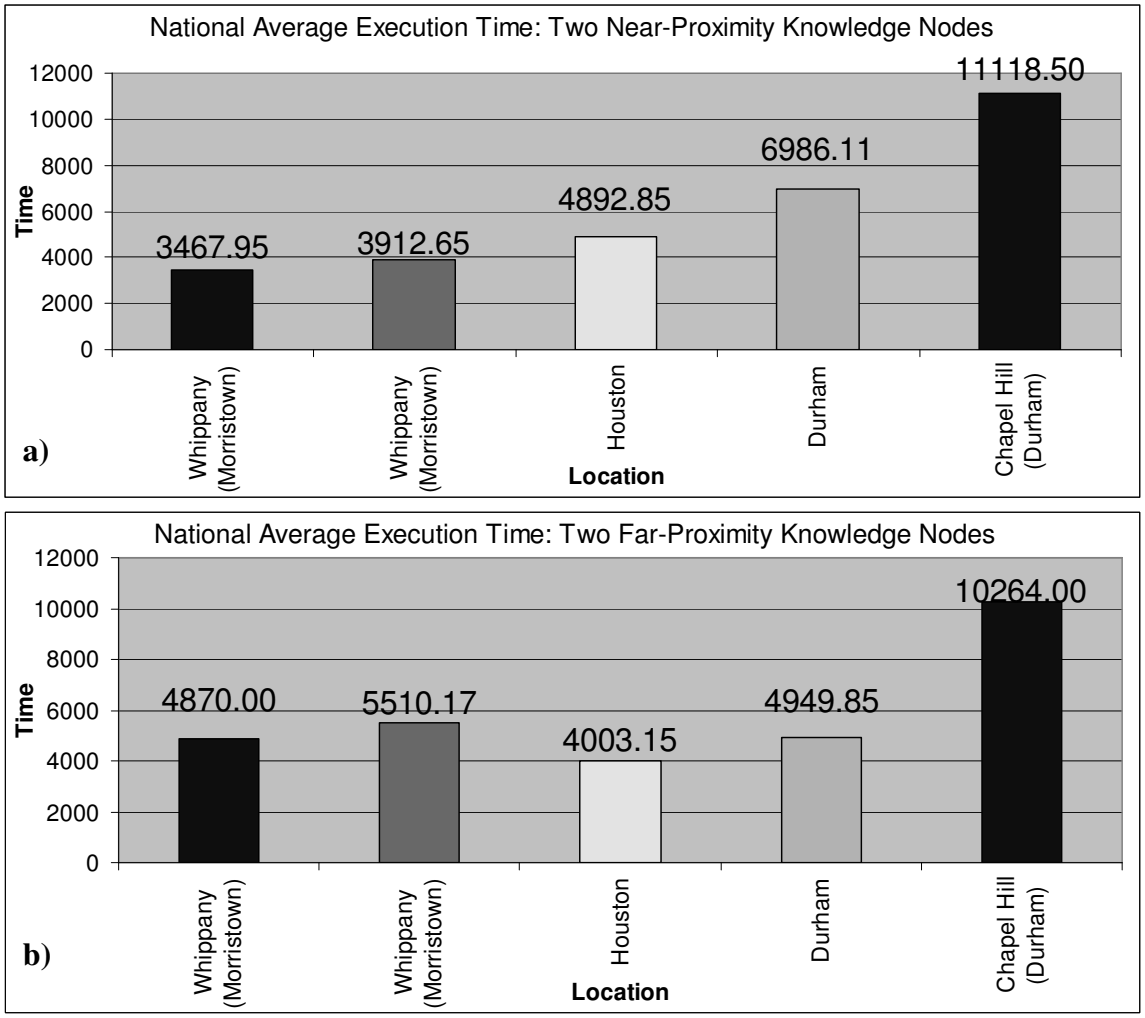


Figure 6- 46 Average Execution Times among Client Nodes at National Simulation on Two Medical Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

Similar to the variance in Figure 6- 45, the average execution times are not affected by the type of computers running on the same criteria. Averages of two set of computers at Whippany in Figure 6- 46 a) are less than both averages at Whippany Figure 6- 46 b).

Even though Whippany tests are considered national because it is not in New York City, it is not far from knowledge nodes at Brooklyn and Long Island which are two near-proximity knowledge nodes; 27 miles from Brooklyn and 65 miles from Long Island.

While, the tests at Durham, Chapel Hill and Houston, are much further from both knowledge nodes; about 400 to 500 miles to Durham and Chapel Hill and about 1500 miles to Houston. The results in Figure 6- 46 a) show the average time of client at far distance is higher than of the client at close distance.

However, considering Figure 6- 46 b) of two far-proximity knowledge nodes; Brooklyn and San Francisco, the distances from San Francisco to Houston, Durham, and Chapel Hill are higher than of Whippany and the balance of distances occurs among national client (but not Whippany) and knowledge nodes. The distances between San Francisco to client and client to Brooklyn are:

San Francisco <- 1503 miles -> Houston <- 1539 miles -> Brooklyn

San Francisco <- 2389 miles -> Durham <- 419 miles -> Brooklyn

San Francisco <- 2382 miles -> Chapel Hill <- 429 miles -> Brooklyn

But the balance is much less between knowledge nodes and Whippany

San Francisco <- 2546 miles -> Whippany **<- 27 miles ->** Brooklyn

Because of the balance on these distances, the average times of clients at Houston, Durham, and Chapel Hill in Figure 6- 46 b), which is far-proximity, are much less than the average time of the same client in Figure 6- 46 a), which is near-proximity. Also the average execution time at Houston is the least among clients on the test on two far-proximity knowledge nodes because of the most balancing distance from Houston to two knowledge nodes at far-proximity.

The following two table; Table 6- 10 and Table 6- 11, show the summary of the tests running from clients at national level to two near-proximity knowledge nodes and to two far-proximity knowledge nodes sequentially.

Client Location	City	Whippany (Morristown)	Whippany (Morristown)	Houston	Durham	Chapel Hill (Durham)
	State	New Jersey	New Jersey	Texas	North Carolina	North Carolina
	Type	Resident	Resident	Resident	Government	Resident
Execution Time	Average	3467.95	3912.65	4892.85	6986.11	11118.50
	Max	4717.00	6760.00	8297.00	15719.00	14550.00
	Min	3064.00	3115.00	2797.00	3391.00	9340.00
	Variance	361.85	1183.45	2022.77	3092.68	1407.65
Distance from Knowledge Node to Client (miles)	Brooklyn Node	27	27	1539	419	429
	Long Island Node	65	65	1581	455	465
Client Connection	Speed (kbps)	30,000	30,000	512	45,000	56
	Media Type	Cable Modem	Cable Modem	DSL	DS3	Modem
Client Computer	Memory (Mbyte)	512	768	1536	512	256
	CPU (GHz)	Pentium III 0.33	Celeron 2.8	Pentium IV 1.8	Pentium IV 2.4	Pentium III 0.5
Execution Hour	Client	9 PM - 10 PM	9 PM - 10 PM	8 AM - 10 PM	11 AM - 2 PM	9 PM - 10 PM
	Brooklyn Node	9 PM - 10 PM	9 PM - 10 PM	9 AM - 11 PM	11 AM - 2 PM	9 PM - 10 PM
	Long Island Node	9 PM - 10 PM	9 PM - 10 PM	9 AM - 11 PM	11 AM - 2 PM	9 PM - 10 PM
	Date	Sat/12/24/05	Thu/12/29/05	Multiple Days	Thu/02/23/06	Wed/03/15/06

Table 6- 10 Summary of National Simulation on Two Near-Proximity Medical Knowledge Nodes

Client Location	City	Whippany (Morristown)	Whippany (Morristown)	Houston	Durham	Chapel Hill (Durham)
	State	New Jersey	New Jersey	Texas	North Carolina	North Carolina
	Type	Resident	Resident	Resident	Government	Resident
Execution Time	Average	4870.00	5510.17	4003.15	4949.85	10264.00
	Max	9854.00	10585.00	9531.00	11468.00	11750.00
	Min	3625.00	2904.00	2657.00	2453.00	9010.00
	Variance	1634.53	2579.16	1950.05	2574.16	783.93
Distance from Knowledge Node to Client (miles)	Brooklyn Node	27	27	1539	419	429
	San Francisco Node	2546	2546	1503	2389	2382
Client Connection	Speed (kbps)	30,000	30,000	512	45,000	56
	Media Type	Cable Modem	Cable Modem	DSL	DS3	Modem
Client Computer	Memory (Mbyte)	512	768	1536	512	256
	CPU (GHz)	Pentium III 0.33	Celeron 2.8	Pentium IV 1.8	Pentium IV 2.4	Pentium III 0.5
Execution Hour	Client	7 AM - 8 AM	8 PM - 9 PM	9 PM - 10 PM	1 PM - 2 PM	9 PM - 10 PM
	Brooklyn Node	7 AM - 8 AM	8 PM - 9 PM	10 PM - 11 PM	1 PM - 2 PM	9 PM - 10 PM
	San Francisco Node	4 AM - 5 AM	5 PM - 6 PM	7 PM - 8 PM	10 AM - 11 AM	6 PM - 7 PM
	Date	Fri/01/06/06	Sat/01/28/06	Thu/01/26/06	Thu/02/23/06	Wed/03/15/06

Table 6- 11 Summary of National Simulation on Two Far-Proximity Medical Knowledge Nodes

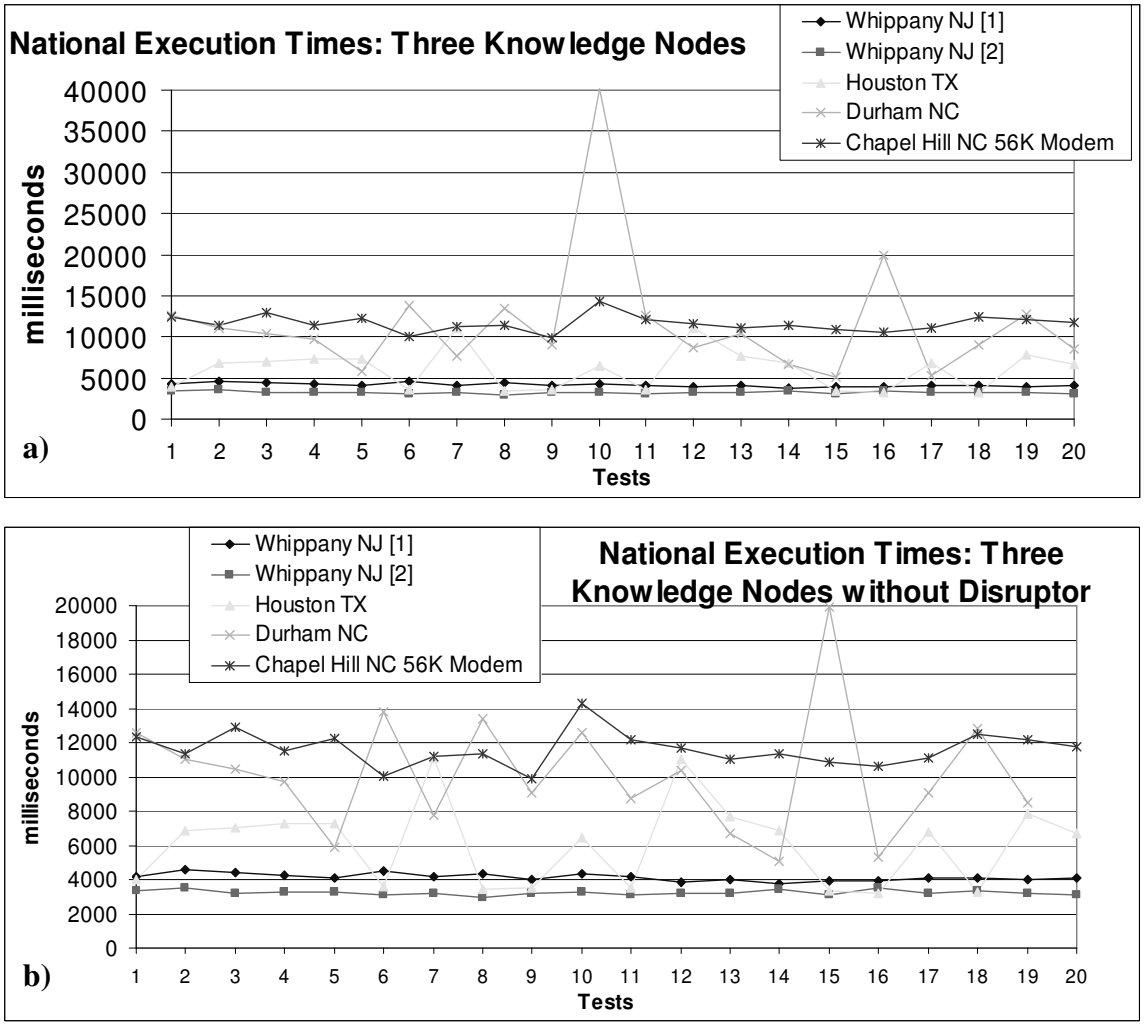


Figure 6- 47 Comparison of Tests among Nodes at National Simulation on Three Knowledge Nodes a) Complete results b) Not Include tenth tests of a)

Figure 6- 47compares the result among clients at national simulation running on three medical knowledge nodes. Without the one disruptive result at the tenth test on Durham in Figure 6- 47 a), Figure 6- 47 b) is the same result as in Figure 6- 47 a) but we see better comparison among clients and to the results on one knowledge node in Figure 6- 43 and on two knowledge nodes in Figure 6- 45.

The results on three knowledge nodes among clients at national level in Figure 6- 47 b) show low variances; 216 ms for PC set 1 and 130 ms for PC set 2, at Whippany

which is similar to results of national simulation on one medical knowledge node in Figure 6- 43. Similar to the tests on two near-proximity knowledge nodes, the results at Houston and Durham show very high variances; 2475ms and 7557 ms sequentially, because they ran during the high network traffic time on weekday, and from the high usage location. The next chart in Figure 6- 48 shows their average execution times.

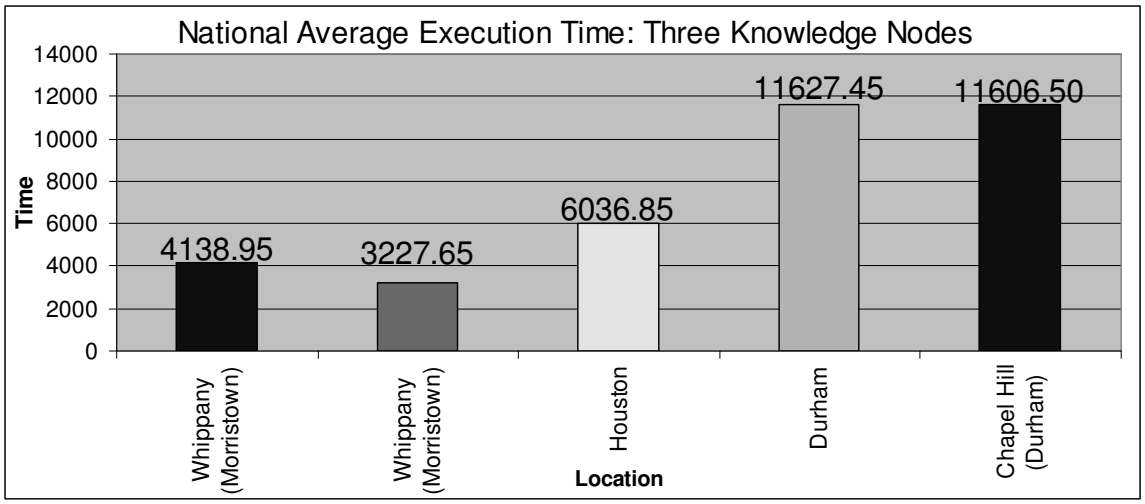


Figure 6- 48 Average Execution Times among Client Nodes at National Simulation on Three Medical Knowledge Nodes

Because the knowledge nodes at Brooklyn and Long Island are close, the distance from national clients to three medical knowledge nodes is not as balance as two far-proximity knowledge nodes. Two knowledge nodes are close and only one knowledge node is far. So the average execution times are in similar order to the results of two near-proximity knowledge nodes; low averages at Whippany and high averages at Houston, Durham, and Chapel Hill. The average at Durham is very high due to the time of execution which is high network traffic at every medical knowledge nodes and at the client itself. The following Table 6- 12 shows the summary of the tests among clients at national simulation running on three medical knowledge nodes.

Client Location	City	Whippany (Morristown)	Whippany (Morristown)	Houston	Durham	Chapel Hill (Durham)
	State	New Jersey	New Jersey	Texas	North Carolina	North Carolina
	Type	Resident	Resident	Resident	Government	Resident
Execution Time	Average	4138.95	3227.65	6036.85	11627.45	11606.50
	Max	4567.00	3495.00	11125.00	40094.00	14280.00
	Min	3795.00	2975.00	3172.00	5031.00	9890.00
	Variance	216.00	130.20	2475.08	7557.21	998.57
Distance from Knowledge Node to Client (miles)	Brooklyn Node	27	27	1539	419	429
	Long Island Node	65	65	1581	455	465
	San Francisco Node	2546	2546	1503	2389	2382
Client Connection	Speed (kbps)	30,000	30,000	512	45,000	56
	Media Type	Cable Modem	Cable Modem	DSL	DS3	Modem
Client Computer	Memory (Mbyte)	512	768	1536	512	256
	CPU (GHz)	Pentium III 0.33	Celeron 2.8	Pentium IV 1.8	Pentium IV 2.4	Pentium III 0.5
Execution Hour	Client	12 AM - 1 AM	12 AM - 1 AM	9 PM - 10 PM	1 PM - 2 PM	10 PM - 11 PM
	Brooklyn Node	12 AM - 1 AM	12 AM - 1 AM	10 PM - 11 PM	1 PM - 2 PM	10 PM - 11 PM
	Long Island Node	12 AM - 1 AM	12 AM - 1 AM	10 PM - 11 PM	1 PM - 2 PM	10 PM - 11 PM
	San Francisco Node	9 PM - 10 PM	9 PM - 10 PM	7 PM - 8 PM	10 AM - 11 AM	7 PM - 8 PM
	Date	Sun/01/22/06	Sun/01/22/06	Thu/01/26/06	Thu/02/23/06	Wed/03/15/06

Table 6- 12 Summary of National Simulation on Three Medical Knowledge Nodes

6.5.Real-Time Global Simulation

For global simulation, the tests are considering around the globe in many countries. The tests are in residential, educational and commercial areas. The details are explained in the section 6.5.1, Topology of Global Simulation. In section 6.5.2, the results are compared between single and multiple knowledge nodes of each node in global simulation whereas the comparisons among client nodes in global simulation are shown in section 6.5.3.

6.5.1. Topology on Global Simulation

For the global network, we tested across three continents, North America, Europe, and Asia. Knowledge servers are in United States while clients are in Munich Germany, Linz Austria, Bangkok Thailand, Chiang Mai Thailand, and Tokyo Japan.

Same as the local, regional, and national tests, there are four sets of testing:

- one knowledge node at Brooklyn site
- two near-proximity knowledge nodes at Brooklyn, and Long Island sites
- two far-proximity knowledge nodes at Brooklyn, and San Francisco sites
- three knowledge nodes at Brooklyn, Long Island and San Francisco sites

This global study includes six different locations:

- Munich, Germany where the tests were run at its medium-density business area. This location connects to internet via E3 line which is considered a high speed connection. But the tests were run during high network traffic at the business hour during the day at the client node. Due to 6-hour difference in time zone, the local time of the knowledge node is in morning which is also considered medium traffic on the knowledge nodes.

- Linz, Austria which is not far from the client node at Munich. However, the location of the client node is in the educational area, Johannes Kepler University of Linz. The client is located in an international housing complex whose internet is also shared among the students in the building, but still less traffic compared to the business building of the client node in Munich.

- Bangkok, Thailand in a high-density resident area where the client connects to the internet via 56K-modem. Due to the heavy internet traffic, client ran multiple periods of time on the tests of each number of knowledge nodes

- Bangkok, Thailand in a high-density business area where the client connects to the internet via high speed connection of the business. Due to 12-hour difference of time zone, the running time at client node is opposite with the local time of the knowledge node. There is always either client or knowledge node involving the high network traffic. The test at this location ran on the business hour at the client node.

- Chiang Mai, Thailand where the tests ran in a medium-density resident area. The client connects to the internet via Advanced DSL (ADSL) which is medium speed of internet connection. In addition, the tests were run at night time on the weekend when is low traffic on the knowledge nodes and very low traffic on the client node.

- Tokyo, Japan where the tests ran in a high-density commercial area. The client connects to internet via high speed connection of its company. This company has a small number of internet users. While the client ran the tests, only few workers are using the internet.

Figure 6- 49 Global Map. Client Nodes are located at Munich Germany, Linz Austria, Two locations in Bangkok Thailand, Chiang Mai Thailand, and Tokyo Japan. Knowledge Nodes are located at Brooklyn NY, Islandia Long Island NY, and San Francisco CA

Global map in Figure 6- 49 shows the locations of clients at Munich, Linz, Bangkok, Chiang Mai, and Tokyo and the locations of medical knowledge nodes; Brooklyn, Long Island, and San Francisco in US whereas the distances from a client node to a medical knowledge node are shown in Figure 6- 50. All clients in global simulation run on Windows-based operating system.

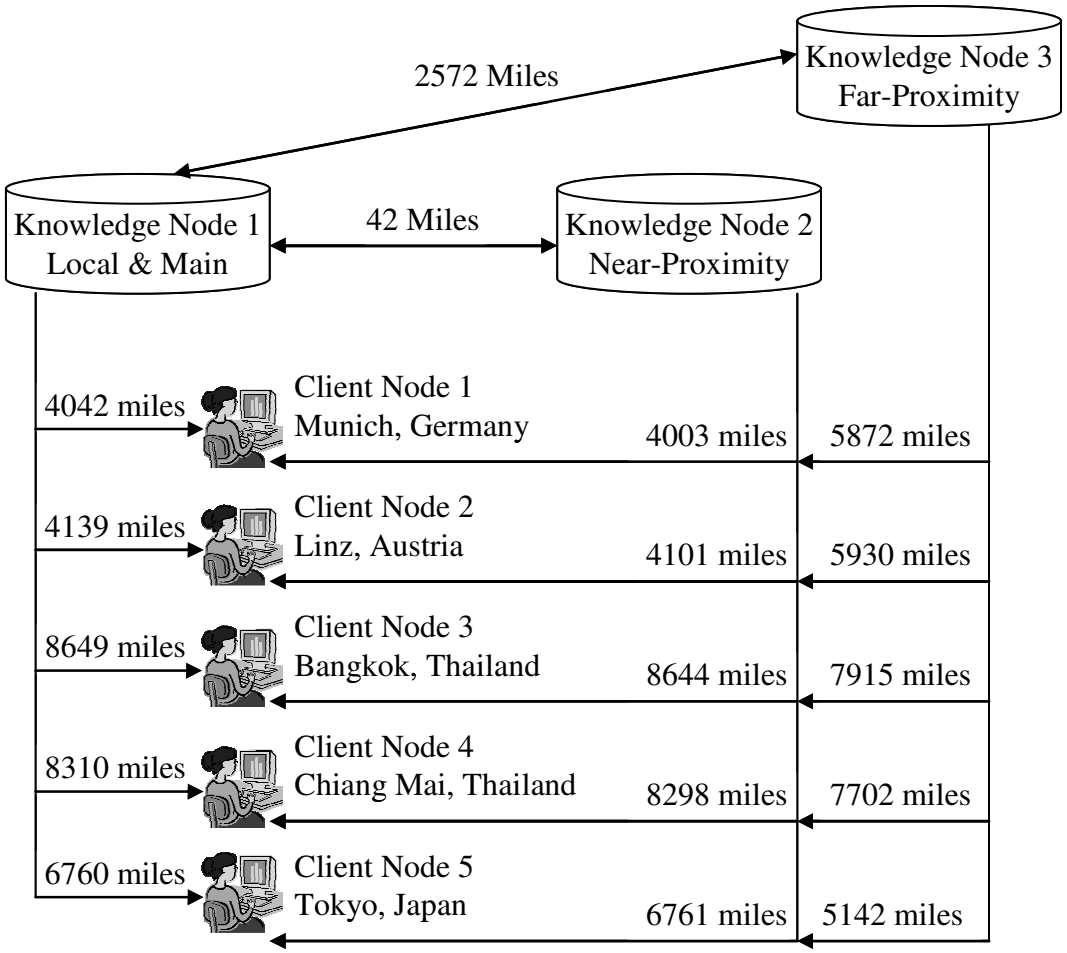


Figure 6- 50 Topology of Global Simulation in Europe and Asia

The distances from the client nodes in Europe; Munich and Linz, are about 4000 miles away to main knowledge node in Brooklyn and the knowledge node in Long Island and almost 6000 miles away to the knowledge node in San Francisco. In contrast, the distances from two clients in Thailand are twice of Europe's; about 8000 miles to the knowledge nodes in Brooklyn and Long Island which is almost the same distance to the San Francisco. Another location in Asia is in Tokyo whose distance to the knowledge nodes in Brooklyn and Long Island is about 6700 miles which is longer than the distances of clients in Europe to the same knowledge nodes. However the distance from client node

in Tokyo to knowledge node in San Francisco is about 5000 miles, which is closer than the distances from clients in Europe to the same knowledge node in San Francisco.

6.5.2. Results and Comparison between Single and Multiple Knowledge Nodes on Global Simulation

The first set of results of global simulation is in Munich Germany. The client ran the tests from a business building using a high speed internet with many users sharing the internet connection. The client ran the tests on a laptop whose processor is Pentium M 1.1 GHz with 384 Mbytes of memory. The results at this location of global simulation are show in Figure 6- 51.

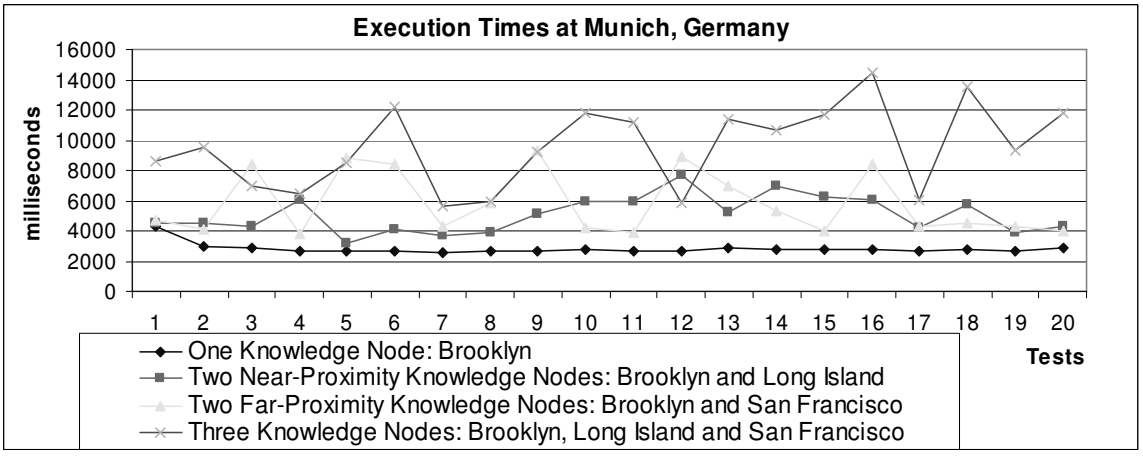


Figure 6- 51 Execution Times at Munich, Germany of Global Simulation

The results of tests in Munich, Germany in Figure 6- 51 show the similar graph to other graphs of national simulation which are stable for tests on one medical knowledge node and increased range and variance when more medical knowledge nodes are involved.

The variances are 370, 1201, 2071, and 2720 ms, for the tests on one, two-near-proximity, two-far-proximity, and three knowledge nodes sequentially, whereas their

averages are 2808, 5081, 5826, and 9549 ms. sequentially. Both variances and averages show similar ordering to the results in Figure 6- 52 at Linz, Austria.

The second set of global simulation is in Linz, Austria. The client ran the simulation from a resident building that shares high speed internet among residents who are mostly students at Johannes Kepler University but, mostly, were away from the apartment during the winter recess when we ran the tests. The client ran the tests using a computer whose processor is Pentium M 1.3 GHz with 768 Mbytes of memory. The results of global simulation at Linz, Austria are shown in Figure 6- 52.

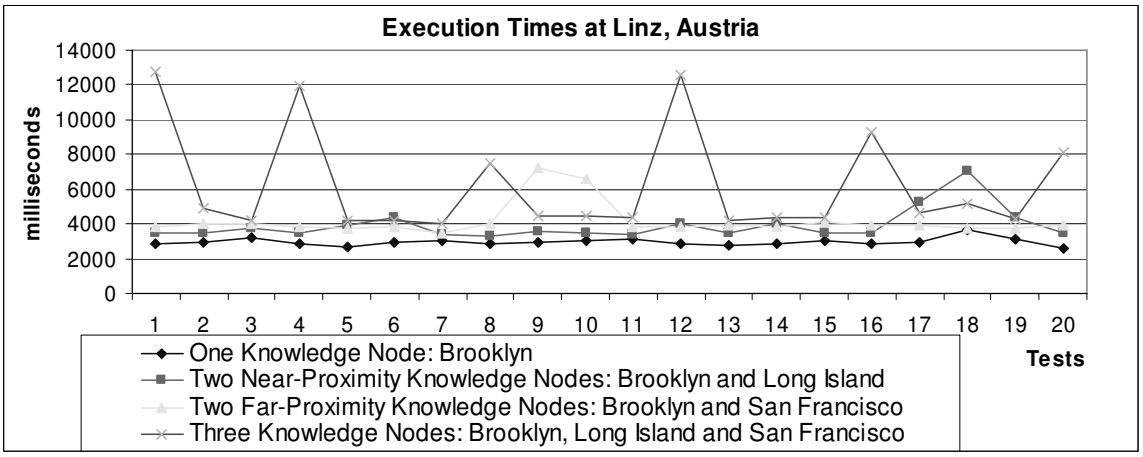


Figure 6- 52 Execution Times at Linz, Austria of Global Simulation

Due to the low network traffic during the tests, results in Figure 6- 52 show stable graph on one knowledge node, only three high points on two near-proximity knowledge nodes, and only two high points on two far-proximity knowledge nodes. The results of tests on three knowledge nodes are varied as usual when more knowledge nodes are involved.

The variances of client at Linz are 217, 878, 956, and 3061 ms, for the tests on one, two-near-proximity, two-far-proximity, and three knowledge nodes sequentially whereas their averages are 2956, 3902, 4167, and 6197 ms. sequentially which affirms the effect of the number of knowledge nodes.

The third location of global simulation is in Bangkok Thailand. There are two locations running the tests in Bangkok by using different type of internet connection. The first location is in high-density residential area and uses Pentium M 1.2 GHz with 256 Mbytes of memory to run the test connecting with 56K-modem. The second location in Bangkok is in high-density business area. Its test uses Pentium IV 1.8 GHz with 256 Mbytes of memory while connecting to the internet via high speed network of the company. Because the tests were run during the business hour inside the company, the high volume of network traffic was expected. The results of global simulation at Bangkok Thailand are shown in the following Figure 6- 53.

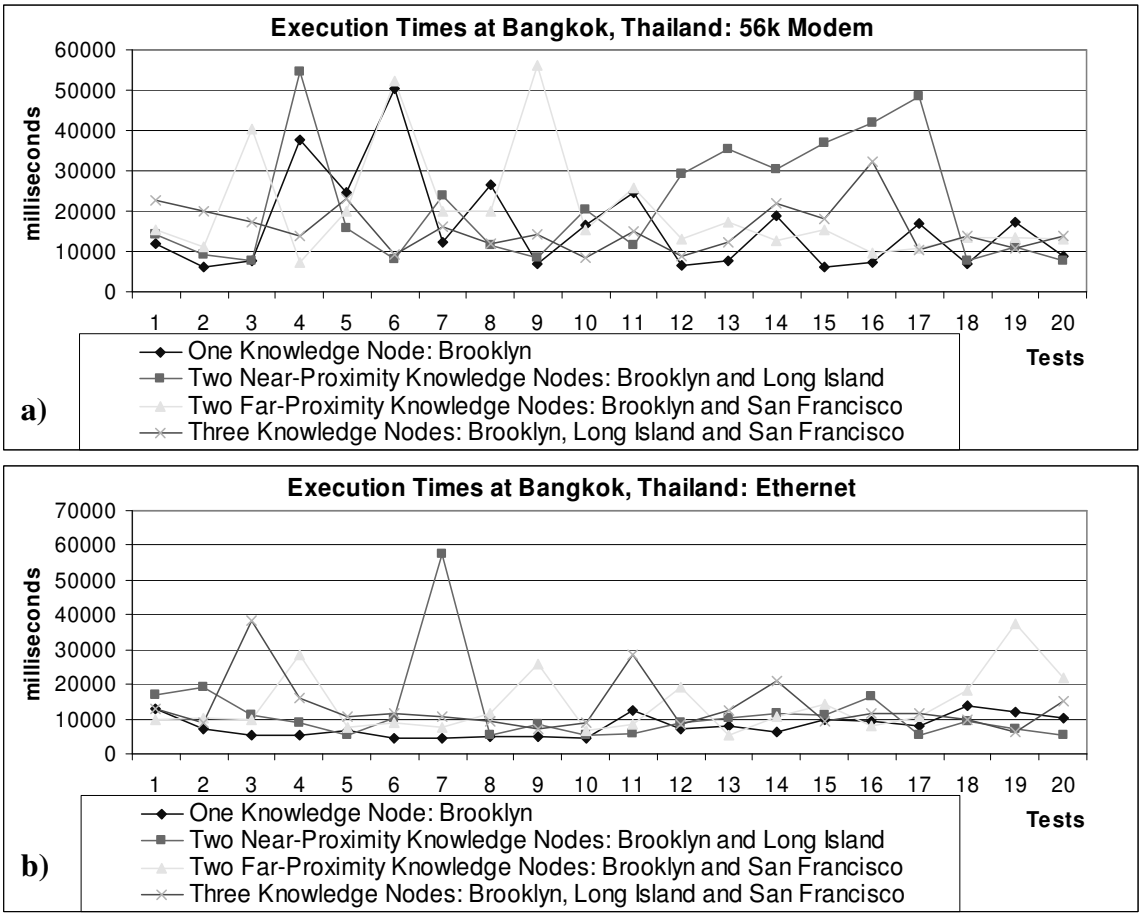


Figure 6- 53 Execution Times at Bangkok, Thailand of Global Simulation a) Using 56K Mode in Resident Building b) Using Eternet Connection in Commercial Building

Results in Figure 6- 53 are separated in two parts; Figure 6- 53 a) shows the results from the location using 56K-modem in a residential building and Figure 6- 53 b) shows the results from the location using high speed internet in a business building. Both Figure 6- 53 a) and Figure 6- 53 b) show high instability of results in both cases of low speed connection and high speed connection with high traffic.

The variances of results of client using 56K-modem at Bangkok are 11845, 14992, 13613, and 6012 ms., for the tests on one, two-near-proximity, two-far-proximity, and three knowledge nodes sequentially whereas their averages are 16124, 21736, 20121, and 15752 ms. sequentially which is the highest results comparing to any other nodes of global simulation in the same category.

Not only the results using 56K-Modem in Bangkok show very high numbers but also the results from Bangkok using high speed network with high volume of network traffic show high numbers. Their variances are 3148, 11428, 8569, and 7780 ms. for the tests on one, two-near-proximity, two-far-proximity, and three knowledge nodes sequentially whereas their averages are 7877, 11979, 14052, and 13439 ms. sequentially which is the second highest results comparing to any other nodes of global simulation in the same category.

The results from another city, Chiang Mai, in Thailand is lower value of execution time than of Bangkok but still considerably high compared to other nodes in global simulation. This draws to a conclusion about the distance between client node and knowledge nodes in which further the distance from the client to the knowledge node, longer the execution time. For these results, Thailand is the furthest away from the

knowledge nodes comparing to other global nodes and its average execution time is the most.

The next set of results was also run from Thailand in Chiang Mai, located in the northern part of Thailand. Not including Bangkok, which is the capital, Chiang Mai is one of the biggest cities in Thailand. The tests were run from a residential area using advanced DSL connecting to internet during the night time on weekend. So, the internet traffic was expected to be low. The tests ran on a personal computer using Pentium IV 1.8 GHz with 512 Mbytes of memory and its results are shown in Figure 6- 54.

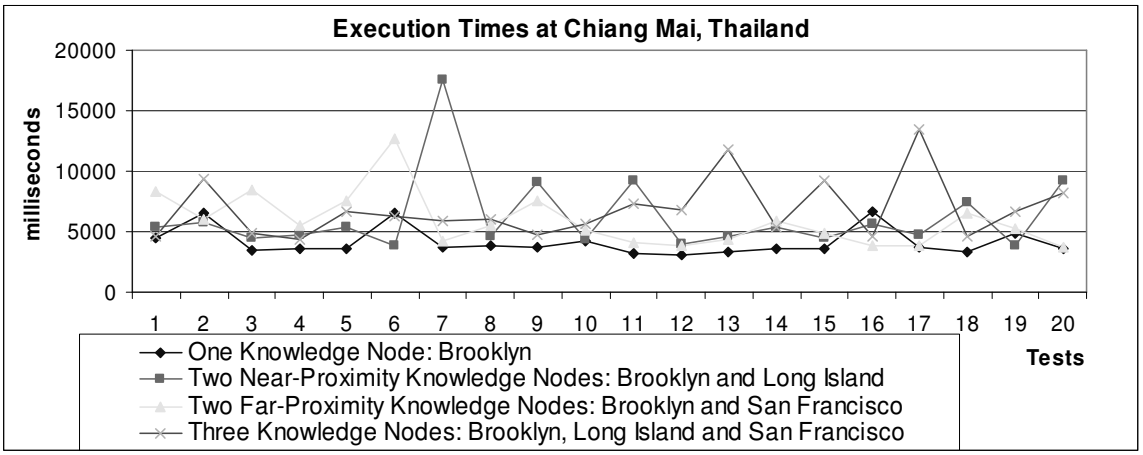


Figure 6- 54 Execution Times at Chiang Mai, Thailand of Global Simulation

Because of low traffic of internet at night on weekend, the graph is less unstable compared to the results in Figure 6- 53 at Bangkok. However, their variances and averages are still higher when more knowledge nodes are involved.

The variances of results of client at Chiang Mai are 1130, 3183, 2206, and 2498 ms, for the tests on one, two-near-proximity, two-far-proximity, and three knowledge nodes sequentially whereas their averages are 4147, 6196, 5857, and 6825 ms. sequentially which affirms the effect of the number of knowledge nodes to the average, but not to the variance.

The last set of results on the global simulation is in Tokyo, Japan where the test was run in the business area using high speed internet. However, the business of the client barely involves the access to internet. So the client ran the tests on the high availability of internet access. Client ran the tests on PC Pentium IV 2.8 GHz processor with 512 Mbytes of memory. The results of global simulation of the client running at Tokyo are shown in Figure 6- 55.

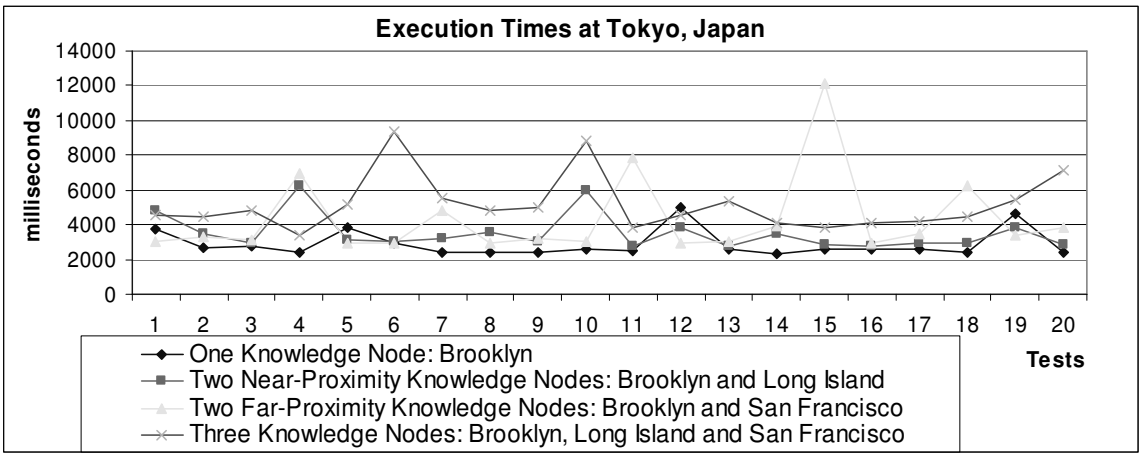


Figure 6- 55 Execution Times at Tokyo, Japan of Global Simulation

Even though the graphs in Figure 6- 55 are not quite stable but, with its lower scale on Y-axis compared to other nodes of global simulation, they are not unstable graphs. Only two to four points on the graphs of each number of knowledge nodes swing away from their averages.

The variances of results of client at Tokyo are 687, 1022, 2341, and 1573 ms, for the tests on one, two-near-proximity, two-far-proximity, and three knowledge nodes sequentially whereas their averages are 2887, 3520, 4306, and 5144 ms. sequentially which affirms the effect of the number of knowledge nodes to the average, but not to the variance.

The following graph in Figure 6- 56 shows the comparison of average execution time of all clients in national simulation together categorized by the number of knowledge nodes.

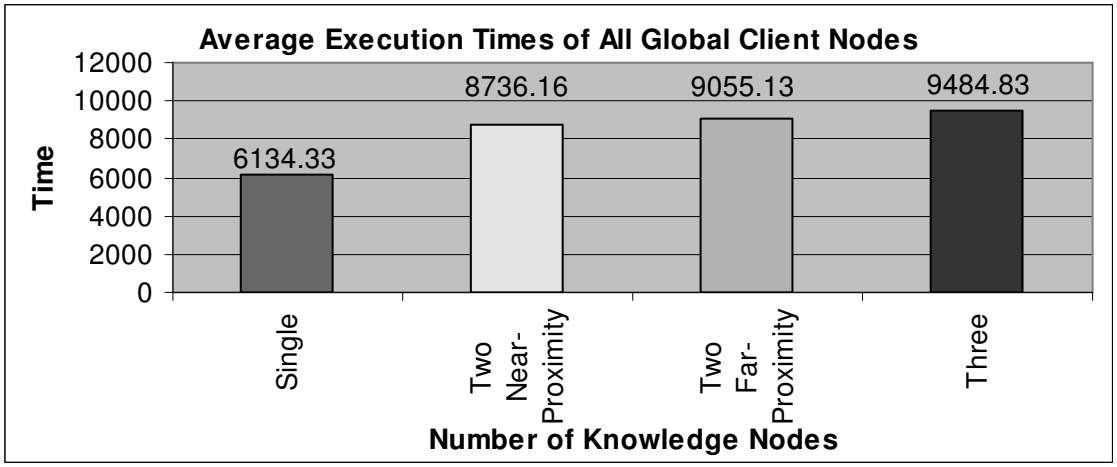


Figure 6- 56 Average Execution Times of all Clients for each Number of Knowledge Nodes at Global Simulation

The mean between the average execution times of two near-proximity and of two far-proximity knowledge nodes from Figure 6- 56 is (8736.16 + 9055.13) / 2 = 8895.645 ms. But the mean between the average of execution time of one and of three knowledge nodes is (6134.33+9484.83)/2 = 7809.58ms., which is much different from the mean of two types of two knowledge nodes .

Using the means, we can find the average execution time when the number of knowledge is increased:

7809.58 – 6134.33 = 1675.25

from one knowledge node –to- the mean between one and three knowledge nodes

9484.83 – 7809.58 = 1675.25

from the mean between one and three knowledge nodes –to- three knowledge nodes

8895.645 – 6134.33 = 2761.315

from one knowledge node –to- the mean of two knowledge nodes

9484.83 – 8895.645 = 589.185

from the mean of two knowledge nodes to three knowledge nodes

The average is (1675.25 + 1675.25 + 2761.315 + 589.185)/4 = 1675.25

That accumulates approximately 1675.25 milliseconds of execution time when increasing the number of knowledge nodes by one (from one node to two nodes and from two nodes to three nodes.) on the global execution.

If 1675.25 milliseconds is the difference of execution times between n and n+1 knowledge nodes and the average of execution time of the single node is 6134.33 milliseconds, then 6134.33 – 1675.25 = 4459.08 ms. that is the average set up time of global simulation when **not** considering the execution time at a knowledge node.

6.5.3. Comparison among Client Nodes on Global Simulation

In this section, we show the comparison among client nodes in global simulation by considering separately each number of knowledge nodes.

The next graph in Figure 6- 57 shows the comparison among different nodes in the global simulation on the execution of one knowledge node.

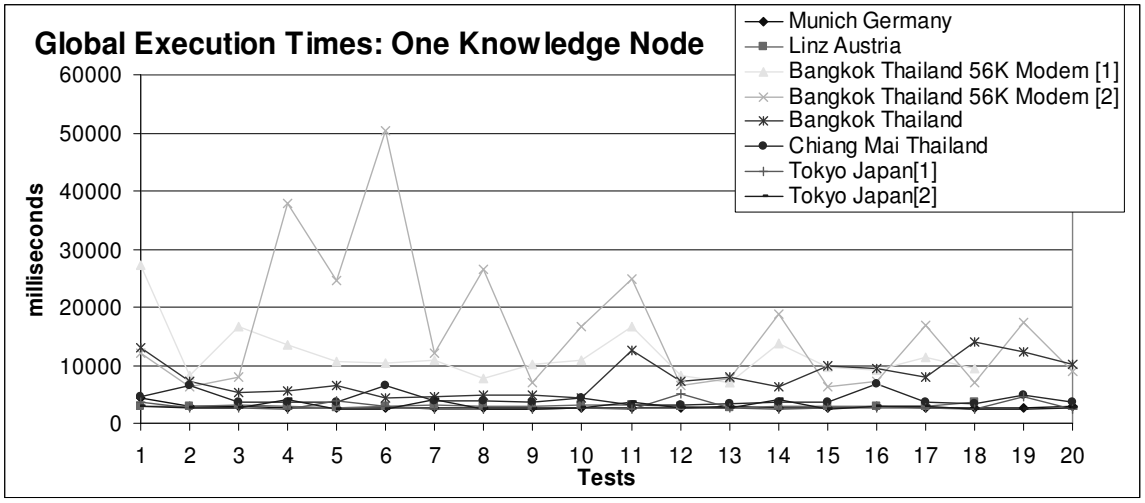


Figure 6- 57 Comparison of Tests among Nodes at Global Simulation on One Knowledge Node

The graphs in Figure 6- 57 shows similar results to the graphs of regional, and national simulations on one knowledge node that two tests on low speed 56K-modem swing the graph on their variances comparing to other nodes of global simulation. Not only their variances are big, but also their averages are tremendous as shown in Figure 6-58.

The average results of the tests in Europe and Japan are as low as some averages in the regional and national networks because of the high speed connection at the client nodes of global simulation.

The average from the tests at Chiang Mai is not as high as in Bangkok due very low traffic during the time of execution but is still higher than of other nodes in Europe and Japan on one medical knowledge node. However, when more knowledge nodes are added, the results of Chiang Mai show in a different direction.

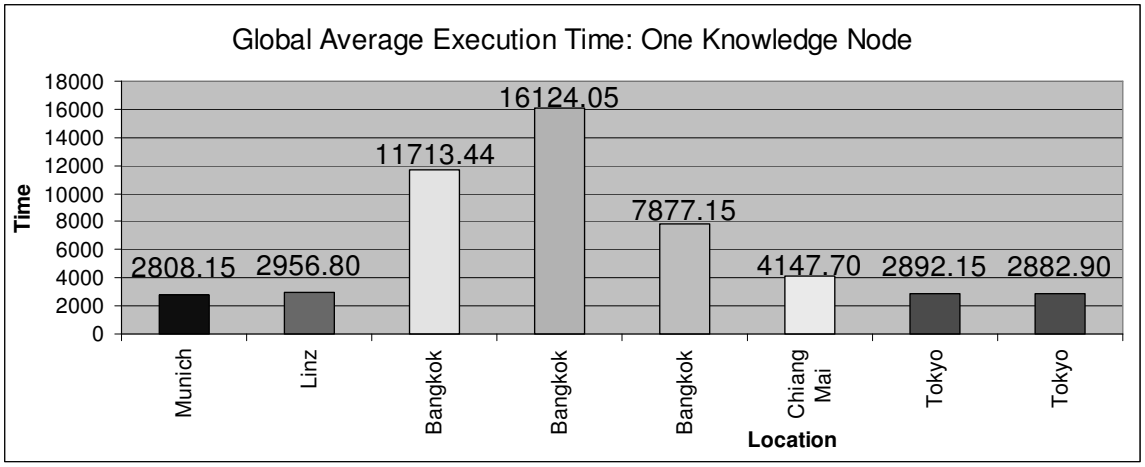


Figure 6- 58 Average Execution Times among Client Nodes at Global Simulation on One Medical Knowledge Node

The following Table 6- 13 shows the summary of simulation on global network where client nodes ran on one medical knowledge nodes.

Location	City	Munich	Linz	Bangkok	Bangkok	Bangkok	Chiang Mai	Tokyo	Tokyo
	Country	Germany	Austria	Thailand	Thailand	Thailand	Thailand	Japan	Japan
	Type	Business	Education	Home	Resident	Business	Resident	Business	Business
Execution Time	Average	2808.15	2956.80	11713.44	16124.05	7877.15	4147.70	2892.15	2882.90
	Max	4329.00	3626.00	27139.00	50442.00	13962.00	6656.00	4968.00	4188.00
	Min	2578.00	2624.00	6889.00	6199.00	4236.00	3140.00	2344.00	2375.00
	Variance	370.78	217.17	4728.61	11845.91	3148.73	1130.67	774.01	600.64
Distance from Client to Knowledge Node at Brooklyn (miles)		4042	4139	8652	8649	8649	8310	6760	6760
Client Connection	Speed (kbps)	34,000	2,000	56	56	10,000	512	30,000	30,000
	Media Type	E3	Cable Modem	56K Modem	56K Modem	Ethernet	ADSL	E3	E3
Client Computer	Memory (Mbyte)	384	768	768	256	256	512	512	512
	CPU (GHz)	Pentium M 1.1	Pentium M 1.3	Celeron 2.8	Pentium M 1.2	Pentium IV 1.8	Pentium IV 1.8	Pentium IV 2.8	Pentium IV 2.8
Execution Hour	Client	2PM-3PM	3PM-4PM	8AM-11AM	7AM-9PM	10AM-12PM	10PM-1AM	12PM-1PM	12PM-1PM
	Brooklyn Node	8AM-9AM	9AM-10AM	8PM-11PM	7PM-9AM	10PM-12AM	10AM-1PM	10PM-11PM	10PM-11PM
	Date	Tue 12/13/05	Sun 01/22/06	Wed 01/19/05	Wed 02/15/06	Mon 12/12/05	Sat 02/11/06	Wed 01/25/06	Wed 01/25/06

Table 6- 13 Summary of Global Simulation on One Medical Knowledge Nodes

The comparison among different nodes in the global simulation when the execution ran on two knowledge nodes is shown in Figure 6- 59.

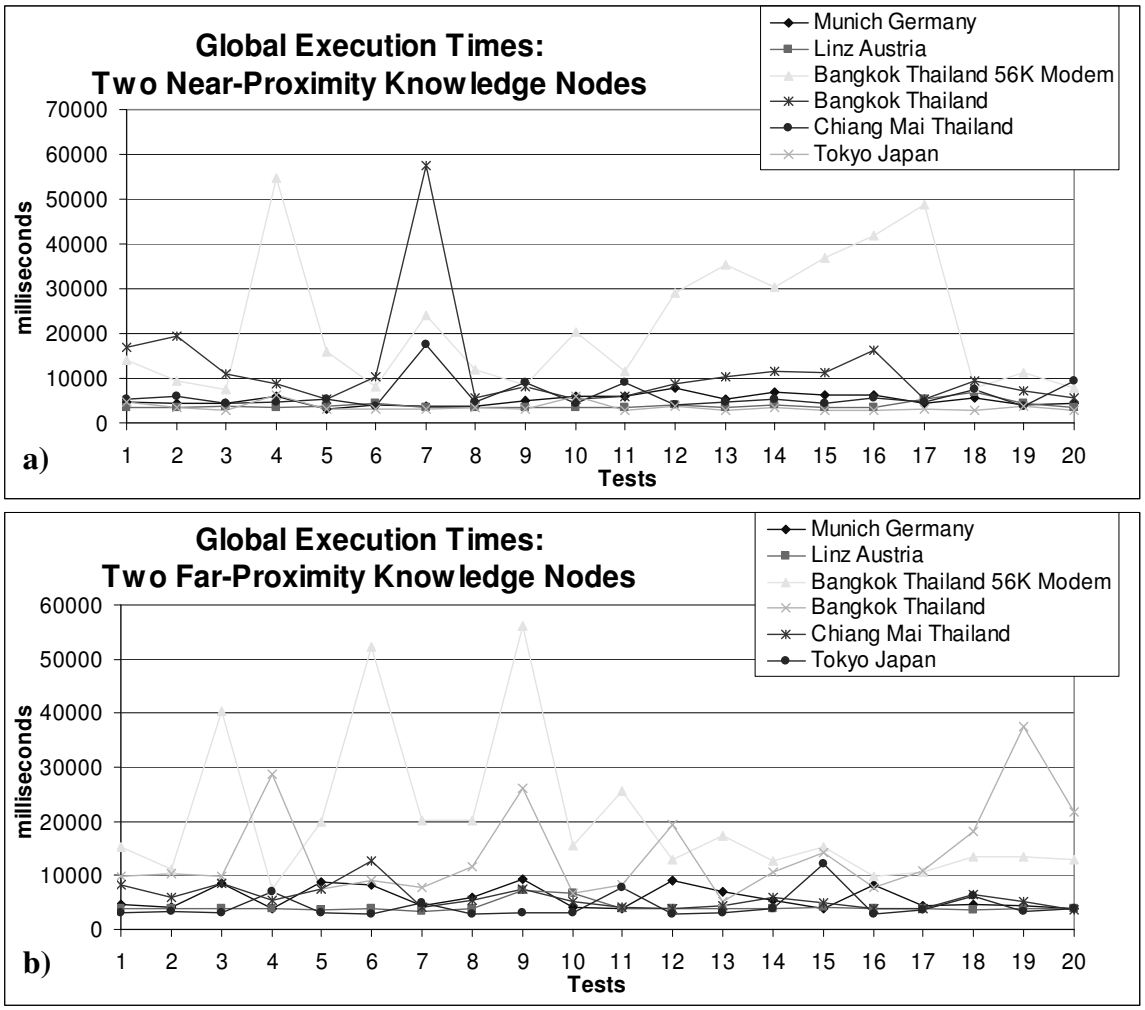


Figure 6- 59 Comparison of Tests among Nodes at Global Simulation on Two Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

The result from Bangkok using high-speed of two far-proximity nodes show a wide variance, 8569 ms. compared to the tests of two near-proximity nodes whose variance is 4178 ms if not including its seventh result which is unusual compared to other results in the same category. Also the far knowledge is at San Francisco which is closer to Bangkok than the near knowledge node at Long Island, NY. This double variance is also shown in the results from Tokyo whose variance is 2341 ms. in the tests of two far-

proximity nodes and only 1022 ms. in the tests of two near-proximity nodes. Both tests were run during the day on week but not like in Chiang Mai which was run during weekend at night of client time.

The following Figure 6- 60 shows the comparison of average times among global clients running on two knowledge nodes.

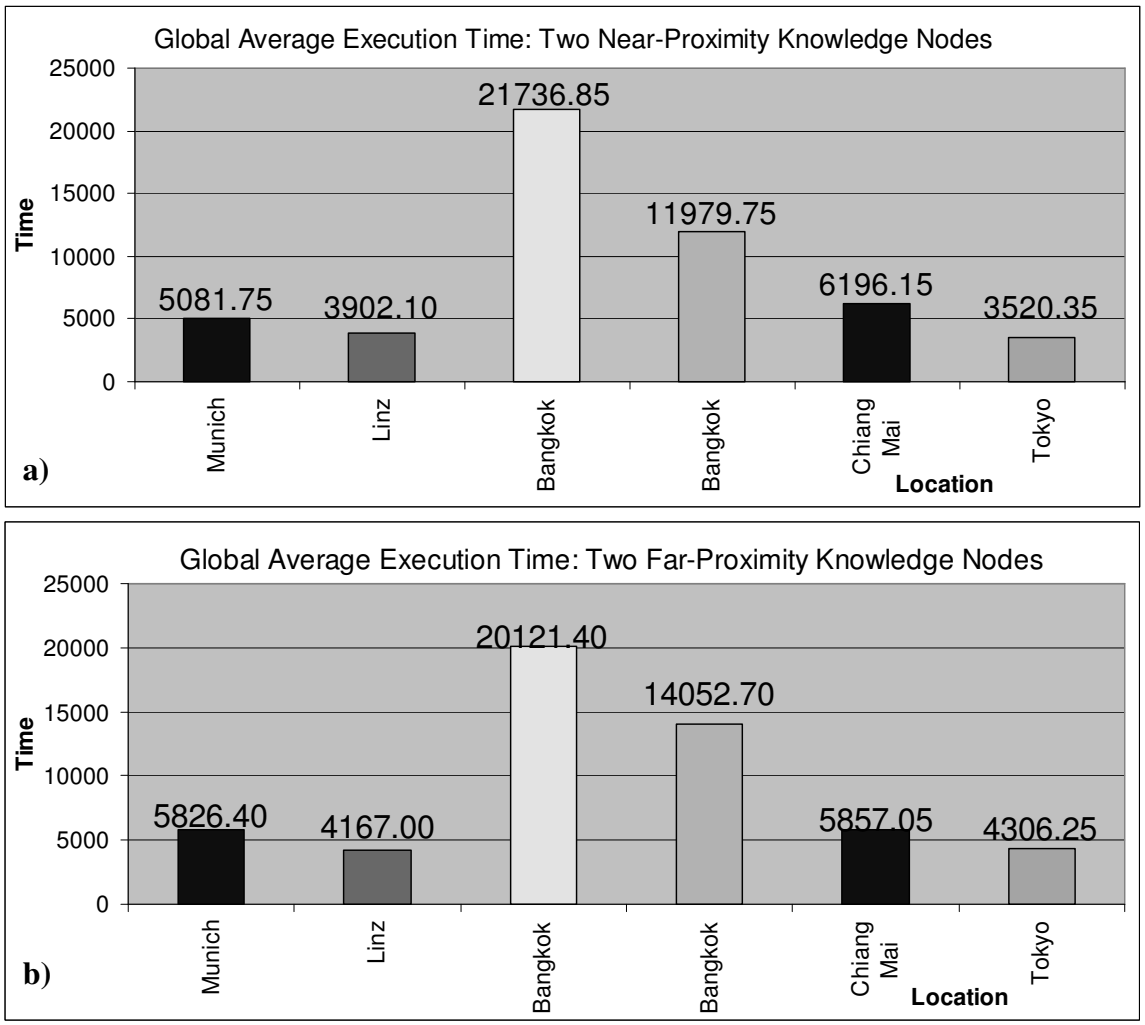


Figure 6- 60 Average Execution Times among Client Nodes at Global Simulation on Two Medical Knowledge Nodes a) Shows Near-Proximity and b) Shows Far-Proximity

Not only variance but the average times of most of client nodes running on two far-proximity nodes in Figure 6- 60 are higher than of their two near-proximity nodes,

except Chiang Mai which ran during low traffic on night on weekend. This observation is more outstanding in the case of global simulation than other cases.

The difference of height of graphs among global clients is in similar order of tests on one knowledge node except the results in Munich which is affected by the high volume of network traffic of its large business type compared to educational type of Linz and light business of Tokyo.

The following two tables; Table 6- 14 and Table 6- 15, show the summary of simulation on global network where client nodes ran on two medical knowledge nodes. Table 6- 14 shows the result when two knowledge node are close to another whereas the results of two knowledge nodes that are far away from another show in Table 6- 15.

Client Location	City	Munich	Linz	Bangkok	Bangkok	Chiang Mai	Tokyo
	Country	Germany	Austria	Thailand	Thailand	Thailand	Japan
	Type	Business	Education	Resident	Business	Resident	Business
Execution Time	Average	5081.75	3902.10	21736.85	11979.75	6196.15	3520.35
	Max	7719.00	7020.00	54548.00	57355.00	17516.00	6266.00
	Min	3141.00	3325.00	7551.00	5283.00	3844.00	2766.00
	Variance	1201.92	878.40	14992.04	11428.34	3183.88	1022.39
Distance from Knowledge Node to Client (miles)	Brooklyn Node	4042	4139	8649	8649	8310	6760
	Islandia Long Island Node	4003	4101	8644	8644	8298	6761
Client Connection	Speed (kbps)	34,000	2,000	56	10,000	512	30,000
	Media Type	E3	Cable Modem	56K Modem	Ethernet	ADSL	E3
Client Computer	Memory (Mbyte)	384	768	256	256	512	512
	CPU (GHz)	Pentium M 1.1	Pentium M 1.3	Pentium M 1.2	Pentium IV 1.8	Pentium IV 1.8	Pentium IV 2.8
Execution Hour	Client	2 PM - 4 PM	2 PM - 5 PM	7 AM - 10 PM	4 PM - 8 PM	10 PM - 1 AM	3 PM - 1 PM
	Brooklyn Node	8 AM - 10 AM	8 AM - 11 AM	7 PM - 10 AM	4 AM - 8 AM	10 AM - 1 PM	1 AM - 11 PM
	Long Island Node	8 AM - 10 AM	8 AM - 11 AM	7 PM - 10 AM	4 AM - 8 AM	10 AM - 1 PM	1 AM - 11 PM
	Date	Tue/12/13/05	Sun/01/22/06	Wed/02/15/06	Wed/02/15/06	Sat/02/11/06	Wed/01/25/06

Table 6- 14 Summary of Global Simulation on Two Near-Proximity Medical Knowledge Nodes

Client Location	City	Munich	Linz	Bangkok	Bangkok	Chiang Mai	Tokyo
	Country	Germany	Austria	Thailand	Thailand	Thailand	Japan
	Type	Business	Education	Resident	Business	Resident	Business
Execution Time	Average	5826.40	4167.00	20121.40	14052.70	5857.05	4306.25
	Max	9297.00	7210.00	56141.00	37494.00	12672.00	12110.00
	Min	3828.00	3465.00	7400.00	5203.00	3719.00	2906.00
	Variance	2071.87	956.33	13613.62	8569.29	2206.23	2341.64
Distance from Knowledge Node to Client (miles)	Brooklyn Node	4042	4139	8649	8649	8310	6760
	San Francisco Node	5872	5930	7915	7915	7702	5142
Client Connection	Speed (kbps)	34,000	2,000	56	10,000	512	30,000
	Media Type	E3	Cable Modem	56K Modem	Ethernet	ADSL	E3
Client Computer	Memory (Mbyte)	384	768	256	512	512	512
	CPU (GHz)	Pentium M 1.1	Pentium M 1.3	Pentium M 1.2	Pentium IV 2.6	Pentium IV 1.8	Pentium IV 2.8
Execution Hour	Client	5 PM - 6 PM	8 PM - 9 PM	7 AM - 10 PM	8 AM - 2 PM	10 PM - 1 AM	3 PM - 4 PM
	Brooklyn Node	11 AM - 12 PM	2 PM - 3 PM	7 PM - 10 AM	8 PM - 2 AM	10 AM - 1 PM	1 AM - 2 AM
	San Francisco Node	8 AM - 9 AM	11 AM - 12 PM	4 PM - 7 AM	5 PM - 11 PM	7 AM - 10 AM	10 PM - 11 PM
	Date	Fri/02/24/06	Mon/01/23/06	Wed/02/15/06	Mon/01/23/06	Sat/02/11/06	Thu/01/26/06

Table 6- 15 Summary of Global Simulation on Two Far-Proximity Medical Knowledge Nodes

The last comparison among different nodes when execution is on the global network is shown in Figure 6- 61.

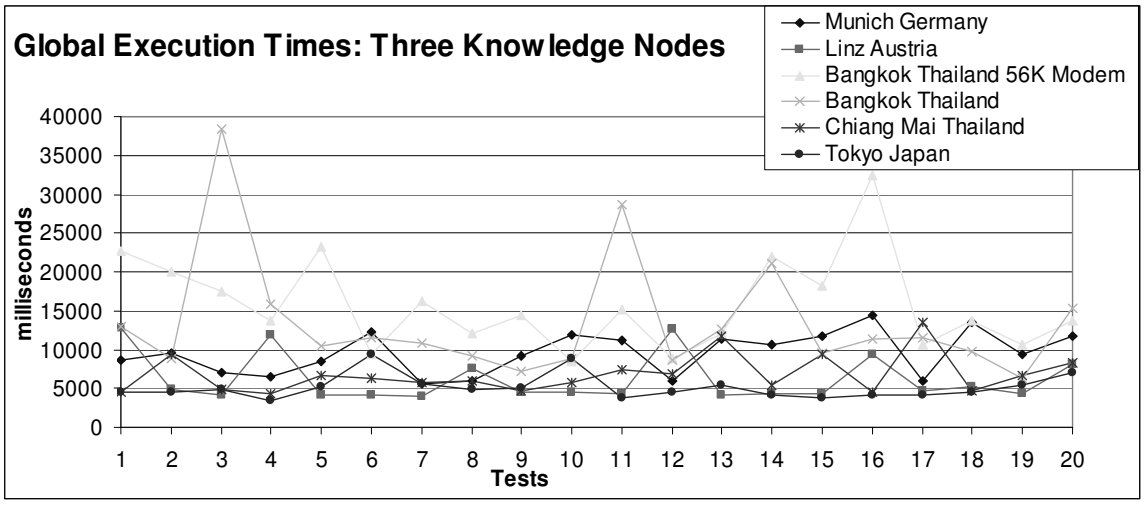


Figure 6- 61 Comparison of Tests among Nodes at Global Simulation on Three Knowledge Nodes

Because it is global and it included the most knowledge nodes of our simulation, the chart show wide variances and large averages on all client nodes including of Linz and Tokyo which were much stable in any previous charts. The widest variance is 7780 ms. at Bangkok with high speed internet but not at Bangkok with 56K-modem whose variance is 6012. However, both variances are very high compared to any other global client nodes. Whereas, Tokyo still shows the lowest variances, 1573 ms.

Figure 6- 62 shows the averages when running on three medical knowledge nodes by the global clients.

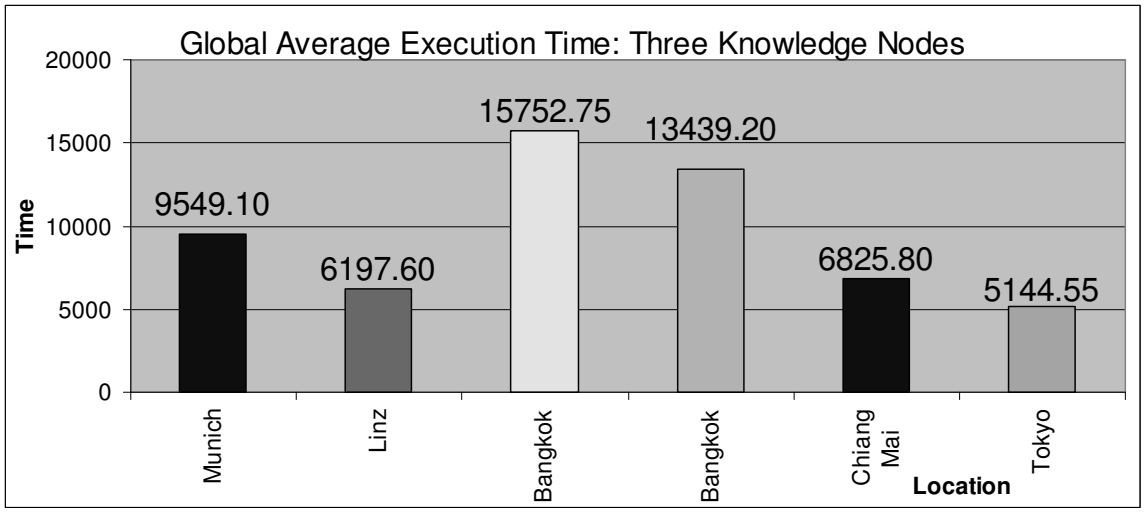


Figure 6- 62 Average Execution Times among Client Nodes at Global Simulation on Three Medical Knowledge Nodes

The same order of the heights of graphs among the global clients is shown in Figure 6- 62 running three knowledge nodes and in Figure 6- 60 running two knowledge nodes. Therefore regardless of numbers of multiple knowledge nodes, the location and the internet speed (including internet connection from its provider, type of client, and hour of execution) of the clients determine the average execution time of distributed medical knowledge nodes for global simulations.

The following Table 6- 16 shows the summary of simulation on global network where client nodes ran on three medical knowledge nodes.

Client Location	City	Munich	Linz	Bangkok	Bangkok	Chiang Mai	Tokyo
	Country	Germany	Austria	Thailand	Thailand	Thailand	Japan
	Type	Business	Education	Resident	Business	Resident	Business
Execution Time	Average	9549.10	6197.60	15752.75	13439.20	6825.80	5144.55
	Max	14423.00	12779.00	32487.00	38375.00	13453.00	9375.00
	Min	5672.00	3986.00	8463.00	6359.00	4312.00	3344.00
	Variance	2720.55	3061.40	6012.63	7780.05	2498.77	1573.89
Distance from Knowledge Node to Client (miles)	Brooklyn Node	4042	4139	8649	8649	8310	6760
	Long Island Node	4003	4101	8644	8644	8298	6761
	San Francisco Node	5872	5930	7915	7915	7702	5142
Client Connection	Speed (kbps)	34,000	2,000	56	10,000	512	30,000
	Media Type	E3	Cable Modem	56K Modem	Ethernet	ADSL	E3
Client Computer	Memory (Mbyte)	384	768	256	512	512	512
	CPU (GHz)	Pentium M 1.1	Pentium M 1.3	Pentium M 1.2	Pentium IV 2.6	Pentium IV 1.8	Pentium IV 2.8
Execution Hour	Client	5 PM - 6 PM	8 PM - 9 PM	8 AM - 9 PM	11 AM - 2 PM	10 PM - 1 AM	3 PM - 4 PM
	Brooklyn Node	11 AM - 12 PM	2 PM - 3 PM	8 PM - 9 AM	11 PM - 2 AM	10 AM - 1 PM	1 AM - 2 AM
	Long Island Node	11 AM - 12 PM	2 PM - 3 PM	8 PM - 9 AM	11 PM - 2 AM	10 AM - 1 PM	1 AM - 2 AM
	San Francisco Node	8 AM - 9 AM	11 AM - 12 PM	5 PM - 6 AM	8 PM - 11 PM	7 AM - 10 AM	10 PM - 11 PM
	Date	Fri/02/24/06	Mon/01/23/06	Wed/02/15/06	Mon/01/23/06	Sat/02/11/06	Thu/01/26/06

Table 6- 16 Summary of Global Simulation on Three Medical Knowledge Nodes

6.6.Parallel Simulation at Main Knowledge Node

In the real world situation, many clients connect to medical knowledge node(s) on the same time for accessing information or being given medical analysis. The database server may be composed of dual, or multiple processors and runs on parallel or distributed operating system with shared memory which accelerates the execution time when many clients concurrently access to the server(s) of medical knowledge node(s). However there are, still, servers using only one processor and runs on non-distributed operating system.

In this additional section, we show the results of parallel execution on the main medical knowledge node at Brooklyn site. Our parallelism tests on multiple threads on a single process. We created four programs which contain one, two, three, and four threads. There is no parallel execution in the program of one thread. However, it is used for the comparison.

In the program containing multiple threads, all threads are created and launched on the same time when the program is started. Therefore all threads will compete for CPU time in order to execute its program. All threads run the same program which contains six steps of simulation of medical deductive analysis:

- Test Preparation
- Patient Given Symptoms
- Mark Symptoms
- Mark Causes
- Disease Conclusion
- Result Collection

Six steps of simulation of medical deductive analysis are explained in section 6.1.2 on page 213.

The order of execution among threads may not be the same because the operating system may choose any thread to run, continue or sleep (on hold while waiting for other needs such as I/O.) Therefore, a thread that starts first may finish last or any order.

We ran the tests about 50 times on SUN UltraSPARC-IIe, sparcv9 processor operating at 512 MHz with 384 Mbytes of memory on UNIX operating system. Their results are shown in Figure 6- 63

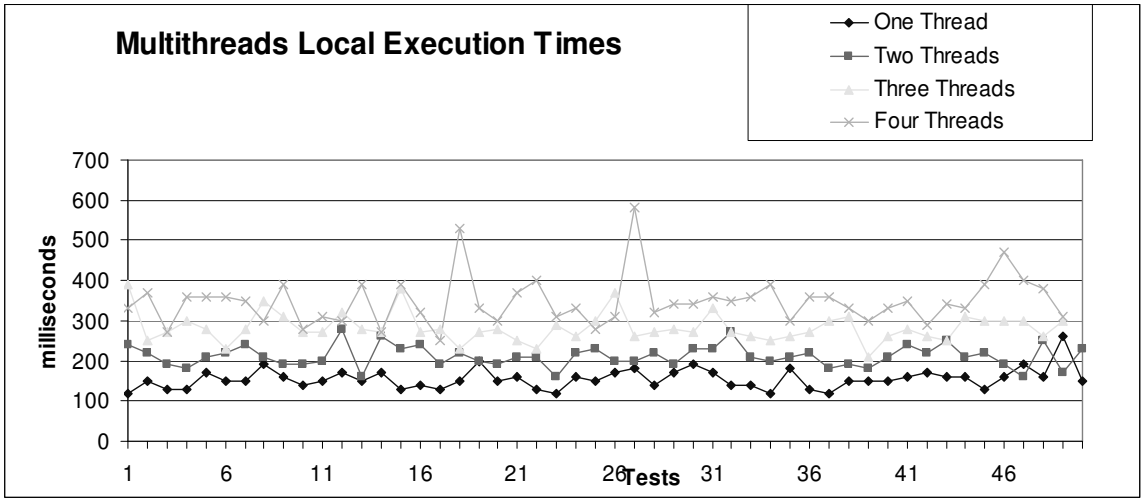


Figure 6- 63 Parallel Comparison among Tests on Multiple Threads

The results are shown in the concurrent manner; more parallelism results in longer execution time and wider fluctuation. Because the tests were run on local computer inside the same LAN of the knowledge node, there is no affect from other internet factors. Therefore, the execution times are very close among the tests on the same number of threads. The following Figure 6- 64 shows the average times of parallel execution among different number of threads.

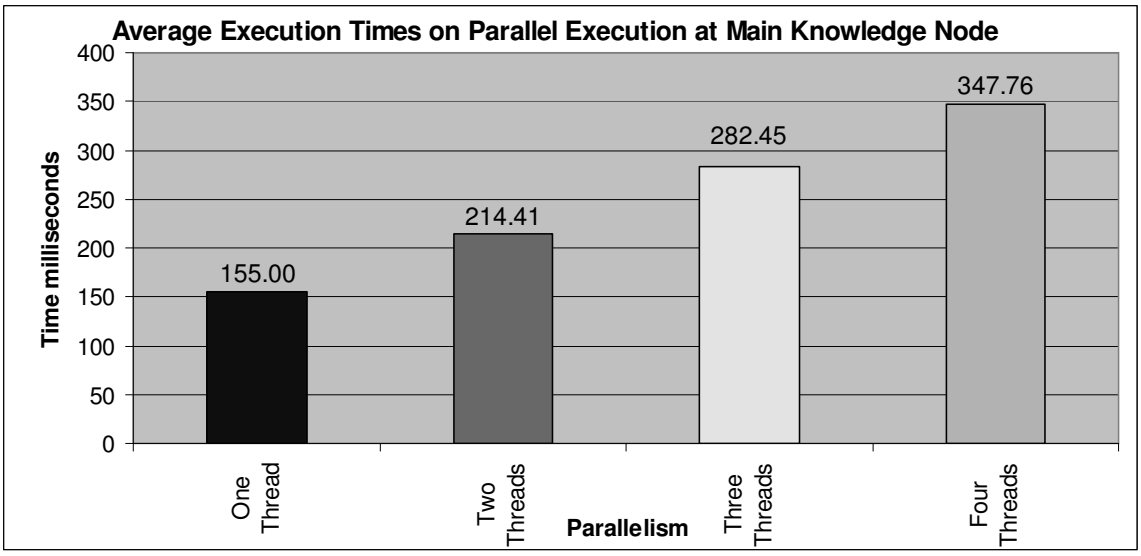


Figure 6- 64 Average Execution Times among Threads on Parallel Execution at the Main Knowledge Node at Brooklyn Site

Because there is no effect from the internet on these tests, the chart shows a steady linear increasing. The difference of average execution time on the number of threads from n to n+1 threads is about 65 milliseconds. Therefore, it is about 90 milliseconds for the fixed execution time plus 65 ms for each thread.

Execution time = 90 + 65n

where n is the number of threads running parallel.

6.7. Summary and Highlights of Simulation

When our geography of simulation is larger, obviously the execution time is bigger too. However it is not the linear factor to the distance as shown in Figure 6- 65. Therefore the performance of the tests on multiple medical knowledge nodes is better when the distance of client is longer from the knowledge nodes. This is one of the major benefits of this dissertation that is to provide help to distant developing countries.

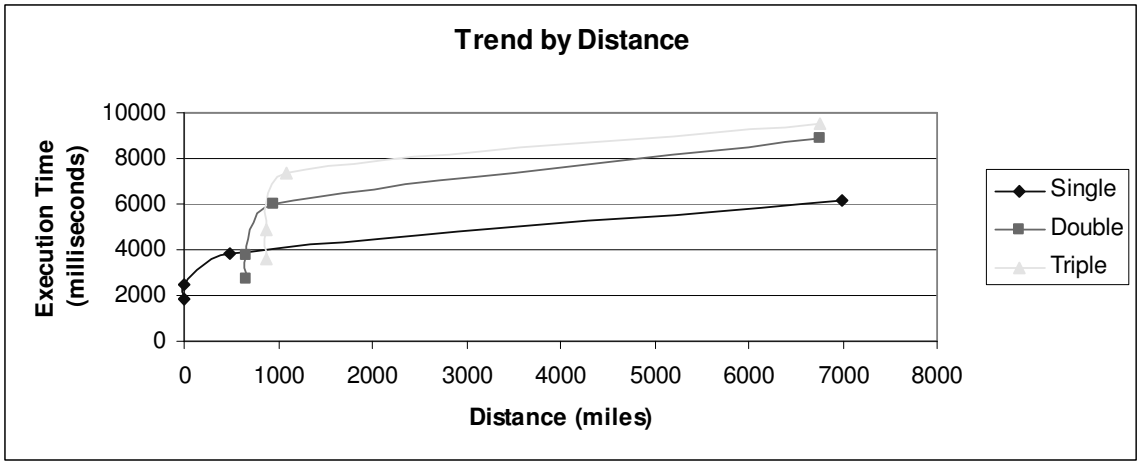


Figure 6- 65 Comparison between Average Execution Time and Average Distance from Clients to Knowledge Nodes among Number of Knowledge Nodes

The tests in this dissertation also found that the ratio between execution time of medical analysis and number of medical knowledge nodes is decreasing when the number medical knowledge nodes increases as shown in Figure 6- 66. Therefore if there are more medical knowledge nodes to collaborate in medical deductive distributed analysis, the execution time will be less and less affect from the added medical knowledge nodes. Combining this conclusion with the geography of execution, the ratio is much better in case of bigger area of execution as shown in Table 6- 17.

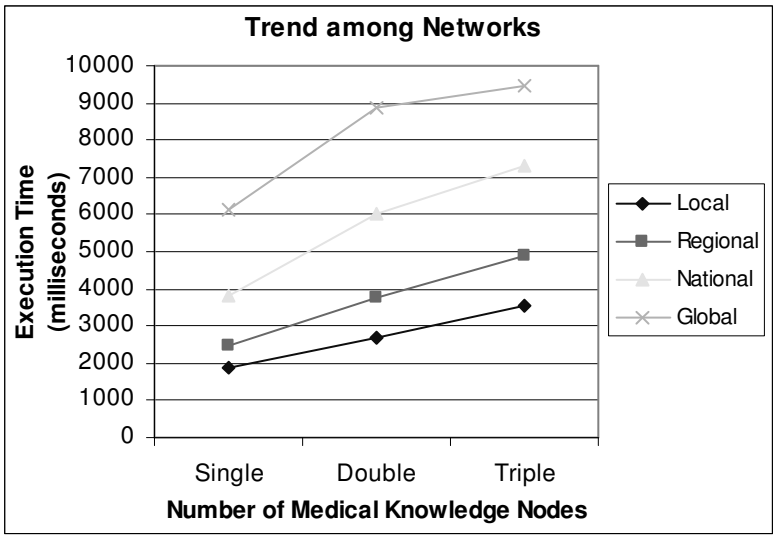


Figure 6- 66 Trend of Execution Time when Increasing the Number of Knowledge Nodes

Table 6- 17 demonstrates the trend of execution time of Figure 6- 66 in term of changing percentage from single to double knowledge nodes and from double to triple knowledge nodes.

Area	Changing Percentage from Single to Double	Changing Percentage from Double to Triple
Local	8.43	8.70
Regional	12.85	11.38
National	21.68	13.30
Global	27.61	5.89

Table 6- 17 Changing Percentage when Increasing the Number of Knowledge Nodes from n to n+1 Knowledge Nodes.

Table 6- 17 also shows the increasing rate in case of local execution where is inside the same institute maintaining the medical knowledge nodes. However, as for the usage of internet, most of clients, who connecting to the medical knowledge nodes, are not from local area. Therefore this value will not affect the execution of client to knowledge node but only to compare with situation of regional, national, and global network which are the real circumstances.

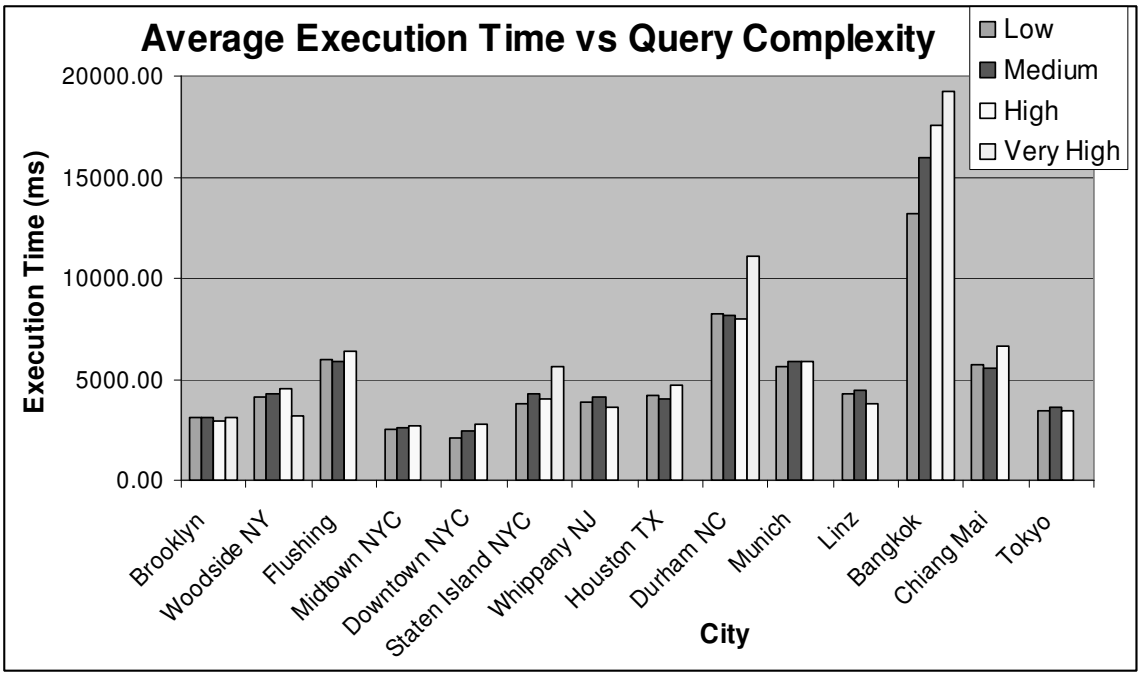


Figure 6- 67 Comparison between Average Execution Times and Four Levels of Query Complexities for Each Location of Simulations

Another comparison of our results is between the average execution time and the complexity of query. Query complexities are defined into four levels; low, medium, high, and very high, according to the complex of the criteria of a query. Because the tests are based on the random values, it is only a small chance of having very high complexity to a query as shown in the forth bars of each city in Figure 6- 67.

Figure 6- 67 shows that the complexities of queries have some effects on the execution times. When queries involve more sets of knowledge, the knowledge machines execute longer. This is shown in the results of cities; Flushing, Midtown, Downtown, Munich, and Bangkok. While the average execution times of other cities are also increased when the complexities of their queries are increased but not to all change of complexities. Therefore, the distance among knowledge nodes and client locations, and the number of knowledge nodes play more significant roles than the complexity of queries.

Chapter 7 Conclusion

Deductive Medical Analysis on Distributed Databases

The deductive distributed system for intelligent medical network improves the medical system in following ways.

- Increasing the dependability of medical experts and expert systems by providing a *confidence level for every phase or every decision point in the medical analysis and for the overall procedure*.
- Expanding the medical analysis capability of medical experts by providing using multiple medical knowledge nodes locally, around the region, around the nation and around the globe. *Medical experts can gather the most recent findings and thus provide the most suitable and up to date care to a patient*.
- Automating the medical analysis over the Internet. Physicians and patients (agents) will benefit from using the *Medical Knowledge Technology (MKT) system, suggested in the dissertation*, over the Internet.
- Expanding medical expertise to serve the needs of developing and undeveloped countries where *improvements in healthcare are urgently needed*.
- *Expedited Transfer of technology from the developed countries to developing and undeveloped countries*.
- *Clarifying the established symptoms, diagnosis, and the treatment* of ailments when medical experts are uncertain or in disagreement about the problem at hand.

- *Reducing the redundancy of medical research* that has already been done.
- *Reducing time and cost of distributing recently discovered medical knowledge* by providing unlimited access to medical knowledge banks.
- Finally, improving healthcare on a worldwide basis.

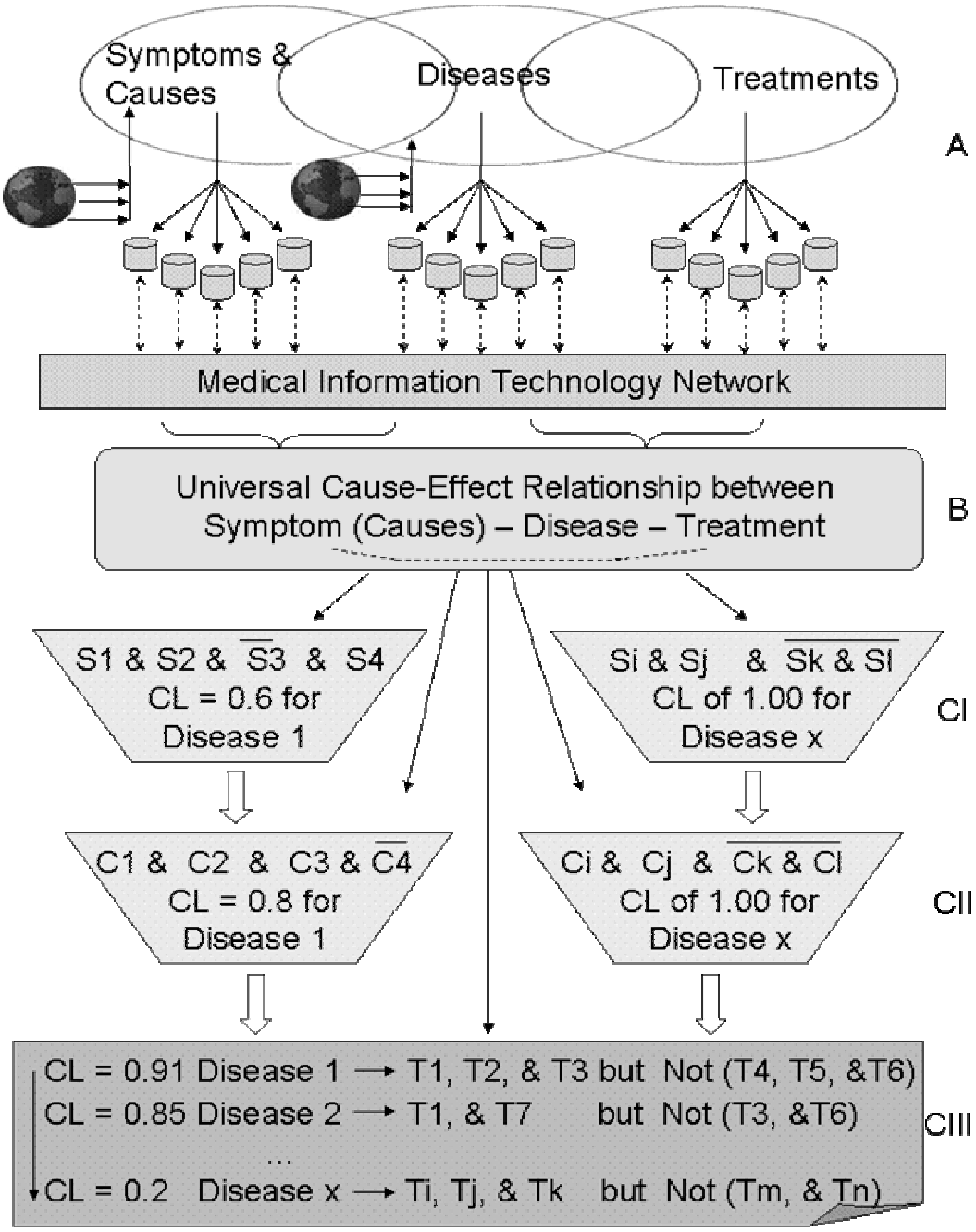


Figure 7- 1 Global picture of Medical Analysis using Medical Knowledge Technology (MKT) system based on Deductive Distributed Databases

We summarize the three key *components A, B, and C* in overall methodology of the MKT system shown in Figure 7- 1.

Component A encompasses the Medical Knowledge Bases (MKB) which is the most significant element of the system for the knowledge sharing and distribution among clients in MKT network. The world-wide knowledge was discovered by medical researchers, collected by computer scientists into the MKB, and connected via internet to anyone who has medical questions or problems.

Component B is to correlate the relationship among major characteristics of illnesses which include diseases to provide general information, symptoms that the patient may report, patient characteristics which might be associated with the cause, method of treatment such as-medicines, physical therapies, or life style changes-, the occurrences of diseases and symptoms that may be different according to the location, medicine and treatment providers such as pharmaceutical company, and health care providers such as outpatient clinics, hospitals and medical rehabilitation facilities.

The relationships among these characteristics of illnesses are well established in our system to provide the accuracy of medical analysis and treatment.

Component C has the procedures for calculating the Confidence Levels (CLs) during the medical analysis. The confidence level is likely (but does not have to) to increase along three levels, I, II, and III of medical analysis starting at:

Level I: Tracking the Symptoms (1) Symptoms reported by the patient, and (2) Verification of symptoms suggested by the MKT system and then queried to the patient. Using these two sets of symptoms, MKT provides the first and second levels of confidence to its user.

Level II: Analysis of the reasons for ailment to find activities, body contact or patients' history that may cause the symptoms the patient is reporting in Level I. Analysis of the reasons for ailment in MKT system produces list of likely causes of diseases based on analysis of symptoms and asks the patients for their confirmation. Using the symptoms and causes provided by the patients, the MKT system adjusts the confidence level into the next level for each disease.

Level III: Conclusion with a proposed treatment based on analysis of the likely diseases. Each disease may have several ways of treatment. C component at level III does not show only confidence level of each disease in the list of likely diseases, but also the confidence level of each proposed treatment of the diseases according to the inputs at levels I and II from the MKT users.

Continuing Research

The on-going of this research is adding more categories of medical knowledge to improve the illness analysis. The new categories are area of *likely occurrences of each disease, and medicine of its treatment*. A disease may occur in only some regions of a country or in many countries. The medicine of treatment might include information about pharmaceutical companies that are marketing its products or researching the treatment of an illness. However, this information must be obtained from companies that may involve business matters such as trade secrets.

After the areas of occurrence of diseases are obtained, *the adjustment of confidence level will be made to improve the confidence level to provide to the user of Medical Knowledge Technology*.

As of medical networking, the proposed further study will add more medical knowledge nodes to distribute in wider distances. With coordination of medical researchers, this medical network can grow world wide to all medical researchers, medical facilities, such as hospitals, clinics, and laboratories and patients who connecting to our medical network.

With more medical knowledge nodes and clients connecting to our medical network, we can further investigate the performance and optimization of the network when various factors are added to the study.

Performance of the medical network is proposed for future study when many more clients are connected to the MKT system. The factors of individual client can be added to the medical knowledge for further improvement of medical analysis. The combination of general medical knowledge currently existing in our database and new information records of patients is a topic of future study. It can provide a precise and individually conclusion and treatment to a specific patient.

Bibliography

Agichtein E., Lawrence S., Gravano L. *Learning Search Engine Specific Query Transformations for Question Answering* Proceedings of the 10th international conference on World Wide Web, Hong Kong, Pages 169 – 178, 2001

Ahamed, S. V., *Constructs of a Wisdom Machine (WM)* First Global Conference on Artificial Intelligence, at Mercure Nestroy Wien, Vienna, December 04-06, 2003, E book by Inter-Disciplinary Net: Critical Issues, May 2004

Ahamed, S. V. *Intelligent Internet Knowledge Networks* Wiley Publishers, 2006

Ahamed S. V., Lawrence V. B. *Design and Engineering of Intelligent Communication Systems*. Kluwer Academic Publishers, 1997

Ahamed, S. V., Lawrence, V. B. *Intelligent Broadband Multimedia Networks* Kluwer Academic Publishers, 1997

Ahamed, S. V., Lawrence, V. B. *Localized Knowledge Based Intelligent Medical Systems* Proceeding of 16th IEEE Symposium on Computer-Based Medical System, IEEE Computer Society 2003 pp. 89-96

Ahamed S. V., Lawrence V. B. *The Art of Scientific Innovation: Cases of Classical Creativity*. Pearson Prentice Hall, 2005

Balen R. M., Miller P., Malyuk D. L. *Medical Informatics: Pharmacists' Needs and Applications in Clinical Practice*, Journal of Informed Pharmacotherapy 2000

Bell G. B., Sethi A. *Matching Records in a National Medical Patient Index.* Communications of the ACM, Pages 83, Volume 44, Number 9, 2001

Cafarella M., Etzioni O. *A Search Engine for Natural Language Applications* Proceedings of the 14th international conference on World Wide Web Chiba, Japan, Pages: 442 – 452, 2005

Carlis J., Maguire J. *Mastering Data Modeling: A User-Driven Approach* Addison-Wesley 2001

Chin G. Jr., Lansing C. S., *Capturing and Supporting Contexts for Scientific Data Sharing via the Biological Sciences Collaboratory* Proceedings of the 2004 ACM conference on Computer supported cooperative work, Chicago, Illinois, USA, Pages: 409 – 418, November 2004

Cho J., Roy S. *Impact of Search Engines on Page Popularity* Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, Pages: 20 - 29 May 2004

Chu W.W., Merzbacher M. A., Berkovich L. *The Design and Implementation of CoBase.* ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.

Connolly T. M., Begg C. E. *DataBase Systems: A Practical Approach to Design, Implementation and Management.* 4th Edition, Edison-Wesley, 2005

Date C.J. *An Introduction to Database Systems.* 8th Edition, Addison-Wesley, 2004

Davis, A. R., *American Association for Medical Systems Informatics Records, History of Medicine* United States National Library of Medicine (NLM) http://www.nlm.nih.gov/hmd/manuscripts/ead/aamsi.html

Deitel H., Deitel P., Choffnes D. *Operating Systems.* 3rd Edition, Prentice-Hall, 2004

Eastman C. M., Jansen B. J. *Coverage, Relevance, and Ranking: The Impact of Query Operators on Web Search Engine Results* ACM Transactions on Information Systems, Volume 21, Number 4, Pages 383 – 411, October 2003

Easton, V. J. and McColl, J. H., *Confidence Intervals* Department of Statistics at University of Glasgow http://www.cas.lancs.ac.uk/glossary_v1.1/confint.html

Elmasri R., Navathe S. B., Sunderraman R. *Fundamentals of Database Systems/Oracle 9i Programming.* 4th Edition, Addison Wesley, 2004

Fagin R., Kolaitis P. G., Popa L., Tan W. C. *Composing Schema Mappings: Second-Order Dependencies to the Rescue.* ACM Transactions on Database Systems, Page 994, Volume 30, Number 4, December 2005

Gadia S. K. *The Seamless Generic Extension of SQL for Querying Temporal Data*, Computer Science Department, Iowa State University, 1992

Grimaldi R. P. *Discrete and Combinatorial Mathematics.* 5th Edition, Pearson Addison Wesley, 2004

Halle M. W., Kikinis R. *Flexible Frameworks for Medical Multimedia* Proceedings of the 12th annual ACM international conference on Multimedia, New York, NY, USA, Pages: 768 – 775, October 2004

Hernandez T., Kambhampati S. *Integration of Biological Sources: Current Systems and Challenges Ahead* ACM SIGMOD record, Volume 33, Issue 3, Pages: 51 – 60 September 2004

Kahanda G, Leung L., Waraporn N., Kazmi T., Mollah N.,Pinto M.,Yahya T, and Ahamed S. *Blending Natural Intelligence with Artificial Intelligence,* Proceedings of 14th International Multiconferences in Computer Science, Las Vegas, Nevada, USA, July 24-27, 2002

Klein A., Puig-Waldmüller E., Trost H. *Robust interpretation of user requests for text retrieval in a multimodal environment,* 19th international conference on Computational linguistics - Volume 2. 2002

Kokol P., *Some Ideas About Intelligent Medical System Design*, 12th IEEE Symposium on Computer-Based Medical Systems (CBMS '99)

Kulikowski C. A. *Artificial Intelligence Methods and Systems for Medical Consultation,* IEEE Transactions on Pattern and Machine Intelligence, September 1980

Lawrence V. B. *Integrated Medical Methods*, Patent filed by AT&T Bell Telephone Laboratories, November 2, 1993. Combination of medical micro functions with intelligent computer systems operating distributed databases and integrated communication facilities. European Patent Number 146248, US/02.11.93, Denmark, France, Great Britain, Italy. Issue date 29/12/94.

Lawrence V. B. *Knowledge Processing System Employing Confidence Levels*, US Patent # 5,809493, Issued on September 15, 1998 to Lucent Technologies, Murray Hill, NJ.

Lewis P. M., Bernstein A., Kifer M. *Databases and Transaction Processing: An Application-Oriented Approach.* Addison-Wesley, 2002

Mano M. M., Kime C. R. *Logic and Computer Design Fundamentals and Xilinx Student Edition*. 3rd Edition, Prentice Hall 2004

Miller RA, Pople HE Jr, Myers JD. *Internist-1, an experimental computer-based diagnostic consultant for general internal medicine*, The New England journal of medicine, Aug 19 1982; 307(8):468-76.

Ntoulas A., Cho J., Olston C. *What's New on the Web? The Evolution of the Web from a Search Engine Perspective* Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, Pages: 1 – 12, May 2004

Olson D. L., Courtney J. F. *Decision Support Models and Expert Systems.* Macmillan, 1992

Ousterhout, J. K., Cherenson, A. R., Douglis, F., Nelson M. N., Welch B. B. *IEEE Computer Society Technical Committee on Operating Systems Newsletter*, Vol. 3, No. 1, Winter 1989.

Ozsu M. T., Valduriez P. *Principles of Distributed Database Systems.* 2nd edition, Prentice Hall, 1999

Rajkumar D., *Operating Systems: A Systematic View*. 5th Edition, Addison-Wesley 2001

Ricardo, C. M. *Database Illuminated,* Jones and Bartlett Publishers, 2004

Riccardi G. *Principle of Database Systems with Internet and Java Applications* Addison-Wesley, 2001

Russell S., Norvig P. *Artificial Intelligence A Modern Approach*. Prentice Hall 1995

Shortcliffe E. *MYCIN: Computer-based medical consultations* American Elsevier 1976

Si L., Callan J., *A Semisupervised Learning Method to Merge Search Engine Results* ACM Transactions on Information Systems (TOIS), Volume 21, Issue 4, Pages 457 – 491, October 2003

Silberschatz A., Korth H. F., Sudarshan S. *Database System Concepts.* 4th Edition, McGraw-Hill, 2002

Singh A. K., Manjunath B. S., Murphy R. F. *A Distributed Database for Bio-Molecular Images* ACM SIGMOD record, Volume 33, Number 2, Page 65-71, June 2004

Snodgrass R., Norvig P. *The temporal Query Language TQuel*. ACM Transactions on Database Systems Volume 12, Issue 2, Pages: 247-298, 1987

Stallings W. *Operating Systems Internals and Design Principles*. 5th Edition, Pearson Prentice Hall, 2005

Sugumaran V., *Intelligent Support Systems Technology: Knowledge Management.* IRM Press, 2002

Tenenbaum, A. S., Sharp, G. J. *The Amoeba Distributed System* http://www.cs.vu.nl/pub/amoeba/Intro.pdf

Tanenbaum A. S., Van Steen M. *Distributed Systems Principles and Paradigms*. Prentice Hall, 2002

Tansel A. U., *Modeling temporal data*", Information and Software Technology, Vol. 32 No. 8, October 1990

Tansel A. U., *Temporal Relational Data Model*, IEEE Transactions on knowledge and data engineering Vol. 9 No. 3 May/June 1997

Tierney L. M., McPhee S. J., Papadakis M. A. *2002 Current Medical Dianosis & Treatment: Adult Ambulatory & Inpatient Management*. 41st Edition, McGraw-Hill, 2002

Tjora A. *Maintaining Redundancy in the Coordination of Medical Emergencies.* Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW 2004, Chicago, Illinois, USA, November 6-10, 2004

Tsai M. C., Melmon K. *Digital Library for Education and Medical Decision Making* Proceedings of the third ACM conference on Digital libraries; Pittsburgh, Pennsylvania, United States, Pages: 311 – 312, 1998

Udagawa M., Sato N., Uehara M., Sakai Y. *Distributed Pipelining Processing for Index Updating Method* Proceedings of 18th International Conference on Advanced Information Networking and Application (AINA'04), 2004

Vaina L. M. *Fuzzy set, usuality and commonsense reasoning, Matters of Intelligence*, D. Reidel Publishing Company, pp. 289-309, 1987

Vaina L. M. *Fuzzy Logic* Computer Vol. 1 no. 4 pp 83-93, 1988

Waraporn N., Ahamed S. V., *Intelligent Medical Search Engine by Knowledge Machine* Third International Conference on Information Technology: New Generations (ITNG 2006) of IEEE Computer Society, Las Vegas, Nevada, USA, April 10-12, 2006

White D., *Knowledge Mapping & Management.* IRM Press 2002

Wyss C. M., Robertson E. L. *Relational Languages for Metadata Integration.* ACM Transactions on Database Systems, Page 624, Volume 30, Number 2, June 2005

Zadeh L. A. *Fuzzy sets, fuzzy logic, and fuzzy systems,* World Scientific Publishing Co., Inc, pp. 738 – 758, 1996