Design, Implementation and Performance Analysis of a Distributed Enterprise Patient Location System for Regional Healthcare Data Networks

by

Lai Fong Ellen Cheung

A dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering
Faculty of Graduate Studies
University of Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

DESIGN, IMPLEMENTATION AND PERFORMANCE ANALYSIS OF A DISTRIBUTED ENTERPRISE PATIENT LOCATION SYSTEM FOR REGIONAL HEALTHCARE DATA NETWORKS

BY

LAI FONG ELLEN CHEUNG

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University

of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science

LAI FONG ELLEN CHEUNG© 2003

Abstract

Medical images are traditionally printed, stored, and retrieved using films. Due to the management process of film their utilization significantly affects healthcare delivery, so Picture Archiving and Communication Systems (PACSs) are introduced to replace film. Nowadays, PACS have become increasingly important in the hospital enterprise because of its many of advantages. Many hospitals employ a PACS or several PACSs to assist healthcare delivery. Although each of these PACSs worked effectively for each of the radiology departments and hospitals using them, patient medical data is not integrated with other PACSs residing at different departments of a hospital and/or remote hospitals. Patients often utilize the services of several hospitals so patient's medical images must be integrated from the independent PACSs when needed. Most importantly, patient data that is generated and stored at different PACSs must be made available in a timely and easily accessible way to be useful from a medical perspective.

In this thesis, a patient location system (PLS) is designed to maintain the meta-information of the location of patient's medical images. The PLS is meant to be used by the underlying distributed health system to integrate and access patient data from different PACSs in a timely fashion. The aim is to provide an efficient and high performance PLS. In this thesis, three PLS configurations are implemented with relational and hierarchical database models for the storage and management of the location information. We also implemented the PLS services using centralized and distributed approaches. Finally, the performance of the three PLS configurations is tested to identify the best PLS configuration. The experimental results show that among the three PLS configurations, the distributed PLS with relational directory model provides the best PLS configuration.

Acknowledgements

First, I would like to express my gratitude to St. Boniface General Hospital Research Centre (SBRC), University of Manitoba - Department of Radiology, and TR*Labs*, Winnipeg, for providing an excellent research environment to do the necessary work and financial support.

What I know today about the process of research, I learned from Dr. Jose A. Rueda, Director of TR*Labs*, Winnipeg, Dr. Ken Barker, the head of Computer Science department, University of Calgary, and Mr. Sergio Camorlinga, Chief Software Architect, St. Boniface General Hospital Research Centre (SBRC). The completion of this thesis is possible only because of their generous guidance and support.

I am deeply indebted to both of my advisors, Dr. Jose A. Rueda and Dr. Ken Barker, for their immense motivation, guidance, encouragement, and stimulating advice throughout the time it took me to achieve this work. Thanks to Dr. Rueda for encouraging me in my research and provided me with a lot of valuable advices throughout the research. Thanks are due to Dr. Barker for his constant encouragement and being there, especially I would never forget his rule of thumb - "Keep it simple stupid (KISS)" and all his instant email replies which provided me a lot of valuable comments on my work. I extend my sincere gratitude and appreciation to Mr. Sergio Camorlinga, whose support, constant encouragement, and insightful suggestion were vital for the research. I especially thanks to Sergio for the help extended to me when I approached him and the valuable discussion that I had with him during the course of research.

I would also like to thank my thesis examining committee, comprising Dr. Jose A. Rueda and Dr. Steve Onyshko from Electrical and Computer Engineering Department, University of Manitoba, Dr. Ken Barker from Computer Science Department, University of Calgary, and Alan F. Graves, the Director of Advanced Optical Network Products from Nortel Networks Advanced Technology Investments Group, for evaluating this thesis dissertation.

Special thanks go to Mrs. Alice Rueda and Jeff Diamond for their constructive comments provided on the preliminary version of this thesis. I also would like to thank all my friends and students at TR*Labs* for their help, discussion, support, and friendship.

I am grateful to my husband Nelson for the inspiration and moral support he provided throughout my research, his patience was tested to the utmost by a long period of separation, and his love enabled me to complete this work. Particularly, I would like to share this moment of happiness with my family and best friends, especially Maggie Lam, Susan Wong, Ken Cheng, and their families.

Finally, I would like to thank my God without Him nothing is possible.

Contents

1 Introduction **1**

1.1 Picture Archiving and Communication System 2

1.2 Health Distributed Archiving System 4

1.3 Motivations and Objectives 5

1.4 Location System 6

1.5 Summary of the Performance Analysis 9

1.6 Summary and Contributions of the Thesis 9

1.6.1 Contributions of the Thesis 10

1.7 Organization of the Thesis 10

2 Literature Survey **11**

2.1 Location Services 12

2.2 Middleware Technologies 18

2.2.1 Remote Procedure Call 18

2.2.2 Remote Method Invocations 20

2.2.3 Object Request Broker 21

3 Patient Location System **23**

3.1 System Architectures 24

3.1.1 Centralized PLS Architecture 25
3.1.2 Distributed PLS Architecture 28
3.2 Directory Models 30
3.2.1 Relational PLS Directory 32
3.2.2 Hierarchical PLS Directory 33

4 The Implementation of PLS Prototypes **36**
4.1 The PLS Components 37
4.1.1 The PLS Clients 37
4.1.1.1 Data Producers 39
4.1.1.2 Data Consumers 40
4.1.2 The PLS Servers 41
4.1.2.1 Node Servers 41
4.1.2.2 Directory Servers 42
4.2 Simulation Tool 42
4.2.1 Graphical User Interface 43
4.2.1.1 A Selection Frame 43
4.2.1.2 A Parameter Frame for Simulating Lookup Requests . . 44
4.2.1.3 A Parameter Frame for Simulating Update Requests . . 45
4.2.1.4 Master Scheduler 48
4.2.2 Node Schedulers 49

5 Performance Study **50**
5.1 Performance Metrices 50
5.1.1 Transaction Response Time 51
5.1.2 Throughput 54
5.2 Study Methodology 55

5.2.1 Simulation Types . . . 55
5.2.2 Hardware and Software Configurations . . . 56
5.2.3 Simulation Procedures . . . 58
5.2.4 Assumptions . . . 60
5.2.5 Simulation Parameters . . . 61
5.3 Experimental Results and Analysis . . . 62
5.3.1 A Performance Study on CPLS-R and CPLS-L Configurations . . 63
5.3.1.1 Interactions Involve Lookup-only Operations . . . 63
5.3.1.2 Interactions Involve Update-only Operations . . . 65
5.3.1.3 Interactions Involve Lookup and Update Operations . . 67
5.3.1.4 The Throughput of CPLS-R and CPLS-L . . . 70
5.3.1.5 Summary of CPLS-R and CPLS-L Performance Study . 73
5.3.2 A Performance Study on CPLS-R and DPLS-R Configurations . . 73
5.3.2.1 Interactions Involve Lookup-Only Operations . . . 74
5.3.2.2 Interactions Involve Update-Only Operations . . . 77
5.3.2.3 Interactions Involve Lookup and Update Operations . . 79
5.3.2.4 The Throughput of CPLS-R and DPLS-R . . . 82
5.3.2.5 Summary of CPLS-R and DPLS-R Performance Study . 84

6 Conclusions and Future Directions **85**
6.1 Ideas for Future Research . . . 88

References **91**

List of Tables

4.1 An example of patient distribution defined for simulation "X" 48

4.2 An example of studies distribution defined for simulation "X" 48

5.1 The definitions of transaction response time 53

5.2 Hardware/software configurations of processing nodes for PLS simulations 57

List of Figures

3.1 Centralized PLS located at hospital 2 of the HDAS node 26

3.2 The interaction between client, server, and directory server in PLS 27

3.3 Distributed PLS in HDAS 28

3.4 An Entity-Relationship diagram of DICOM database and centralized PLS directory 33

3.5 An organization of hierarchical PLS directory based on the DIT structure 34

4.1 A data producer process 39

4.2 A data consumer process 40

4.3 A selection frame for defining a simulation type 43

4.4 A parameter frame for setting up lookup request simulation 44

4.5 A parameter frame for setting up update request simulation 46

4.6 Patient distributions represented in a Venn diagram 47

4.7 A master scheduler process 49

5.1 The PLS transaction response times collected at different layers 54

5.2 A conception view of the simulation model when testing centralized PLS architecture. 60

5.3 The DSPTs of the CPLS-R and CPLS-L for handling "get-hosts-list" requests in LO simulations 64

5.4 The DSPTs of the CPLS-R and CPLS-L for handling update requests in UO simulations 65

5.5 The DSPTs of the CPLS-R and CPLS-L in LU simulations. (a) DSPTs for handling "get-hosts-list" lookup requests. (b) DSPTs for handling update requests. 68

5.6 The throughput of the CPLS-R and CPLS-L in LU simulations. (a) Number of data consumer routines per hour. (b) Number of data producer routines per hour. 71

5.7 The TRTs-Ns of the CPLS-R and DPLS-R for handling "get-hosts-list" lookup requests in LO simulations 74

5.8 The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling "get-hosts-list" lookup requests in LO simulations 75

5.9 The DSPT reported by each processing node of the DPLS-R and the DSPT of CPLS-R configurations for handling "get-hosts-list" lookup requests in LO simulations 76

5.10 The TRTs-Ns of the CPLS-R and DPLS-R for handling update requests in UO simulations 77

5.11 The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling update requests in UO simulations 78

5.12 The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling "get-hosts-list" lookup requests in LU simulations 80

5.13 The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling update requests in LU simulations 80

5.14 The throughput of the CPLS-R and DPLS-R in LU simulations. (a) Number of data consumer routines per hour. (b) Number of data producer routines per hour. 83

Acronyms

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
CPLS-L	Centralized PLS with a LDAP-based Directory
CPLS-R	Centralized PLS with a Relational Directory
CPR	Computerized Patient Record
CSCT	Client/Node Server Communication Time
CT	Communication Time
DBMS	Database Management System
DICOM	Digital Imaging and Communication In Medicine
DIMSE	DICOM Message Service Element
DIT	Directory Information Tree
DPLS-R	Distributed PLS with a Relational Directory
DSPT	Directory Server Processing Time
EHCR	Electronic Health Care Record
HDAS	Health Distributed Archiving System
HIS	Hospital Information System
HL7	Health Level 7
GUI	Graphical User Interface
IDL	Interface Definition Language
IOD	Information Object Definition
ISIS	Interactive System for Image Selection
JNDI	Java Naming and Directory Interface

JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LO	Lookup-Only Simulations
LU	Lookup and Update Simulations
MPI	Master Patient Index
NSPT	Node Server Processing Time
ORB	Object Request Broker
PHC	Primary Health Care
PHCC	Patient Health Care Centre
PID	Patient Identifier
PIDS	Person Identification Server
PLS	Patient Location System
PMR	Patient Meta-Record
PT	Processing Time
RIS	Radiology Information System
RM	Roaming Manager
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SBRC	St. Boniface General Hospital Research Centre
SCP	Service Class Provider
SCU	Service Class User
SDCT	Node Server/Directory Communication Time
SOP	Service Object Pair
SQL	Structure Query Language
TRT	Transaction Response Time
TRT-C	Transaction Response Time - Client
TRT-NS	Transaction Response Time - Node Server
UDP	User Datagram Protocol
UID	Unique Identifier
UO	Update-Only Simulations

Chapter 1

Introduction

Medical images are key assets that provide a means for healthcare professionals to evaluate patient diagnosis and treatments, conduct research of underlying diseases, facilitate residents training programs, *etc.* Traditionally, medical images are captured, printed, and stored using films. Film utilization is inefficient because images must be physically stored and manually retrieved, which significantly affects the speed of healthcare delivery. This also introduces costs for storing, delivering, and managing films in terms of both capital resources and people. To speed up healthcare delivery and reduce operation costs, digital imaging and digital image management system are used to replace radiographic films. This transition changes the operations and diagnostic procedures of radiology departments. Digital imaging enables medical images generated from a variety of radiologic imaging modalities such as Computer Tomography, Magnetic Resonance, Computer Radiography, Ultrasound, *etc.* to be digitized and made available for online viewing on computer workstations. Digital image management system also called *picture archiving and communication system* (PACS) is designed to facilitate the integration and distribution of digital images and image related data from various diagnostic imaging modalities in a radiology department.

1.1 Picture Archiving and Communication System

A PACS is a computer system composed of several subsystems. These subsystems include a data acquisition system, PACS controller system, archive system and display system. A data acquisition system consists of acquisition computers used to acquire image data from radiologic devices. Once the data acquisition system has acquired the image data, it converts the image data from the vendor specific format to a PACS standard format for storage and transmission. The formatted image data will then be delivered to the PACS controller system and/or display system for storage and/or interpretation. The PACS controller system is the PACS engine that controls the data flow within the PACS. The PACS engine consists of two components: the database server(s) and archive system, responsible for updating patient's records and storing related image data sent from the acquisition system. It also services requests from the display system. The archive system provides short, medium, and long term storage devices for storing images and patient related information. Finally, the display system consists of a set of display workstations for displaying images and assists users in interpreting images along with relevant data. These subsystems are interconnected with various network facilities to ensure digital images and image related data are available to multiple imaging departments so it is efficiently stored and retrieved throughout a hospital. A PACS also interfaces with other medical information systems. For example, it interfaces with the *hospital information system* (HIS) and the *radiology information system* (RIS) to acquire important data such as patient information, examination schedules, and study types for effective operations [Hua99].

The components of healthcare information systems such as PACS, RIS, and HIS vary with computer applications, platforms, imaging modalities, and vendors. Therefore, two standards have been proposed and used to deal with the heterogeneous textual data and medical images among these components. These two standards are *health level 7* (HL7) and *digital imaging and communication in medicine* (DICOM).

HL7 is a standard used to handle the heterogeneous textual data formats of different computer applications from different vendors in a hospital environment. The goal of the HL7 is to simplify the interoperability problems by standardizing the data format and the protocol for exchanging key textual data among healthcare information systems [Hua99]. While DICOM provides guidelines on how image data can be formatted and exchanged. DICOM introduces the concept of an information object that is used to define and describe the contents of images, studies, reports, *etc.* Information objects are entities or a collection of entities defined using the DICOM *information object definition* (IOD) model. IOD is an object-oriented abstract model used to represent a real world entity such as a patient, study, image, *etc.* There are two groups of IODs. One IOD that represents a single real world entity is called a normalized IOD. A second IOD combines multiple normalized IODs and is called a composite IOD. Each IOD consists a set of attributes that describe a single piece of information about the entity when it is created. Attributes may include patient name, image identifier, study unique identifier, study date, modality type, *etc.* IODs permit objects to be defined precisely and to be used by heterogeneous medical imaging applications. An information object is uniquely identified through an entity-relationship model. DICOM also defines different service classes that are used for exchanging information objects between computer applications and perform operations on these objects. There are two roles defined in DICOM. A machine that issues a service request is called *service class user* (SCU). A machine that receives the SCU requests and performs operations on the information objects is called a *service class provider* (SCP). A service and object form a fundamental DICOM unit called *service object pair* (SOP).

Services are requested through a set of *DICOM message service elements* (DIMSEs). A DIMSE describes the SCU requests and functions to be carried out by the SCP on the information objects. The goal of the DICOM standard is to facilitate interoperability and to integrate disparate medical images and related data generated from a variety of modalities and manufacturers [Che01].

1.2 Health Distributed Archiving System

There are many advantages of introducing PACS to the traditional film-based approach in radiology and medicine. These include reduced film costs, storage, and reduce the liability for misplacing films. PACS also facilitates the diagnostic process as technicians and radiologists require less time spent on image acquisition, delivery, reading, diagnosis, writing reports, *etc.* From the advantages provided by PACS, it becomes a critical component in any healthcare enterprise. Many hospitals employ a PACS or several PACSs to assist healthcare delivery. A PACS can be either simple (small-scale) or complex (large-scale) depending on the applications and the needs of a hospital. A simple PACS may consist of an image acquisition device with a digitizer connected to a display workstation. The display workstation has a database management system for storing image data. A complex PACS may be a hospital-wide PACS that integrates independent PACSs residing at several imaging departments inside a hospital. However, given the distributed nature of any healthcare industry and the reality that patients often utilize the services of several hospitals, the integration of PACSs should not be limited to those collocated in a single hospital. A PACS can also be extended to multiple PACSs residing at several remote hospitals inside or outside of a specific health region. In addition, the PACS networks at different hospitals need to be interconnected to share radiology medical images and related patient data. In response to these necessities, a distributed computing system called *health distributed archiving system* (HDAS) has been proposed and is currently being developed at the St. Boniface General Hospital Research Centre (SBRC) [CB01]. The main objective of the HDAS is to provide a common layer for the integration, transmission, storage, and retrieval of medical images and related patient data scattered/distributed at different PACSs throughout regional hospitals that consists of two or more facilities (e.g. Winnipeg health region has nine hospitals).

1.3 Motivations and Objectives

The first mission of the HDAS has been to integrate radiology components of different health regions. Establishing a common layer in different health regions results in several challenges and issues that the HDAS must overcome. One of the challenges is to efficiently and effectively access, integrate, and share patient's medical images and related data. In particular, data that is being generated, distributed, and stored at different radiology departments across several remote hospitals must be managed correctly. For example, consider a radiologist who reads newly acquired images of a particular patient in front of a display workstation. For diagnostic purpose, the radiologist may wish to compare the newly acquired images with the patient's previous images that are possibly spread across several remote systems residing at different hospitals. Assume that hospitals within a health region are interconnected by a network facility. One way of locating the patient's previous images is broadcast a message that contains patient's information to all systems inside the health region. Upon receiving the message, each system is required to check whether it has images related to that patient. If so, the system responds to the calling system by sending a message indicating that some images associated with that patient are found. Once all messages from all systems have been received, the calling system returns a list to the radiologist. The radiologist will then select one or more images from the list and subsequently obtain all images. The major drawback of this approach is inefficient as the network grows and the number of users increases within the hospital region. The network bandwidth is wasted by messages seeking to locate images and related patient data. Users may have to wait a longer time due to the lookup operation at each system. This has motivated the development of a *patient location system* (PLS) to track and locate where patient's images and their related generated data have been stored. The PLS is a critical component of the HDAS. It provides information that is necessary and subsequently used by other components of the HDAS to respond to user requests in a timely fashion. To provide an efficient and effective PLS, its architecture and the defined database scheme for the storage and management of location information are critical.

The objective of this thesis is to design, implement, and characterize the performance of different architectures and database schemes for the efficiently tracking and locating image related data to support the delivery of HDAS services for regional healthcare data networks.

1.4 Location System

A location system is a facility that manages information about the locations of objects and performs operations on the information. In general, a location system comprises two main components: a *directory* and a *directory service*.

A directory is a database designed for storing information about objects that exist in a system. An object can be practically anything, depending on the domain in which it is administrated. For example, an object can be an employee of a company, a service or a device of a computer system. It can also be a directory, a mailbox, or a department of an organization unit. Objects in a directory are identified through their names. Names offer a convenient mechanism for accessing objects in a system. A name can be a human-readable name tailored to be used by humans, a non-reusable unique name (identifier), or an address that provides an access point where a particular object can be contacted [Tan02]. In addition to the name, each object is typically associated with a set of attributes. The attributes to be stored in the directory are different from one context to another, which defines explicitly the purpose and the use of information needed by the system. The set of attributes, for example, may include the telephone numbers of a user or resources and services provided by systems [Nov03]. It could also be the location or Internet address of a Web page server. An attribute carries a list of data repositories [Ord93], a service available in an open environment [Bbv95], and a set of forwarding pointers for keeping track of the current location of migrating agents in a mobile communication system [Mor02].

Names and attributes can be organized in many different ways, depending on the data model used and the directory's implementation. Access to objects and their associated attributes also depends on the directory implementation. A directory can be implemented as a set of flat files, relational tables, or hierarchical trees [Shi00]. When a directory is implemented using a relational data model, information may be stored in one or more pre-defined tables and accessed with *Structure Query Language* (SQL) manipulation language. On the other hand, a *Lightweight Directory Access Protocol* (LDAP) based directory, which organizes information hierarchically in a tree structure called *Directory Information Tree* (DIT) and can be accessed through the LDAP manipulation language [Fit97].

Objects in a directory can be accessed and manipulated through names and their associated attributes. Directories are usually accessed using a client/server model. The standard rule used to access a directory is called an *access protocol*. In the previous examples, SQL and LDAP are access protocols that describe how objects can be accessed from the relational tables and DIT, respectively. A process that performs operations and accesses to any information stored in a directory is called a *directory server*. A process that requests a service from a directory server is called a *client*. The operations offered by a directory server to its clients are collectively referred to as the *directory service*. Basic directory operations include store, lookup, update, search, delete, and so on. For example, looking up or searching directory information can be done as follows: a client sends a lookup/search request that contains the name of an object or a search criterion expressed in terms of a set of attributes to the directory server. Upon receiving the client's request, the directory server accesses the directory, performs lookup/search operations, and returns results to the client.

Because of the directory service is tightly associated with a directory and implemented by means of a directory server, the directory service is sometimes referred to a "directory". A directory can be implemented either centralized or distributed. A directory is centralized if there is only one directory and the directory server is located at a single node (computer) of a computer network. On the other hand, a directory is distributed if some of its data are spread across several nodes and is implemented with multiple directory servers interconnected through a computer network. In addition, each directory server is autonomous and capable of executing some local operations on its data. A directory server may also participate in the execution of some global operations that require accessing data at several nodes. Information contained in a distributed directory can be partitioned, replicated, or a combination of both, to several nodes to improve the performance and the availability of the directory.

1.5 Summary of the Performance Analysis

Performance is a standard way of measuring the design decisions made on a system. In a large-scale system, no matter if it is centralized or distributed, the performance of the system is especially critical. There are many performance measures that can be used to evaluate the performance of a system. Performance characteristics, such as response time and throughput, play an important role in defining the quality of a system, particularly in large-scale system with multiple components that spread across several locations. For the PLS to be truly effective and to have a high performance, it must be demonstrated that it can provide services with its users with low response time. Moreover, it must be capable of handling extensive workload without substantially degrading the system throughput. Response time is a major concern from the radiologists' perspective as they often expect services to take place almost instantly to facilitate their daily diagnoses. Additionally, HDAS is expected to have large numbers of users utilizing its services in the near future. Thus, the throughput of the PLS is important, especially aimed at providing location support to large-scale systems such as HDAS. For these reasons, the response time and the throughput of the PLS are the major performance measures considered in this thesis.

1.6 Summary and Contributions of the Thesis

One of the major challenges of the HDAS is to efficiently and effectively access, integrate, and share patient's medical images and related data. In particular, the data generated, distributed, and stored at different radiology departments across several remote hospitals. This thesis presents a design for a PLS with centralized and distributed approaches for tracking and locating patients' medical images and their related generated data to support the delivery of HDAS services. The system's directory will be designed using relational and LDAP based (i.e. hierarchical tree structure) data model. The prototypes of various system architectures are implemented. In addition to these, a simulation model is developed to conduct the performance analysis on various system architectures.

1.6.1 Contributions of the Thesis

In this thesis, the emphasis is on the performance analysis of the PLS. The results of the performance analysis provide insight on the combined effect of the database scheme and the architecture used to design the system. Results can also be used as a building block to develop a suitable database model for the storage and management of DICOM images and related patient data.

1.7 Organization of the Thesis

The balance of the thesis is organized as follows. Chapter 2 provides a survey on various location services specifically used in hospital environment and different middleware technologies used in distributed computing systems. Chapter 3 describes the PLS architectures and directory designs. The implementation details of various components of a PLS and the performance simulation model are given in Chapter 4. Chapter 5 presents the performance analysis details and discusses the experimental results of the performance of various PLS configurations. Finally, Chapter 6 provides the conclusion of the thesis and discusses the future directions of the PLS.

Chapter 2

Literature Survey

Many approaches have been proposed and discussed in the literature concerning how to deal with location and identification of patients and their related medical information in a hospital environment. The main objective of a location service/system is to provide users with fast access to objects that may generate, store, and/or be distributed to various sites interconnected through network facilities. In Section 2.1, we overview some existing approaches applied to the healthcare domain.

As mentioned in Section 1.4, directories are generally implemented using the client/server model. If a location system is intended for a distributed system with a small number of entities and users are restricted to a local area network, it is often feasible to implement the location system using the centralized approach. However, in enterprise-wide distributed system with many entities residing at geographically dispersed sites interconnected through network facilities, the implementation of a location system may also need to be distributed. In such a setting, the distributed location system is necessary to interoperate with other components, applications, and systems in distributed heterogeneous environment. A distributed system incorporated a middleware technology into the client/server architecture aimed at improving interoperability, flexibility, and maintainability of the system. In the past few years, many middleware technologies have emerged aimed at simplifying and facilitating the development of applications for distributed system. In Section 2.2, we will overview some of these technologies.

2.1 Location Services

A framework proposed [Gsm00] attempts to improve the diagnostic processes of regional and national *patient health care centres* (PHCCs) in the *primary health care* (PHC) environment. It adopts the *Common Object Request Broker Architecture* (CORBA) and web-based (Intranet/Internet) technologies to support distributed access to *electronic health care records* (EHCRs). The framework focuses on the EHCRs being generated and distributed at several remote PHCCs. Each PHCC is associated with a HIS responsible for maintaining a database, which stores all locally registered EHCRs. It is assumed that each patient's EHCR will register at the PHCC located near the location where the patient lives and subsequently become the local PHCC for storing the patient's EHCRs. Since patients may visit several remote PHCCs scattered throughout different regions, the concept of roaming EHCR (R-EHCR) is introduced. A R-EHCR is created from the EHCR, as the patient visits a new PHCC, possibly located at a remote region. Each region's HIS is required to maintain a cache database that stores the patients' R-EHCR. To facilitate the location of patient's EHCRs or R-EHCRs, the framework also introduced a centralized entity called the *roaming manager* (RM). The RM is responsible for maintaining a global master EHCR index of all EHCRs (i.e. the locations of all HISs). To locate EHCR or R-EHCR in a timely way, the global master EHCR index is cached at each HIS and updated in a specific period of time. Locating a patient's EHCR or R-EHCR is done in the following manner. When a patient visits a remote PHCC, the remote HIS lookups its locally cached index to identify the location of the patient's EHCR. Once specific location information is obtained, the remote HIS gets a copy of the EHCR from that specific location and subsequently stores it in the local R-EHCR cache. After the EHCR is obtained, the HIS then notifies the RM to update the location information of the R-EHCR. The cached EHCR (R-EHCR) will remain at the remote PHCC and be deleted after a specific period of time. However, if a patient keeps visiting the remote PHCC, the master EHCR will be deleted from its local PHCC and the remote cached R-EHCR will then become the patient's master EHCR. If transferred, the HIS contacts the RM and updates the EHCR location information. However, the authors [Gsm00] did not provide specific details on how and which database model was adopted to implement the global master EHCR index.

The PACS database described by Trayser [Tra94] maintains image and radiological examination data associated with patients. The PACS database does not contain clinical data but only that used to describe images. Clinical data and patient associated information such as patient identification, medical history, and the sequence of radiological examinations are handled by the HIS core database. The HIS core database provides an access point for the RIS to obtain patients' demographics and clinical data associated with patients. The PACS database in turn interfaces to the RIS to mange patients' digital images. The main function of the PACS database is to provide a directory of all digital images obtained from different imaging modalities and subsequently delivering these images to several display severs for interpretation. The PACS database is implemented using the INGRES relational database management system. The directory is in turn implemented using several relational tables, which include patient table, examination tables, and a set of image description tables. The patient table is the main table of the PACS database as it contains patient identification entries. Each patient identification entry is linked to a set of examination tables, which in turn are linked to a set of images description tables. The patient table is linked to the HIS core database as well for referencing patients' master entries. In addition to these tables, the PACS database also maintains an update list to record the locations of all duplicated files that are being distributed to different display servers throughout the hospital. The update list is used to facilitate the cleanup (removing necessary images) at display servers. Users access to the PACS database is through a computer application called *Interactive System for Image Selection* (ISIS), which acts as a browser of the PACS directory and a transfer operator for digital images.

Tsiknakis *et al.* [Tsi95] describe the *patient meta-record* (PMR) concept that provides a unified way to access *computerized patient record* (CPR) segments maintained at different healthcare institutions scattered throughout a hospital region. The PMR stores meta-information about patients and provides the information source for accessing a patient's CPR. Meta-information includes the type, location, date of generation, means of access, *etc.* that can be used to access patient CPR segments local to a healthcare institution. A CPR segment includes several information objects that are used to describe a patient's diagnostic study, examinations performed on a single modality, primary or follow-up interpretations, and the results of diagnostic examinations. Information objects are organized in a way that follows the information object model described in the DICOM 3.0 standard. CPR segments are organized hierarchically in a tree structure database (CPR directory) local to the healthcare institution. Access to a CPR segment of a given patient is achieved through a CPR directory server, which uses the information contained in the PMR and the LDAP directory access protocol. There are several CPR directory servers distributed across the healthcare network to provide directory services for healthcare institutions. Asynchronous messages are broadcasted regularly among the

CPR directory servers to keep their CPR directories consistency. The PMR concept is further extended and used in a domain-specific framework [Lei97]. The proposed framework integrates heterogeneous healthcare information systems that are widely distributed and in an attempt to establish a distributed telemedicine services environment. It uses the PMR concept as well as standard directory services to facilitate services provided for data mediation, distributed directory access, and workflow management among various heterogeneous healthcare information systems. The PMR and the distributed directory services are also used to form the concept of a virtual patient record by integrating CPR segments maintained by those systems. Information contained in the extended PMR includes the meta-information about the information systems involved (information structures and their semantics), the locations of service objects, and healthcare data. The CPR directory is implemented using the X.500 directory model and accessed through directory servers using the LDAP access protocol. The framework also made use of the CORBA, a distributed computing technology to integrate, store, discover, and advertise information about object services.

Forslund *et al.* [For98] introduced a collaborative environment in which multiple users at different locations can simultaneously access, view, edit, and interpret patient data. The main purpose of the collaborative environment is to allow interactive consultations and timely communication among healthcare providers (e.g. physicians), patients, and insurers, who are possibly located at different healthcare facilities across a wide-area network in different healthcare domains. The collaborative environment was implemented on a system called *TeleMed*. The TeleMed system is developed based on the virtual patient record concept. The virtual patient record concept is introduced to handle various data formats and data structures of information systems involved in a collaborative consultation. For various reasons, a patient's medical records might scattered at different healthcare facilities, possibly across national or international boundary. The virtual patient record concept provides users with a unified view to patient's medical information maintained at different healthcare facilities. A virtual patient record for a given patient is created on demand from collaborating healthcare institutions when required. Forslund *et al.* [For96], argue that fast and securely identify patients as well as their healthcare providers are major factors affecting the feasibility of the virtual patient record concept. For these reasons, the *master patient index* (MPI) is used to identify patients and their respective healthcare providers in a timely and secure way. The MPI consists of two components: (1) unique names used to uniquely identify patients and (2) pointers used to point to the locations where patients' medical information can be found [Kil97]. MPI provides a linkage to distributed heterogeneous systems that constitute an interactive consultation. Information for creating MPI consists of patient demographics such as name, gender, date of birth, and so on that can be used to identify a patient and his/her disparate medical records. A patient or a healthcare provider is identified by means of a *person identification server* (PIDS) [For98]. The PIDS is one of the components of the TeleMed system. However, the authors do not specify how the PIDS accesses the MPI to locate patient medical information.

Zisman [Zis98] proposes a distributed information discovery system to support location and identification of medical data related to users' requests. It also supports data access among heterogeneous autonomous databases that were created and administrated at different departments in a hospital environment. The main objective of the system is to limit information discovery to a group of database systems called a federation, which either contains data relevant to the user's request or information related to a database system that possibly holds the requested data. A federated database systems is formed when a set of database systems wish to share and exchange data with each other. The creation of a federation is based on the shared data of each database system as well as users and applications that are authorized to access the shared data. Within a federation, database systems interoperate to provide functions for processing users' requests, locating requested data, and performing query translations. A database system in one federation can also participate in other federations when needed. A federation is further divided into a second level with groups based on the type of the shared data contributed by the participating databases. This second level is used to limit and facilitate the search during the information discovery process. Additionally, each database system consists of a set of additional structures to facilitate interoperability between heterogeneous databases for each federation where the database system is participating. *LOCate* is one of the additional structures used to locate each database system. The main function of the LOCate is to record the contents and the locations of different database systems to support data location and data distribution. Information contains "the names of the groups of the related federation and specialized terms" that are organized hierarchically in the LOCate structure. Specialized terms are used to reference related database systems to provide support for users during an information discovery process. Terms are defined based on the local schema of a database system and the instances of participated databases. A term is selected based on the interests of users and applications as well as data shared by each database system. Due to the process of defining and administrating terms is complex, human assistance is required to define and validate terms. Further, the hierarchical structure of the LOCate is changed as the results of any addition, modification, and removal of databases to the information discovery system. Therefore, a tool is provided to construct LOCate interactively with the coordinator of a related federation and the participating database systems. The information discovery system also implemented an algorithm to facilitate dynamic construction of LOCate hierarchy.

2.2 Middleware Technologies

By using middleware technology the client/server model has improved the interoperability, portability, and flexibility of a distributed system. Middleware technology is designed to provide connectivity between applications, programs, or processes. It manages communication and data exchange between components in distributed systems. It supports heterogeneous applications to interact and communicate with each other, without regard to the languages, platforms, and locations. Therefore, developing distributed computing systems using middleware technology supports interoperability across languages and platforms, as well as enhancing maintainability and adaptability of the system. In the past few years, many middleware technologies have been developed and proposed to simplify the development of applications for distributed system. The most widely used middleware technologies include *Remote Procedure Call* (RPC), *Remote Method Invocation* (RMI), and *Object Request Broker* (ORB), among others. These technologies have been widely adopted for implementing object-based systems in general. Each of these technologies is briefly reviewed in this section.

2.2.1 Remote Procedure Call

The *Remote Procedure Call* (RPC) is designed to handle communications between processes (programs) residing at separate machines interconnected through a computer network. The basic idea of the RPC is to allow a client component of a distributed system to make a procedure call to a server regardless of location, platform, or operating system. The idea behind RPC is to hide the remote execution from the client application and make the remote procedure look like a local procedure call. A remote procedure call is achieved by exchanging messages between the client's implementation (client stub) and the server's implementation (server stub). The client and server stubs are served as "drivers" to transmit requests between client and remote procedures. Stubs are used for marshalling, delivering, and unmarshalling messages between a client and remote procedures. A message delivered by the client stub contains the client's request, the name of the remote procedure, and the necessary parameters. The message sent from the server stub contains the server's reply to the remote procedure call. In RPC, stubs are generated using the *Interface Definition Language* (IDL), which contains the interfaces for remote procedure calls. An interface consists of a set of procedures or methods implemented by a server for its clients. The IDL is a declaration language, which describes the procedures or methods that can be called over the network and the information that can be passed to and from those procedures or methods. When utilizing RPC, the client and server stubs must agree on the RPC protocol. The RPC protocol determines the message format to be used for marshalling parameters and results of a procedure call, encoding rules for representing simple data structures, and the transport protocol for message exchange.

The main advantage of the RPC is that it increases the interoperability and access transparency of a system by allowing a client to employ a procedure call to access a server located at a remote system. The interfaces defined in IDL enable RPCs to interoperate between applications written in different languages. Most RPC employ the synchronous request-reply mechanism, which involves blocking of the client until the server replies to its request. The advantage of this mechanism is that it can be used to guard against overloading a network. However, this mechanism also has disadvantage, as it requires the client and the server to always be available and operational. Since most RPC implementations provide poor support for objects references, they make RPC less attractive when implementing applications in distributed object-based systems as compared to other middleware technologies [Tan02].

2.2.2 Remote Method Invocations

Instead of supporting transparent access to remote procedures, *Remote Method Invocation* (RMI) focuses on object method invocations in the object-oriented paradigm. It applies the RPC concept to make remote objects accessible through their object interface. Unlike RPC, object interfaces are defined using the same object-oriented language as the one for creating the remote object. In RMI, a server is responsible for creating remote objects and making them available for their clients by registering them with a bootstrap naming facility called the *RMI registry.* A client can make a call on a remote object that resides on a different machine once it obtains a reference to the remote object. Obtaining references to remote objects can be achieved either by looking up remote objects in the RMI registry or by receiving references as parameters or return values. Similar to the RPC, interfaces are compiled to produce the client's proxy (client stub) and the server's skeleton (server stub). However, the message delivered by the proxy involves method invocations rather than procedure calls.

Objects can be passed as parameters or returned as values using RMI by a remote method call between components in a distributed system. Objects can be passed by value or by reference. When an object is passed by value, a copy of the object is created and passed along with the method invocation provided the object is serializable. When an object is passed by reference, the object's proxy implementation and the necessary classes will be downloaded to the client's address space for method invocations. Once the proxy of a remote object has been loaded into the client's address space, the client can then use the proxy to make any method invocations to the object locally. In RMI, local objects are usually passed by value, whereas remote objects are passed by reference. The dynamic class and stub downloading is an essential feature of RMI. This approach allows developers to build a variety of proxies, which avoid the need to make remote invocations in certain cases. This also facilitates software distribution and ease of maintenance because classes and stubs can be loaded dynamically on demand when needed and do not need to be preinstall on client machines. Since RMI adopts the RPC mechanism for method invocations, a client is blocked until it received a response. One of the disadvantages of RMI is that all objects (interfaces) that constitute the distributed system are assumed written in the same language. Thus, RMI is limited to a single-language environment and does not allow interactions with objects written in different languages. Therefore, RMI is not well suited for implementing a large-scale distributed system that involves heterogeneous components [Sun95, Sut97, Tan02, Wal98].

2.2.3 Object Request Broker

Similar to the RMI, *Object Request Broker* (ORB) is designed to handle communication between clients and objects in the object-oriented paradigm. Unlike RMI, ORB supports communication and data exchange between objects regardless of platforms, software, and vendors. The main idea of ORB is to promote interoperability of heterogeneous distributed systems and portability across different programming languages, platforms, and ORB implementations. The basic functions of ORB include locating and activating remote objects, marshalling and unmarshalling parameters and results, and establishing communication between clients and objects. Most ORB products generally offer a directory of services, which help in locating and establishing communication between clients and services [Tan02].

ORB is the core communication infrastructure of any CORBA distributed system. CORBA is a specification of a standard architecture, which allows different vendors to develop

ORB products. It can be thought of as architecture for integrating diverse applications. All objects and services are defined in the CORBA IDL. Static stubs and skeletons are generated at compile time by an IDL compiler. Section 2.2.1 explains how IDL is used to describe the methods that can be called over the network and the information that can be passed to and from those procedures or methods. CORBA also defines mappings from IDL to all supported programming languages such as C++ or Java. In CORBA, a method can be declared as a one-way operation that makes method invocation unidirectional without returning a result or success acknowledgement to the sender. This is an advantage for the client, as it is not blocked until the competition of the operation. Many CORBA compliant systems support a dynamic invocation interface that allows dynamic request generation [Inp00]. This is useful when the client has no compile time knowledge about the interfaces it is accessing. CORBA also provides specifications for implementing CORBA compliant services such as naming, collection, transaction, query services, *etc.* In CORBA, flexibility is obtained by using object adapters (e.g. Basic Object Adapter and Portable Object Adapter) to associate an object implementation with an OBR that de-multiplexes and dispatches the requests [Inp00]. A CORBA system typically consists of a collection of CORBA services, which are considered fundamental for building interoperable and portable distributed applications [Tan02]. There are a number of ORB/CORBA products available including Java IDL, Java RMI over IIOP, VisiBroker® for Java, *etc.*

This thesis uses the VisiBroker® for Java developed by the Borland Software Corporation to implement the location system because of its robustness [Che02]. The ORB of the VisiBroker® for Java is fully compliant with the CORBA 2.3 specification. It provides various services to facilitate the development of distributed applications. These services include naming, distributed directory, location, service discovery services, *etc.*

Chapter 3

Patient Location System

The goal of a *patient location system* (PLS) is to provide an efficient mechanism for tracking and locating a patient's images and their related generated data. In particular any data that are being stored and/or distributed to various nodes (sites) interconnected through network facilities. The PLS is one of the critical components of the HDAS. The PLS is primarily used to provide information that is necessary and subsequently used by other components of the HDAS to provide users with timely services.

The PLS is designed using the client/server model with the use of VisiBroker® for Java (i.e. ORB product) that enables communication between clients and servers. In the client/server model, a client is a process that requests a service from a server by sending it a request. A server is a process that provides (implements) specific service(s) for its clients. Client applications usually manage the front-end (user-interface) portion of a system and facilitate interactions between users and other parts of a system. Servers, on the other hand, provide the back-end processing of a system such as managing shared resources, maintaining and protecting persistent data, providing application services, and so on. Depending on the system configuration, a client and server may reside both on a single machine or separate machines that may be located at different nodes. In the former case, the communication between a client and a server is typically done over a network even though they both reside on a single computer.

There are many different ways to organize clients and servers in a system. The client/server model can be organized as two-tier architecture, three-tier (multi-tier) architecture, and multi-tier architecture with middleware technology, to narrow a few models. An architecture in turn can be centralized, distributed, or a combination thereof that facilitate the connections between users and shared resources. The distinction between these architectures is based on the distribution of system processing, functions, data, and/or control. This thesis presents two PLS designs that adopt the centralized and distributed approaches. The performance of the two architectures is studied in terms of system transaction response time and throughput. Both PLS designs are implemented using Java technology because of its simplicity, openness, and portability. According to Java documentation, Java based application can run on any platform provided that the machine used for running the application has a *Java Virtual Machine* (JVM). Java also provides a variety of *application programming interfaces* (APIs) and utility packages to facilitate the system development process.

3.1 System Architectures

To provide an efficient tracking and locating mechanism, two PLS architectures: centralized and distributed have already been designed based on the distribution of the directory and the directory service that performs operations on that directory. The centralized PLS is designed to provide a single directory for all processing nodes (hospitals) within the hospital environment. On the other hand, the distributed PLS contains multiple directories; each directory contains the information stored at one node. This chapter describes the components that constitute the two PLS architectures. The corresponding directory design is given in Section 3.2.

3.1.1 Centralized PLS Architecture

The centralized architecture is considered a baseline model for implementing the PLS. In the centralized PLS, the directory is maintained centrally at one of the processing nodes within the HDAS. The PLS directory contains information that is linked to a set of DICOM databases. A DICOM databases is used to hold metadata about patient images. In this architecture, the directory service is provided through a single directory server, which resides at one of the processing nodes across the system. The directory server is responsible for performing update and lookup operations on the information contained in the PLS directory. In addition to the directory server, there exist several components that constitute the PLS. These components include clients, node servers, ORB, and database management systems.

A client represents an entity (a user or an imaging modality) that belongs to the health domain or another component of HDAS. A node server is responsible for invoking services (objects) that are made available to its clients. An object invocation is based on the request sent from a client. An object is referred to a servant that offers an implementation of a specific service (e.g. updates the metadata of a patient's record, obtains the location of DICOM database(s) associated with a patient's record, *etc.*) invoked by clients. A servant may contact the directory server on behalf of a client when needed. The communication between a client and a servant is handled through the ORB. Finally, a *database management system* (DBMS) is a software component that provides a storage

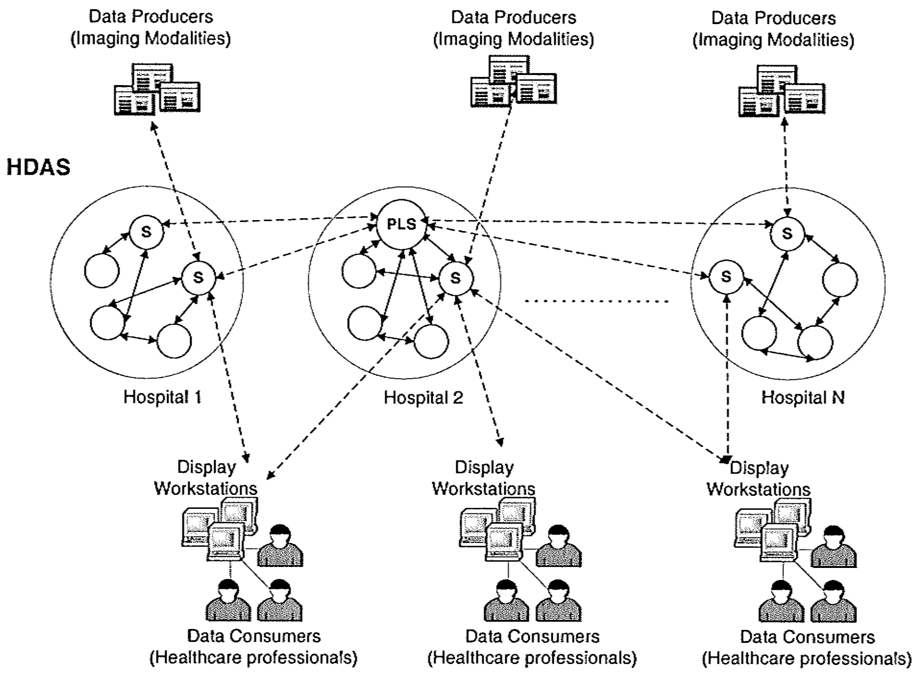


Figure 3.1: Centralized PLS located at hospital 2 of the HDAS node

mechanism and performs operations (e.g. queries, updates, *etc.*) on data items that are being stored in a database. Within the PLS (centralized or distributed), DICOM databases are implemented using the IBM® DB2® Enterprise Extended Edition (EEE) relational DBMS. Figure 3.1 shows the conceptual view of the centralized PLS.

The interactions between the components in the centralized PLS can be described using the following example: assume a client at one of the processing nodes within the PLS needs to update the metadata of newly acquired images of a patient. To make an update request, the client first uses the ORB bound to a servant (i.e. object) that implements an update service residing in the local node server. Upon receiving the client's bind request, the ORB locates the node server and obtains the reference to the update servant. When the ORB successfully obtained a reference to the update servant, it returns the reference

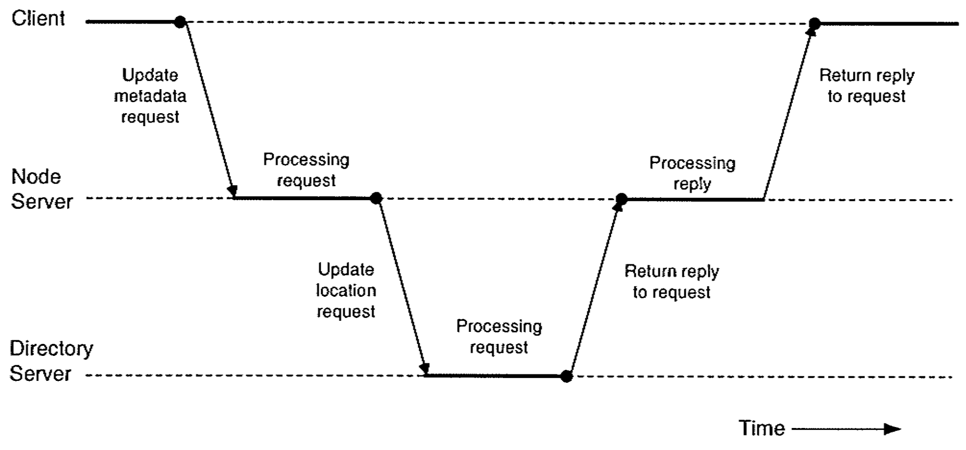


Figure 3.2: The interaction between client, server, and directory server in PLS

back to the client and establishes a connection between the client and the servant. When the client holds the reference, it invokes the methods implemented by the servant. The servant then contacts the DBMS of the local DICOM database, which will then update the metadata of a patient's record on behalf of the servant. When the metadata of the patient's images is successfully stored in the local DICOM database, the update servant generates an update request to the directory server. Upon receiving the servant's request, the directory server updates the location information (i.e. the network address of a node where the DICOM database resides) of the patient and then sends a reply message back to the servant. Finally, the servant processes the reply sent from the directory server and subsequently sends its reply back to the client. The interaction between the client, server, and the directory server is depicted in Figure 3.2[1]. The interaction between components to perform lookup request is the same as the update request except that the client may go to one or more nodes to request the metadata of a particular patient after receiving a response from the local node server.

[1]For simplicity, the ORB, servant, and the DBMS are not shown in the figure.

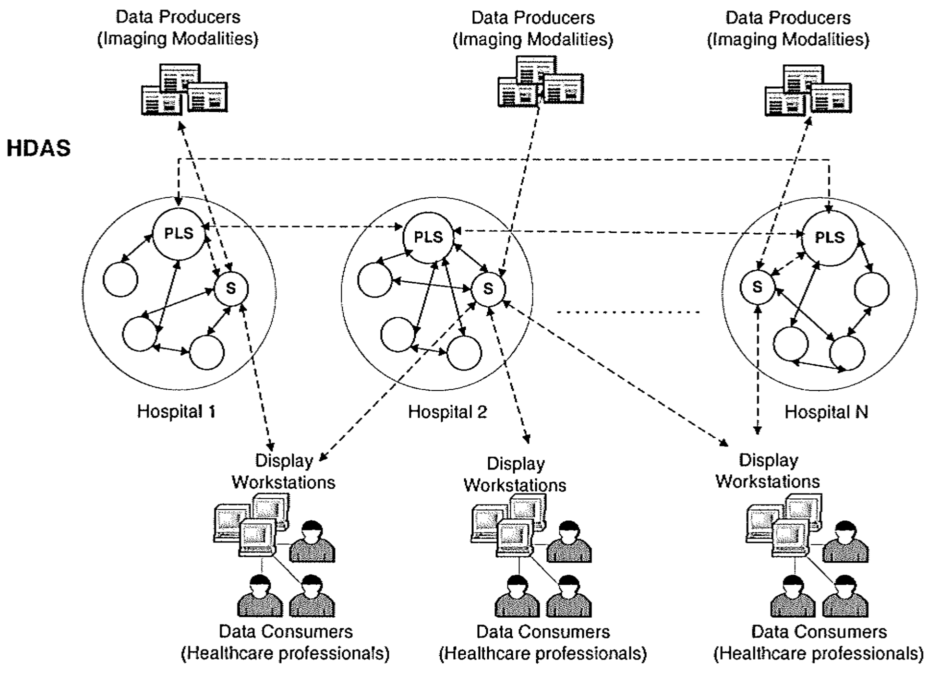


Figure 3.3: Distributed PLS in HDAS

3.1.2 Distributed PLS Architecture

The distributed PLS is similar to the centralized PLS except that each processing node maintains its own directory, which is subsequently replicated to other nodes to form a system-wide directory service. In addition, each node has a directory server that performs update and lookup operations on location information stored in the local directory. Interactions between components are similar to that of the centralized PLS, with the only difference being that the servants of a node server will contact its local directory server to perform directory operations. Figure 3.3 shows the conceptual view of the distributed PLS within the HDAS.

In addition to performing update and lookup operations, the directory server at each node must update the replicated location information at other directory server nodes regularly (i.e. an interval that is appropriate for system consolidation). The main purpose of replication is to improve the data availability, accessibility, and reliability of the distributed PLS. There are many advantages to having data replicated at multiple nodes. These include reducing single points of failure and protecting data against corruption as the availability and the accessibility of data increase. Replication also increases the reliability of the system because the PLS allows user access to data continuously even when some of the nodes are not operational. Furthermore, local queries can be distributed to nearby nodes to achieve better load balance, which in turn improves the overall performance of the PLS. However, replication poses the problem of maintaining consistency among replicated directories. This is because the content of an individual directory is changed whenever an update (insertion, modification, and deletion) operation is performed on the replicated data. Therefore, whenever an update operation is performed on any replica, the update has to be propagated to all replicas to ensure global consistency across the system.

To design an update propagation strategy for the distributed PLS, we need to specify what data has to be transferred as an update (entire content or semantic), where update propagation should be initiated (push or pull), and when an update needs to be transmitted (immediate or eventual) [Gra96, Sai01, Wie00]. In the distributed PLS, we adopt the push-based and lazy replication approaches to apply updates among replicas across the system. Push-based approach states that each replica (node) has the responsibility for pushing pending updates to other replicas (nodes) [Sai01]. Lazy replication focuses on data availability and maintains a high degree of data accessibility. Lazy replication allows temporary inconsistencies among replicas because of the assumption that all replicas will eventually be updated. Therefore, whenever an update operation is performed on any one of the replicas, the update operation is not propagated immediately to other replicas. Instead, update operations are propagated to other replicas eventually to reflect the changes [Gra96, Sai01, Tan02, Wie00].

The propagation of update operation in the distributed PLS is done as follows: whenever a directory server receives an update request, it updates the local directory and pushes the data to an update buffer. An update buffer is used to store new updates (i.e. the patient identifiers). Each local directory server is required to propagate (push) updates that are being stored in the buffer to other remote directory servers regularly to reflect the changes. When a remote directory server has received the updates, these updates are reflected in the local directory accordingly. This guarantees that all local directories (replicas) within the system will eventually become consistent. As this research focuses on studying the performance of different PLS architectures, all components are assumed within the distributed PLS so they are reachable and operational at the time the performance simulation takes place. Therefore, issues related to recovery from network partitioning, hardware or software failures are not addressed in this thesis.

3.2 Directory Models

A directory is a database, which stores information about objects that exist in a system. A database is a structured collection of data related to those objects. Objects can be virtually anything depending on the purpose of a database. For example, a database may be created for storing information about employees in an organization. Information could be a set of attributes (e.g. names, employee numbers, salaries, *etc.*) that can be used to uniquely identify an employee. From the PLS perspective, objects are referred to patient information stored in one or more DICOM databases located at different locations in the hospital network. Data contained in a database are usually organized in way that can be easily accessed, managed, and updated. There are many data models proposed for organizing data in a database. The two most commonly used are the relational and hierarchical tree data models.

In the relational data model, data are organized into one or more predefined tables called relational tables. A relational table consists of one or more rows and columns. Each row represents a unique instance of data for the attributes defined by the columns [OV91]. Relational tables are related through a set of attributes. In the hierarchical tree data model, data are organized in a hierarchical tree structure. A hierarchical tree such as the DIT described contains a collection of entries [Fit97, How95, How96]. Each entry represents some types of objects and is composed of one or more attributes. Each attribute has a type and may have one or more associated values. The attribute type determines the syntax of an attribute, which in turn determines how data will be compared with the values contained in a query. A set of attributes is referred to an object class. An object class controls a set of mandatory and optional attributes contained in an entry. An entry type is used to determine which attributes are mandatory attributes and optional. Entries are typically arranged in the DIT follow a geographical and organizational structure but is not limited to such an organization.

In the PLS, the DICOM database is created based on the relational data model and implemented using IBM® DB2® EEE relational DBMS. A DICOM database stores the metadata of patients' images and their associated attributes. The database is designed with an assumption that a master patient index mechanism already exists in the hospital network. Each DICOM database contains several relational tables, which includes a patient table, a study table, and a series table. The patient table stores identifiers that are used to uniquely identify patients in a hospital. A study table consists of identifiers that uniquely identify a particular study of a patient. Finally, the series table contains identifiers that uniquely identify a particular series of a study.

The PLS directory in turn stores the network addresses of machines where the DICOM data for a particular patient exists. The PLS directory consists of patient and hospital identifiers. Patient identifiers contained in the directory are used to link to the patient table contained in a DICOM database. In the following sections, we present two PLS directory designs based on the relational and DIT hierarchical tree data models as described previously.

3.2.1 Relational PLS Directory

The relational PLS directory contains one relational table called *Host*. The Host table consists of two attributes: patient identifiers and hospital identifiers. A *patient identifier* (PID) is a unique identification number generated by the master patient index mechanism when a patient is registered at one of the hospitals interconnected to the hospital network. A hospital identifier is a network address of a machine where the DICOM database resides. The Host table has a relationship with the patient table of a DICOM database. PIDs contained in the PLS directory are indexed to the PIDs contained in one or more DICOM databases. The backend data store for the relational PLS directory is provided with the IBM® DB2® EEE relational DBMS. A directory server interfaces with the DBMS when performing directory operations on the relational PLS directory. Access to the relational PLS is through the SQL query language. Figure 3.4 shows the entity-relationship of the relational PLS directory and a DICOM database.

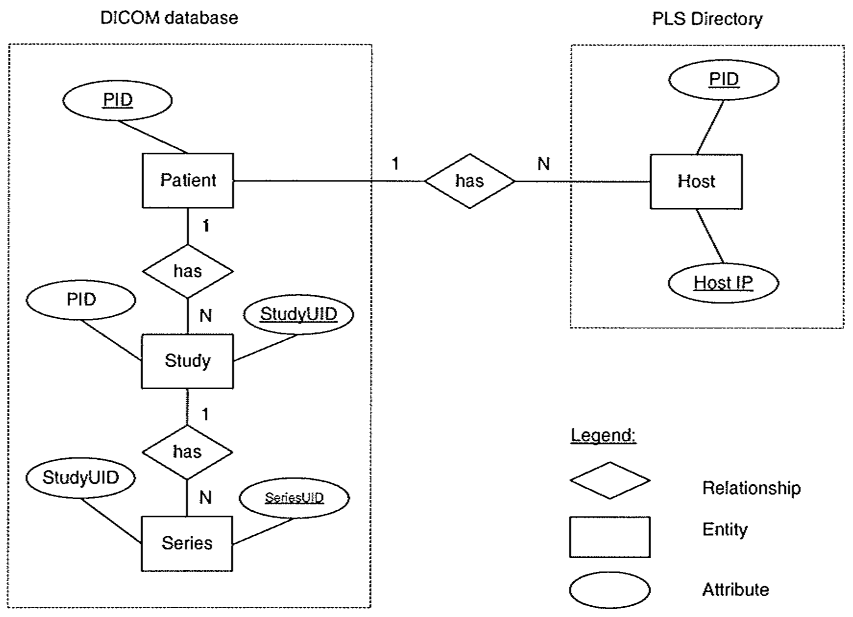


Figure 3.4: An Entity-Relationship diagram of DICOM database and centralized PLS directory

We assume that each hospital has a DICOM database that holds a metadata of patients' images. When a patient has new images, the metadata of the images will be stored in the DICOM database that resides at a hospital where the patient's images are acquired. Following this, the PID of a patient along with the hospital identifier will be stored/updated in the PLS directory. In this way, the PLS provides users (e.g. radiologists) with fast access to patient's DICOM data even though some of the DICOM data might have been stored in one or more DICOM databases at different hospitals.

3.2.2 Hierarchical PLS Directory

The hierarchical PLS directory is designed based on the DIT structure as described previously. The hierarchical PLS directory consists of one or more entries. The top-level entry contained in the PLS directory hierarchy is the root entry named *Patient*. The root

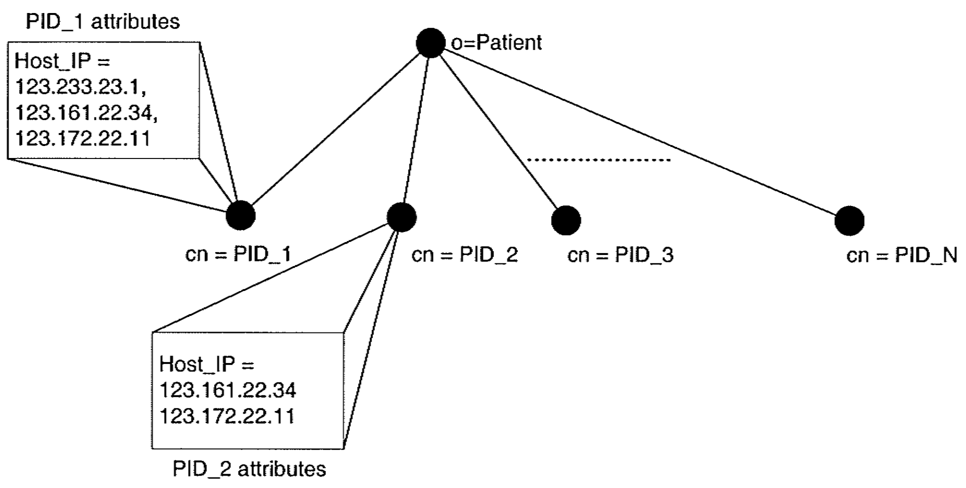

PID_1 attributes
Host_IP = 123.233.23.1, 123.161.22.34, 123.172.22.11
o=Patient
cn = PID_1
cn = PID_2
cn = PID_3
cn = PID_N
Host_IP = 123.161.22.34 123.172.22.11
PID_2 attributes

Figure 3.5: An organization of hierarchical PLS directory based on the DIT structure

entry always exists in the PLS directory hierarchy and is associated with one or more entries. Each entry under the root entry represents a patient who has been registered at a hospital interconnected to the hospital network. Each entry is uniquely identified by a PID and consists of one or more attributes. Each attribute represents a hospital identifier that indicates the location of a DICOM database. PIDs are indexed to the PIDs contained in one or more DICOM databases. The update operation is similar to the relational PLS directory except that when an entry for a particular patient already exists in the hierarchy, the new hospital identifier is simply added to that entry as an attribute. Figure 3.5 shows the hierarchical PLS directory.

The directory service for the hierarchical PLS directory is provided through a stand-alone LDAP server called *Slapd* [Oag02]. Slapd is developed by the OpenLDAP® that is aimed at supporting LDAP based directory service. According to the documentation, Slapd can run on many different UNIX-like platforms and supports LDAP over both IPv4 and IPv6 protocols. The backend data store for the hierarchical PLS directory is the Lightweight DBM called *ldbm* that is shipped with the OpenLDAP. Interfaces between the directory server and the Slapd are implemented using the Java API called *Java Naming and Directory Interface* (JNDI), which is aimed at supporting Java applications to interface with a variety of naming and directory services [LS01].

This thesis focuses on studying the performance of a PLS when different architectures and directory models applied to the implementation of the system. This is accomplished to reveal the combined effect, the advantages, and the disadvantages of different system designs. The PLS configurations under study include: centralized PLS architecture with a relational directory (CPLS-R), centralized PLS architecture with a LDAP-based (i.e. hierarchical) directory (CPLS-L), and distributed PLS architecture with a relational directory (DPLS-R). The end result of the performance study will be used to identify the best PLS configuration that is suitable for implementing a patient locating mechanism in a large-scale distributed hospital environment.

Chapter 4

The Implementation of PLS Prototypes

This research focuses on studying and characterizing the performance of three different PLS configurations: a centralized PLS architecture with a relational directory (CPLS-R), a centralized PLS architecture with a LDAP-based directory (CPLS-L), and a distributed PLS architecture with a relational directory (DPLS-R). The performance evaluation of various PLS configurations are measured in terms of system transaction response times and throughput. To study the performance of these configurations, a simulation tool is designed and implemented specifically for modeling the lookup and update operations on patient data from digital imaging departments within a hospital environment.

In this chapter, we first provide the details of how clients are modeled within the PLS. We then specify the functions of a node server and a directory server. Finally, we present the implementation of a simulation tool that is designed specifically for simulating various update and lookup requests in the PLS being tested.

4.1 The PLS Components

The PLS, whether it is centralized or distributed, is made up of several components, which include clients, node servers, a directory server(s), ORB, and a DBMS. Chapter 3 presented two PLS architectures and directory models, and discussed the interactions between clients, node servers, and directory servers. This section provides details of how clients are modeled to represent digital imaging modalities and radiologists in a radiology department. We also describe the functions (i.e. services) provided by node servers and directory servers that constitute the PLS.

4.1.1 The PLS Clients

In PLS, two types of clients are defined to emulate digital imaging modalities and the radiologists of a radiology department in a hospital environment. Digital imaging modalities and radiologists are modeled as data producers and consumers, respectively. Requests from data producers and consumers are simulated based on typical work practices of a radiology department described elsewhere [Lun99, WL01]. The radiology work practiced from the workflow perspective can be summarized as a sequence of generic activities [WL01]. These activities encompass a variety of services and functions provided by several components in a hospital environment. These components include HIS, RIS, PACS, and different types of digital imaging modalities that facilitate the overall patient examination and diagnosis process.

Lundberg [Lun99] described that for a particular patient examination, the sequence of activities starts with a request sent from a department (e.g. clinical ward, outpatient unit, *etc.*) to the radiology department. After receiving a request, a staff (i.e. secretary) at the radiology department registers the examination request by entering the patient information such as name, ID, date of birth, the name of referring clinician, type of radiological study, *etc.* into the RIS database. The patient name and the ID are then put into a work list to make it easier for a radiologist to identify a patient, fetch his/her data, *etc.* Following this, the secretary schedules the examination by reserving an examination room for the patient according to the type of a study, a radiographer, *etc.* Exam scheduling also triggers automatic prefetching of previous medical images that may be relevant to the current study from the PACS long-term archive to be displayed at the workstation for interpretation. Prefetching previous images usually occurs after midnight the night before the study occurs. When the patient arrives at the examination room, the radiographer positions the patient, calibrates the imaging modality (based on the type of study) and then acquires images. When acquisition has finished, the newly acquired images are directly stored into the PACS short-term archives using a standardized data exchange protocol (e.g. DICOM).

In the diagnostic area, radiologists interpret patient studies by reading and comparing new images with previous images (if available) in front of a group of display workstations. In general, a radiologist starts diagnosis by first clicking the patient name in the work list and checks the name and ID provided in a written request list. Following this, the diagnostic process starts including fetching patient's data, reading and/or comparing images, *etc.* After the diagnosis has finished interpretation, a report of a given patient is transcribed. The radiologist "signs off" the transcribed report, which subsequently returns to the requesting department. The radiologist starts a new diagnosis process by clicking a patient name in the work list and a sequence of activities described previously starts again. Based on the workflow aspect of a radiology department, the clients: data producers and data consumers of PLS are modeled as described below.

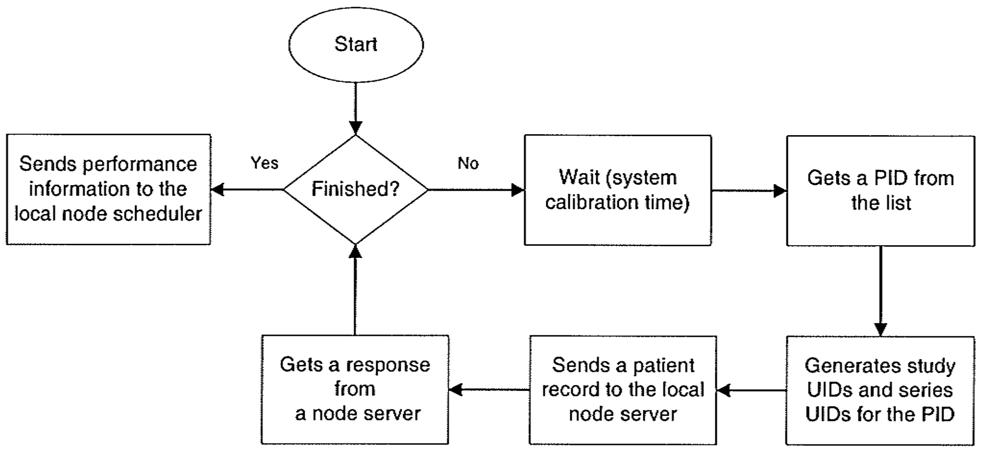

Start
Finished?
Yes
No
Sends performance information to the local node scheduler
Wait (system calibration time)
Gets a PID from the list
Generates study UIDs and series UIDs for the PID
Sends a patient record to the local node server
Gets a response from a node server

Figure 4.1: A data producer process

4.1.1.1 Data Producers

Figure 4.1 shows a data producer process that emulates an imaging modality in a hospital environment. Data producers are used to simulate update requests within a PLS. Each data producer is responsible for generating a set of metadata associated with each PID to be stored by a node server on its local DICOM database. We refer to the set of metadata associated with a PID as a patient record. Each patient record may have one or more unique identifiers called *study UIDs*. A study UID is used to identify a patient's study for a given examination. Each study may contain one or more series identified by a set of unqiue identifiers called *series UIDs*. A series UID is used to identify a series of study UID. A series contains a set of images of a patient. In PLS performance simulations, a data producer only generates a set of study UIDs and series UIDs for each patient record without any medical image. The total number of patient records generated by each producer varies, depending on how patients (i.e. PIDs) are being distributed across the hospital network. An update operation from a data producer is accomplished as follows: when a data producer has generated a record of a given PID, it sends an update

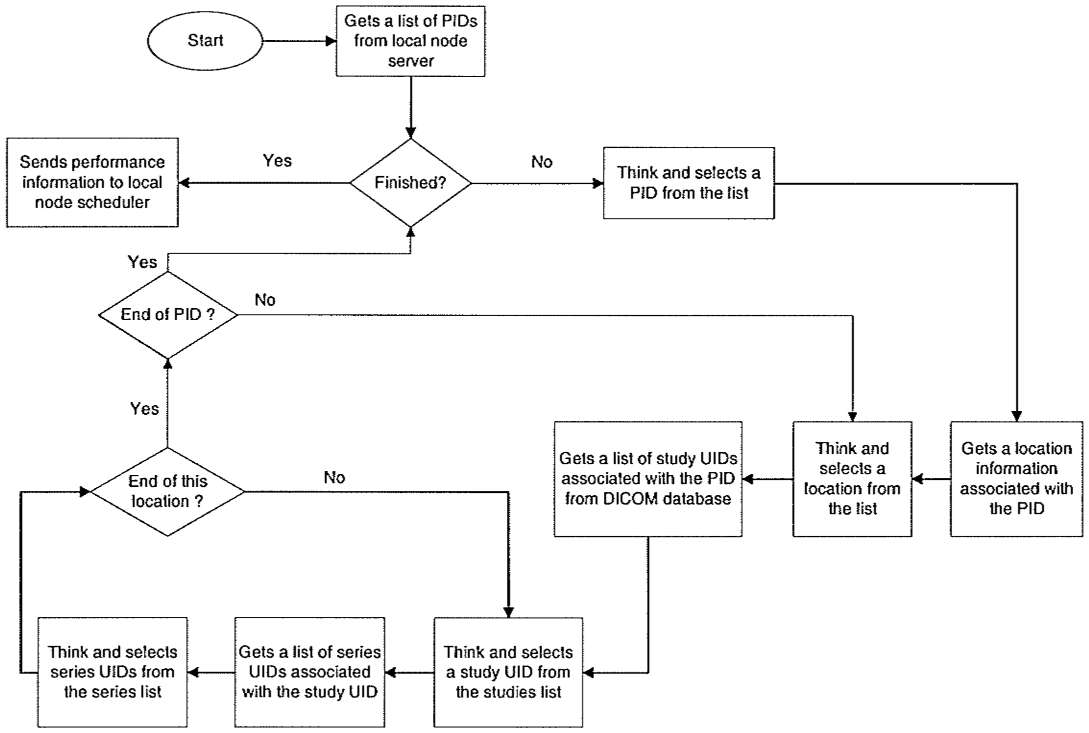

Start
Gets a list of PIDs from local node server
Finished?
Yes
Sends performance information to local node scheduler
No
Think and selects a PID from the list
Yes
End of PID ?
No
Yes
End of this location ?
No
Gets a list of study UIDs associated with the PID from DICOM database
Think and selects a location from the list
Gets a location information associated with the PID
Think and selects series UIDs from the series list
Gets a list of series UIDs associated with the study UID
Think and selects a study UID from the studies list

Figure 4.2: A data consumer process

request to its local node server. The local node server performs an update operation on its local DICOM database and then sends an update request to the directory server. Upon receiving the node server's request, a directory server updates the location information of the DICOM database where the metadata of the patient resides.

4.1.1.2 Data Consumers

Figure 4.2 shows a data consumer process representing a radiologist who accesses a patient's record by issuing a sequence of lookup requests to the local node server. A data consumer accesses particular patient's record as follows: a data consumer first generates a lookup request to a local server to obtain a list of all available PIDs. The node server will then issue a lookup request to the directory server, which performs a lookup operation on the directory and returns a list of all PIDs to the node server. When the node server has received a list of PIDs, it returns the list to the data consumer. The data consumer then selects a PID from the list and sends a location lookup request to the local node server. When the local server receives a PID from the data consumer, it sends a location lookup request to the directory server. The directory server performs a location lookup operation on the directory and returns a list of locations to the node server. The list contains locations where associated DICOM databases can be found. The node server returns the list of locations from the directory server to the data consumer. Finally, the data consumer selects some studies (i.e. study UIDs) and series (i.e. series UIDs) associated with a given PID from one or more locations contained in the list.

4.1.2 The PLS Servers

The PLS consists of two types of servers: node servers and directory server(s). Each processing node (i.e. site) has a node server that is responsible for performing operations on the metadata stored in its local DICOM database that is requesting services from a directory server when needed. A directory server is responsible for managing a directory and performing lookup and update operations on the directory.

4.1.2.1 Node Servers

A node server process is responsible for performing lookup and update operations on metadata stored in local DICOM database. When a node server starts, it registers its services (lookup and update implementations) with a directory service provided from the VisiBroker® through the ORB and waits for client requests. A node server also acts as a client which requests services from a directory server on behalf of a data producer or consumer when needed.

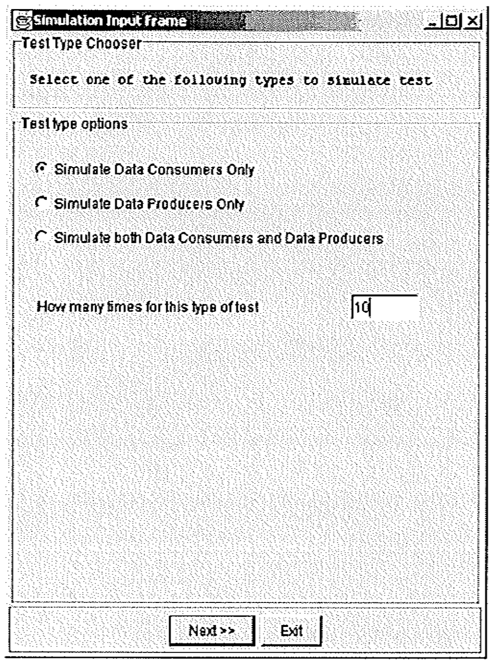
4.1.2.2 Directory Servers

A directory server is a process that performs update and lookup operations on the information contained in the directory. When performing update/lookup operations, a directory server interfaces with a relational DBMS where the relational directory model is used. A directory server interfaces with Slapd (a stand-alone LDAP server) to perform update/lookup operations when the hierarhical (i.e. LDAP-based) directory is used. In the distributed PLS, a directory server is also required to replicate its local directory information to other remote directory servers regularly so each directory server is aware of all patients and the locations of patients' metadata within the distributed PLS.

Performance metrics are measured and collected at the client side of a PLS when during the performance simulations. The tool for simulating clients and collecting these metrics is described in the next section.

4.2 Simulation Tool

Different PLS configurations are simulated and the performance characteristics of these configurations are collected using a simulation tool. The simulation tool is designed specifically for simulating update and lookup requests issued by varying the number of data producers and consumers within the PLS under test. The simulation tool consists of several modules: a graphical user interface (GUI), a master scheduler, and a group of node schedulers. These modules are provided for setting up simulation parameters, delivering parameters to each processing node, and collecting performance characteristics at each processing node in an experiment.



Select one of the following types to simulate test

[✓ Simulate Data Consumers Only]
Simulate Data Producers Only
Simulate both Data Consumers and Data Producers

How many times for this type of test: 10

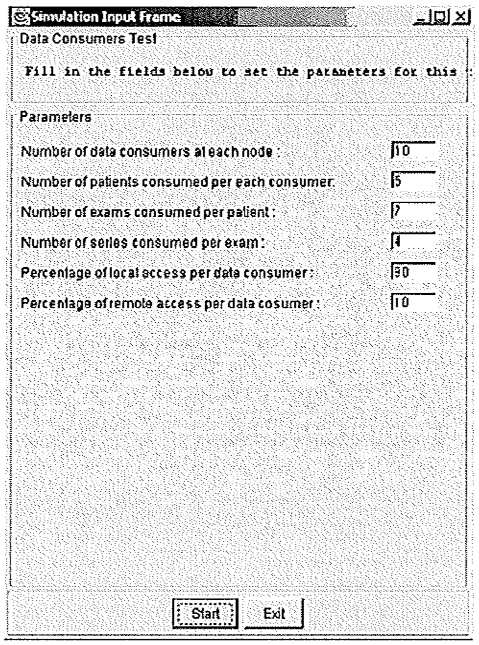
Figure 4.3: A selection frame for defining a simulation type

4.2.1 Graphical User Interface

A *graphical user interface* (GUI) is the entry point of a PLS simulation. The GUI only occurs at one of the processing nodes within a PLS and provides a variety of options to define a set of simulation parameters for an experiment. Options are contained in several input frames including a selection frame and two parameter frames. These frames are used to define the type of events (i.e. update and/or lookup events) to be simulated in a particular simulation, provide options to setup a set of simulation parameters, and validate those parameters for each simulation.

4.2.1.1 A Selection Frame

A selection frame (see Figure 4.3) is provided to define what type of events (update and/or lookup requests) are to be simulated at each processing node within a PLS. It



Data Consumers Test

Fill in the fields below to set the parameters for this :

Parameters

Number of data consumers at each node :	10
Number of patients consumed per each consumer:	5
Number of exams consumed per patient :	7
Number of series consumed per exam :	4
Percentage of local access per data consumer :	90
Percentage of remote access per data cosumer :	10

Figure 4.4: A parameter frame for setting up lookup request simulation

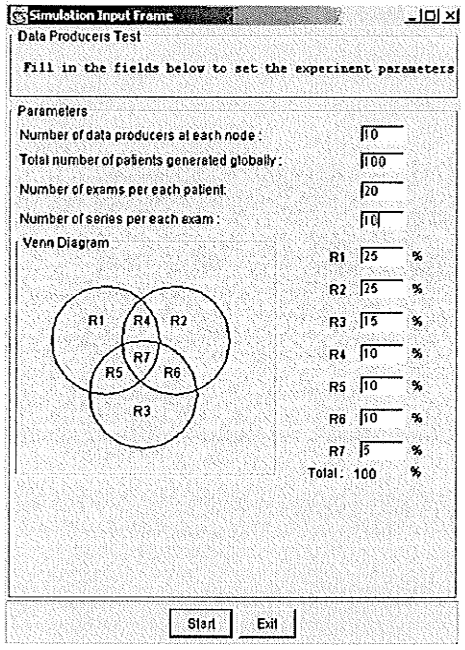
also defines how many repeated experiments will be performed with the same set of input parameters. The selection frame provides three simulation options, which include: data consumers (simulate lookup requests only), data producers (simulate update requests only), and data consumers and producers (simulate update and lookup requests). Once the type of a simulation is determined, a corresponding parameter frame appears to define a set of input parameters for a simulation.

4.2.1.2 A Parameter Frame for Simulating Lookup Requests

A parameter frame shown in Figure 4.4 is provided to define a set of parameters for simulating lookup requests (data consumers) to a PLS. This parameter frame contains a set of input fields for specifying how many data consumers will be invoked at each processing node within the PLS, the total number of patient records (i.e. PIDs) to be requested by each data consumer, the total number of studies (i.e. study UIDs) contained in a patient record, and the total number of series (i.e. series UIDs) contained in a study. It also provides options to specify the percentage of local and remote patient records consumed by each data consumer to capture the event when a patient may have several records at one or several processing nodes. We consider a data consumer's request *local* when all metadata associated with a given PID is stored at only one location and that location is the same as where the data consumer currently resided. Although the percentage of local and remote requests is defined for each data consumer before a simulation, the total number of local and remote is automatically adjusted in some cases. This occurs may be due to the total number of PIDs is not an integer and/or remote requests defined for each data consumer is less than one. For example, assume that each data consumer requests five patient records and the percentage of local and remote requests defined for each data consumer is 90 and 10 percent, respectively. Based on the defined local and remote request ratios, each consumer will consume 4.5 PIDs locally and 0.5 PIDs remotely. Since the total number of PIDs for remote requests is not an integer, the total number of remote requests is automatically rounded to the nearest integer when defining testing parameters for the simulation. In addition, each data consumer should go to at least one remote site(s) to obtain patient's data to address the fact that requested patient data may not available locally. Therefore, in this example, when a simulation starts, each data consumer will randomly select four patient records from its local node server and one patient record from a remote node(s).

4.2.1.3 A Parameter Frame for Simulating Update Requests

Figure 4.5 shows a parameter frame for defining a set of simulation parameters for simulating update requests (data producers) to a PLS system. This frame contains a set of input fields for specifying how many data producers will be invoked at each processing


Simulation Input Frame

Data Producers Test

Fill in the fields below to set the experiment parameters

Parameters

Number of data producers at each node :	10
Total number of patients generated globally :	100
Number of exams per each patient :	20
Number of series per each exam :	10

Venn Diagram

R1	25	%
R2	25	%
R3	15	%
R4	10	%
R5	10	%
R6	10	%
R7	5	%
Total:	100	%

Figure 4.5: A parameter frame for setting up update request simulation

node within the PLS. It also defines the total number of patient records (i.e. PIDs) that will be generated globally by the total number of data producers within a PLS. Additionally, it defines the total number of studies (i.e. study UIDs) contained in each patient record and the total number of series (i.e. series UIDs) contained in each study. Moreover, an option is provided to adjust patient distribution across multiple processing nodes to address the event where patients utilize services at several different hospitals. To illustrate this option, assume a PLS only consists of three processing nodes interconnected to share patient data. By using a Venn diagram, the total number of patients in a hospital environment will be distributed to seven different areas as shown in Figure 4.6.

For a particular simulation "X" patients (i.e. PIDs) may be distributed among three processing nodes as follows:

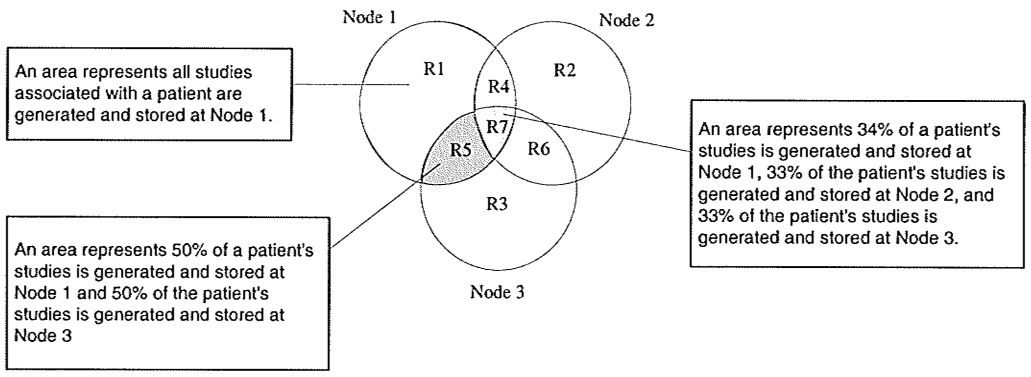

Node 1
Node 2
Node 3
R1
R2
R3
R4
R5
R6
R7
An area represents all studies associated with a patient are generated and stored at Node 1.
An area represents 50% of a patient's studies is generated and stored at Node 1 and 50% of the patient's studies is generated and stored at Node 3
An area represents 34% of a patient's studies is generated and stored at Node 1, 33% of the patient's studies is generated and stored at Node 2, and 33% of the patient's studies is generated and stored at Node 3.

Figure 4.6: Patient distributions represented in a Venn diagram

- Distribute 25% of the total patients to areas labeled as R1 and R3.
- Distribute 15% of the total patients to area labeled as R7.
- Distribute 10% of the total patients to areas labeled as R2, R4, and R6.
- Distribute 5% of the total patients to area labeled as R5.

Assume that parameters for simulation "X" are being defined as follows:

- Total number of data producers will be invoked at each node: 20 data producers/node
- Total number of patient records will be generated globally: 100 patient records.
- Total number of studies per each patient records: 20 studies/patient
- Total number of series per each study: 5 series/study
- Patient Distribution: R1=25%, R2=25%, R3=15%, R4=10%, R5=10%, R6=10%, R7=5%

Thus, a set of parameters to be delivered to each processing node within a PLS for a simulation "X" will be the parameters illustrated in Table 4.1 and Table 4.2.

Table 4.1: An example of patient distribution defined for simulation "X"

Node	No. of PIDs distributed to R1	No. of PIDs distributed to R2	No. of PIDs distributed to R3	No. of PIDs distributed to R4	No. of PIDs distributed to R5	No. of PIDs distributed to R6	No. of PIDs distributed to R7	Total No. of PIDs will be generated at each node
1	25			10	10		5	50
2		25		10		10	5	50
3			15		10	10	5	45

Table 4.2: An example of studies distribution defined for simulation "X"

Node	No. of studies distributed to R1	No. of studies distributed to R2	No. of studies distributed to R3	No. of studies distributed to R4	No. of studies distributed to R5	No. of studies distributed to R6	No. of studies distributed to R7	Total No. of studies will be generated at each node
1	500			100	100		35	735
2		500		100		100	35	735
3			300		100	100	30	530

Once a set of simulation parameters have been defined for each processing node, the GUI invokes a master scheduler. A master scheduler will then deliver the simulation parameters to each processing node within a PLS. Upon receiving a set of simulation parameters from the master scheduler, a node scheduler at each processing node will further distribute the parameters (number of PIDs and study UIDs) among a group of data producers.

4.2.1.4 Master Scheduler

The master scheduler shown in Figure 4.7 is a component embedded in the GUI. It controls each simulation and is responsible for delivering simulation parameters assigned by

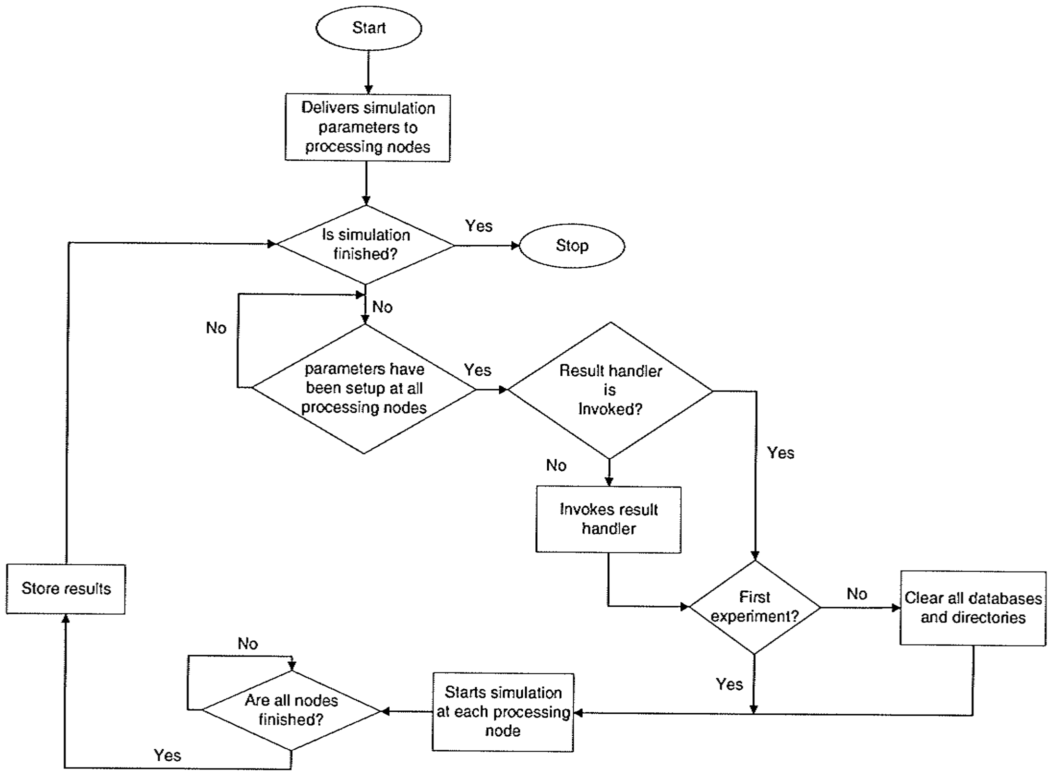

Start
Delivers simulation parameters to processing nodes
Is simulation finished?
Yes
Stop
No
No
parameters have been setup at all processing nodes
Yes
Result handler is Invoked?
Yes
No
Invokes result handler
Store results
First experiment?
No
Clear all databases and directories
No
Yes
Are all nodes finished?
Starts simulation at each processing node
Yes

Figure 4.7: A master scheduler process

the GUI to each processing node within the PLS. Once parameters have been setup at each node, it instructs the node scheduler at each node to invoke a group of clients (data consumers and/or data producers) according to the assigned parameters. It also collects results from each node scheduler.

4.2.2 Node Schedulers

Each processing node has a scheduler responsible for invoking a group of clients to simulate update and/or lookup requests to the node server based on the simulation parameters. A node scheduler returns the performance information to the master scheduler when all of its clients have finished their assigned tasks.

Chapter 5

Performance Study

This research focuses on studying the performance of different PLS configurations to reveal the advantages and disadvantages of different system designs. The end result of this study will be used to identify the best PLS configuration that is suitable for implementing a patient locating mechanism in a large-scale distributed hospital environment. Three PLS prototypes were implemented and tested under various loads to study their performance. The PLS performance characteristics are captured by varying the update and/or lookup requests (i.e. different number of data producers and consumers) to the system. The performance of each PLS configuration is measured in terms of two performance metrics: transaction response times and throughput. In this chapter, we first give the definitions of the two performance metrics. We then describe the performance study methodology. Finally, we present the experimental results of different PLS prototypes and analyze the results by comparing their performance in terms of transaction response times and throughput.

5.1 Performance Metrices

The performance of a PLS is measured in term of system transaction response time and throughput. These performance metrics are commonly used to measure the quality of a system in a complex environment where a mixture of read and write transactions are intensive [TPC02]. The transaction response time of a PLS is critical because it indicates the waiting time the client expects when requesting service from the PLS. In the hospital environment, the PLS must be able to locate the patient's data quickly because radiologists are often expecting medical images to be made available almost instantly and to be accessible at all times, wherever and whenever necessary to facilitate their diagnostic process. In addition to this requirement, the growing role of PACS in the healthcare domain suggests that the PLS must be able to handle large numbers of user requests from other components within the hospital network. For these reasons, the PLS, no matter if it is centralized or distributed, must be able to provide services to its clients with low response time and be capable of handling extensive load without substantial degradation in system throughput. The definitions of a transaction response time and throughput used in this thesis are given in the following sections.

5.1.1 Transaction Response Time

As discussed in Section 4.1.1.1 and 4.1.1.2, a client's request (update or lookup) involves sending a request to a node server who performs operations on its local DICOM database. In some cases, a node server acts as a client that requests service(s) from a directory server when the client's request involves an update or lookup operation on the directory. Upon receiving a request from a node server, a directory server performs operations on the directory and returns a reply to the node server. Finally, the node server processes the directory server's reply and then sends its reply back to the client. Thus, a request from a client (i.e. data consumer or producer) initiates a sequence of messages transmitting between a client and a node server and/or a directory server. The client's request also initiates one or more operations performed by a node server and/or a directory server.

The term transaction defined in this thesis is a request initiated by a data producer or a data consumer that generates one or more update and/or lookup operations on a database/directory. An update request issued by a data producer consists of updating the metadata of a patient record. The metadata consists of a PID, a set of study UIDs for the PID, and a set of series UIDs of each study UID. An update request from a data producer also initiates an update request from a node server to the directory server. The update request from a node server to the directory server includes updating the PID of a patient record and the location where the metadata of a PID has been stored. A lookup request issued by a data consumer includes one or more read operations performed on a database and/or directory. As illustrated in Figure 4.2, there are many types of lookup requests. These include retrieving a list of available PIDs (i.e. get-patients-list), get a list of study UIDs associated with a given PID (i.e. get-studies-list), get a list of series UIDs associated with a given study UID (i.e. get-series-list), and get a list of locations where the metadata of a PID can be found (i.e. get-hosts-list). Similar to the update request, a lookup request such as get-PIDs-list and get-hosts-list also initiates a lookup request sent from a node server to the directory server. Therefore, a reply from a node server to the client's request (i.e. update request, get-PIDs-list, or get-hosts-list) depends on the reply sent from a directory server. A *transaction response time* (TRT) is the amount of time required to perform a client's transaction.

The TRT indicates the time a client expects to wait when requesting a service from a server. Clients of a node server are data consumers and data producers. Clients of a directory server are node servers residing at different processing nodes. The TRT is similar to the transaction response time defined by Transaction Processing Performance Council *et al.* [TPC02] except that the TRT defined in this thesis encompasses two main parts: communication time (CT) and processing time (PT) per request. The reason is

Table 5.1: The definitions of transaction response time

Performance Metrics	Descriptions
Client/Node Server communication time (CSCT)	The amount of time spent on transmitting a request and response between a client and a node server.
Node Server/Directory communication time (SDCT)	The amount of time spent on transmitting a request and response between a node server and a directory server.
Node Server processing time (NSPT)	The amount of time a node server spent on performing update or query operations on local DICOM database. This time may include the amount of time spent on processing a response from a directory server when directory operation(s) is involved.
Directory Server processing time (DSPT)	The amount of time a directory server spent on performing update or lookup operation(s) on the directory.
Transaction response time - Node Server (TRT-NS)	Sum of SDCT and DSPT
Transaction response time - Client (TRT-C)	Sum of CSCT and NSPT or Sum of CSCT, NSPT, and TRT-NS

to identify the limiting factor(s) of a PLS. The CT is the time used for transmitting a request/respond between a client (i.e. a data consumer, a data producer, or a node server) and a server (i.e. a node server or a directory server). The PT is the time a server (i.e. a node server or a directory server) uses to perform update or lookup operations on the database/directory. These metrics are summarized in Table 5.1 and illustrated in Figure 5.1.

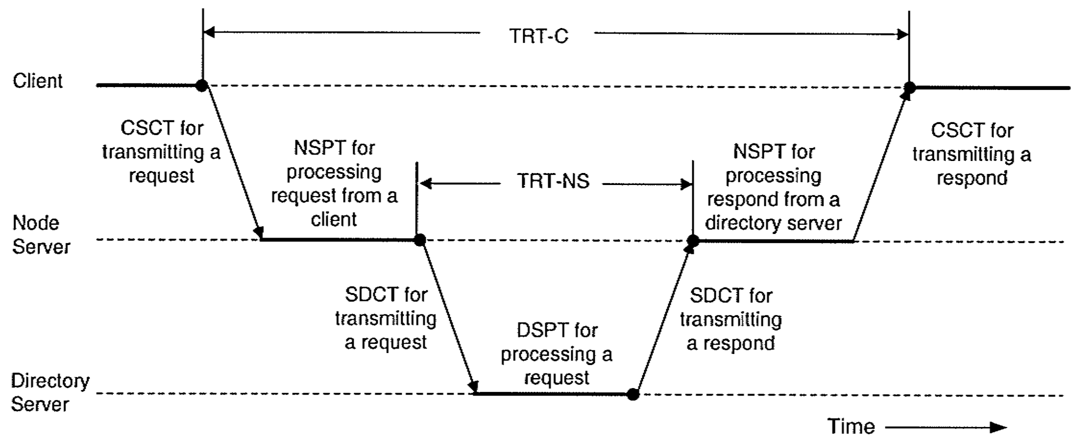

TRT-C
Client
CSCT for transmitting a request
NSPT for processing request from a client
TRT-NS
NSPT for processing respond from a directory server
CSCT for transmitting a respond
Node Server
SDCT for transmitting a request
DSPT for processing a request
SDCT for transmitting a respond
Directory Server
Time

Figure 5.1: The PLS transaction response times collected at different layers

5.1.2 Throughput

Figure 4.1 and 4.2 illustrate requests issued by each client (data consumer or data producer) in a well structured way that emulates the behaviour of radiologist or imaging modality in the healthcare domain. The request pattern of each client consists of sending a request to a node server followed by a sleep phase in a range between 5 and 15 seconds. The sleep phase incorporated between the completion of one request and the submission of the next takes into account the dynamic data access behaviour of a radiologist (i.e. data consumer) or the time spent on calibrating a modality (i.e. data producer) for each patient's examination. Instead of measuring a transaction execution rate, the throughput of a PLS measures how many data consumer and/or data producer routines can be handled by a PLS per hour with each imaging modality (or radiologist) has a request pattern similiar to the one shown in Figure 4.1 (or 4.2). Parham [Par00] states that "When examining PACS systems it is important to look closely at how fast the system under consideration allows the radiologists to step through the typical workflow process". As such, the throughput of a PLS is collected by measuring the total number of clients existing in a processing node of the PLS divided by the total elapsed time those clients spent on updating or looking up patient records.

5.2 Study Methodology

The PLS configurations under study are: centralized PLS architecture with a relational directory (CPLS-R), centralized PLS architecture with a LDAP-based (i.e. hierarchical) directory (CPLS-L), and distributed PLS architecture with a relational directory (DPLS-R). The difference between the CPLS-R and CPLS-L is the database model used for implementing the directory. The difference between the CPLS-R and DPLS-R is the way the directory service is implemented. This thesis compares the performance of different PLS designs from two perspectives: the architecture and the directory designs. Thus, PLS configurations are compared from two standpoints: (1) the same PLS architecture with different directory designs; and (2) different architectures with the same directory design.

5.2.1 Simulation Types

The TRTs and throughput of a PLS configuration were captured by a set of simulations. In the simulations, two parameters: the total number of participating data consumers and/or data producers that represent different lookup/update loads in a PLS are varied. For each simulation, the experiment is repeated 10 times. Each simulation is classified as one of the three simulation types. Simulation types are defined according to the database/directory operations and are described as follows:

- Lookup-only simulations (LO): simulations contain varying number of lookup operations in the PLS being tested. A simulation of this type does not consist of any update operation. Each simulation consists of the same number of data consumers invoked at each processing node existing in the PLS. Data consumers are used to simulate lookup requests on DICOM databases and directory/directories. The number of data consumers participated in a simulation is varied from one simulation to another simulation.

- Update-only simulations (UO): simulations contain varying number of update operations in the PLS being tested. A simulation of this type does not consist of any lookup operation. Each simulation consists of the same number of data producers invoked at each processing node existing in the PLS. Data producers are used to simulate update requests on DICOM databases and directory/directories. The number of data producers participated in a simulation is varied from one simulation to another simulation.
- Lookup and Update simulations (LU): simulations contain varying number of lookup and update operations in the PLS being tested. Each simulation consists of the same number of data consumers and producers at each processing node existing in the PLS. Data consumers and producers are used to simulate variable lookup and update requests on DICOM databases and directory/directories. The number of data consumers and producers participated in a simulation is varied from one simulation to another.

5.2.2 Hardware and Software Configurations

Three workstations (i.e. processing nodes) are used to implement and study the performance of PLS prototypes. The reason for using three processing nodes is to simulate typical work practices of radiology departments across three major hospitals in Winnipeg health region. Therefore, each PLS configuration is studied on three different nodes, which represent the three major hospitals in the Winnipeg health region. We anticipated that the results of the performance study can provide an estimate on the performance of the PLS for the entire Winnipeg health region. All processing nodes with approximately the same hardware and software configurations (see Table 5.2). Processing nodes are interconnected by means of a dedicated 100 Mbits/sec Ethernet to eliminate any "noise" that may be caused by other applications. Each processing node has VisiBroker® for Java (i.e. ORB product), IBM® DB2® EEE relational DBMS version 7.2, and a stand-alone LDAP server (i.e. Slapd).

Table 5.2: Hardware/software configurations of processing nodes for PLS simulations

Hardware/software	Node 1	Node 2	Node 3
Central processing unit (CPU)	Pentium III	Pentium III	Pentium III
CPU speed	500 MHz	450 MHz	600 MHz
Total physical memory (RAM)	192 MByte	256 MByte	192 MByte
Operating system	Red Hat Linux 7.0	Red Hat Linux 7.0	Red Hat Linux 7.0

One node server and the same number of clients are invoked at each processing node when a simulation is executed. At each processing node, clients and a node server are concurrently running on the same node. In addition, the ORB's directory service called "osagent" is also invoked at each processing node to locate, register, and un-register services (i.e. lookup, update, and replication services) provided from node servers and directory server(s)[1]. When a server (i.e. a node server or a directory server) starts; the server's ORB first locates an osagent through a *User Datagram Protocol* (UDP) broadcast

[1]VisiBroker® also provides a Naming Service for registering and locating object implementations

message[2] and then registers the server's services with the osagent. When a client makes a request, the client's ORB contacts the osagent to locate the desired service. According to VisiBroker® documentation, if more than one osagent running on different hosts (i.e. computers) in the client's local network, the first osagent response to the ORB's UDP message is used to obtain server's object references.

In the case of centralized PLS configurations (i.e., the CPLS-R and CPLS-L) only one directory server is invoked at one of the processing nodes. While, in DPLS-R configuration, each processing node has a directory server running on it. In CPLS-L configuration, the Slapd LDAP server is also running on a node where the directory server is invoked. Testing parameters for each simulation are defined by means of a simulation tool. The simulation tool occurs at one of the processing nodes to input a set of parameters for each simulation, deliver inputs to each processing node, and receive results from those nodes. Each processing node has a node scheduler responsible for receiving a set of simulation parameters assigned and distributed by the master scheduler.

5.2.3 Simulation Procedures

The procedures for setting up a simulation to collect the performance measures of the CPLS-R configuration is as follows: a node scheduler, a node server, and an osagent are started at each processing node (i.e. processing node 1, 2, and 3). A directory server and the simulation tool are started at processing node 3. When all components (excepts the clients) are started, a set of simulation parameters is entered into a simulation

[2]The UDP broadcast scheme is the default scheme for locating osagents in the network. An alternative approach can also be used to locate osagents. This includes configuring the IP address of an osagent as a runtime parameter, system's environment variable, or in a file.

tool according to the selected simulation type (i.e. LO, UO, or LU). The simulation parameters for each processing node is assigned and distributed by a master scheduler to each node scheduler. The tasks assigned to an individual client (a data consumer or data producer) are allocated based on a subset of simulation parameters assigned by its local node scheduler. When all parameters are set at each processing node, the master scheduler instructs each node scheduler to start a simulation.

When a node scheduler receives the master scheduler's instruction, it generates a set of clients by spawning threads according to the defined simulation type and parameters. In each request, a client captures the performance metrics (i.e. the TRT-C and throughput) when it has received the node server's response. A node server captures the TRTs-NS when making a request from a directory server. Each client sends a notification to the node scheduler when all of its tasks have finished. A client's notification contains the total elapsed time and a set of TRTs (i.e. a set of TRT-C and TRT-NS) collected when performing the assigned tasks. When a node scheduler received notifications from all of its local clients, it sends the average TRTs and throughput to the master scheduler. A new experiment starts when the master scheduler recorded the average TRTs and the throughput of the current experiment from all of the node schedulers. A simulation is stopped when all of the experiments have finished.

Figure 5.2 illustrates the interactions between the GUI, the master scheduler, node schedulers, node servers, and a directory server when experimented with the centralized PLS architecture. Figure 5.2 also indicates the locations where the performance metrics were collected during the simulation. This model is also used for testing the distributed PLS architecture except that a directory server is invoked at every processing node. The procedures for setting up a simulation to collect the performance metrics of the CPLS-L

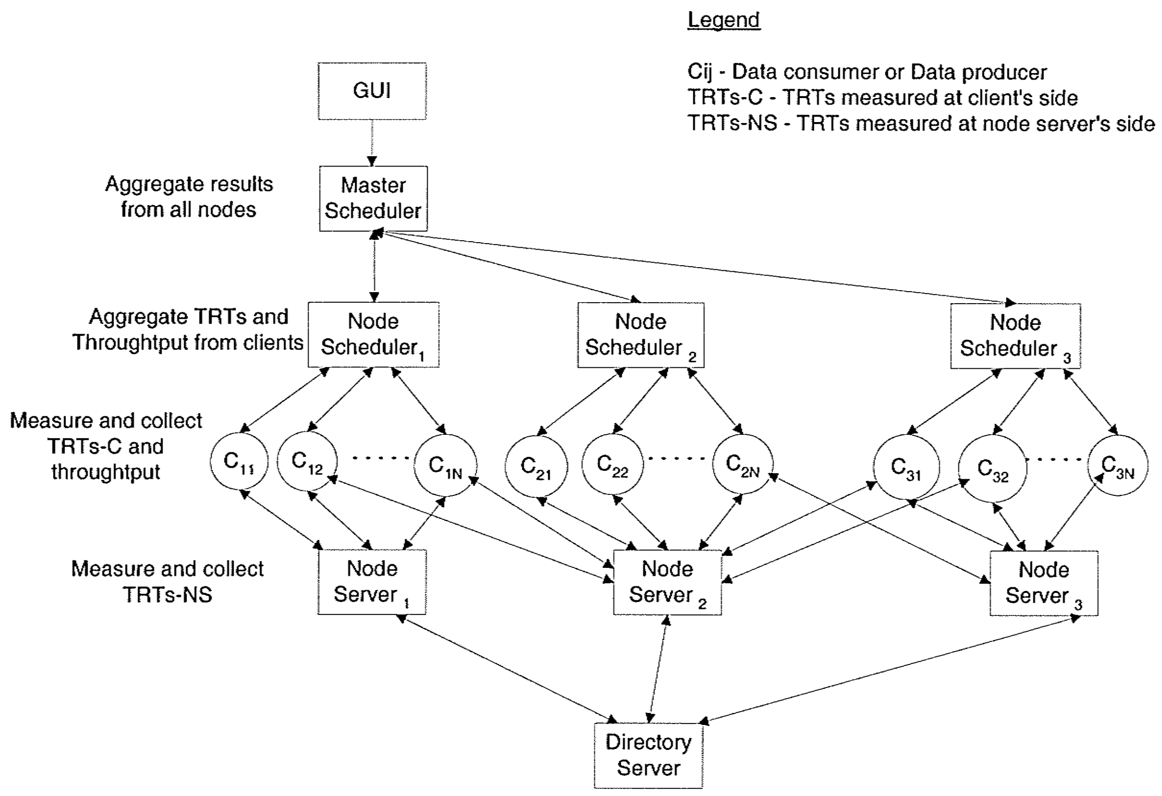

Legend
Cij - Data consumer or Data producer
TRTs-C - TRTs measured at client's side
TRTs-NS - TRTs measured at node server's side
GUI
Aggregate results from all nodes
Master Scheduler
Aggregate TRTs and Throughtput from clients
Node Scheduler 1
Node Scheduler 2
Node Scheduler 3
Measure and collect TRTs-C and throughtput
C11
C12
C1N
C21
C22
C2N
C31
C32
C3N
Measure and collect TRTs-NS
Node Server 1
Node Server 2
Node Server 3
Directory Server

Figure 5.2: A conception view of the simulation model when testing centralized PLS architecture.

configuration is similar to the CPLS-R except that the Slapd LDAP server is also invoked at processing node 3. The difference between the CPLS-R and DPLS-R is that a directory server is invoked at every processing node.

5.2.4 Assumptions

The main purpose of this study is to find the optimal PLS configuration by comparing the system performance, thus, all components at each processing node are assumed operational in the PLS being tested. Therefore, issue related to recovery from hardware/software failures and network partitioning is not considered in this study. In addition, clients (i.e. data consumers and producers) invoked at a specific node only contact its local node server when performing requests such as update patient records, get PIDs lists, and get host lists. In some cases, a data consumer will contact other remote node server(s) to get a list of study UIDs for a PID and a set of series UIDs for a specific study UID when the location(s) of the metadata is identified. Each PID is unique and used to identify a patient existing in the PLS. The total number of PIDs and the PIDs distributed among the processing nodes are fixed in all simulations. The portion of the total PIDs distributed to processing node 1 and 3 are the same, while processing node 2 has fewer PIDs compared to node 1 and 3. The reason is to take into account that some hospitals may have more patients than the others for various reasons.

5.2.5 Simulation Parameters

The dependent parameters of each simulation are TRTs and throughput. The independent parameter(s) for the LO type simulations is/are the number of data consumers invoked at each processing node. While the independent parameter for the UO type simulations is the number of data producers invoked at each processing node. Finally, the independent parameter for the LU simulations is the number of data consumers and producers invoked at each processing node. According to Mr. Sergio Camorlinga at SBRC, radiologists usually work in a group of 5 to 10 in the department and there are 3 to 5 imaging devices per modality type. For example, there are about 11 different types of imaging modalities currently installed at St. Boniface General Hospital. Therefore, the independent parameter (i.e. data consumer or producer) is varied in the range between 5 and 50 clients in this study.

Parameters defined for the LO type simulations are as follows:

- Number of PIDs consumed by each data consumer: 10 PIDs
- Number of study UIDs consumed per each PID: 5 study UIDs
- Number of series UIDs consumed per each study UID: 5 series UIDs
- Percentage of PIDs consumed at a single node (local): 90%
- Percentage of PIDs consumed at multiple nodes (remote): 10%

Parameters defined for the UO type simulations are as follows:

- Number of PIDs generated globally for each experiment: 100 PIDs
- Number of study UIDs associated with each PID: 10 study UIDs
- Number of series UIDs associated with each study UID: 10 series UIDs
- PIDs Distribution: R1=25%, R2=25%, R3=15%, R4=10%, R5=10%, R6=10%, R7=5% of the total PIDs

For each LO simulation, the percentage of local and remote requests defined for each data consumer is 90 and 10 percent, respectively. The values of these two parameters considered for the simulations are based on the statistics (i.e., radiologists accessing patients' data locally as well as globally) given by Dr. Blake McClarty at SBRC. Parameters defined for the LU type simulations are the combination of parameters defined for the LO and UO types. In addition, a set of PIDs and its distribution are predefined following the UO type simulations before the LO type simulations take place. In the simulations, patient distribution (i.e. PIDs) is fixed in each PLS configuration. When testing the DPLS-R configuration, the replication time is set to two minutes between replications.

5.3 Experimental Results and Analysis

All simulations are performed to capture the TRTs and the throughput of different PLS prototypes. As described in Section 5.1.1, a reply from a node server to a client's request such as to update a patient record, get-PIDs-list, or get-hosts-list is dependent on the reply sent from a directory server to a node server. Therefore, the performance of different PLS configurations (i.e. CPLS-R, CPLS-L, and DPLS-R) is compared based on the TRT-NS and throughput.

5.3.1 A Performance Study on CPLS-R and CPLS-L Configurations

This section presents the performance comparison between the CPLS-R and CPLS-L configurations. Both configurations used the centralized architecture but differ in the database model used for implementing the directory. The performance study of these two configurations is therefore based on their processing time (i.e. DSPT) for handling clients' lookup and/or update requests. Section 5.3.1.1 through 5.3.1.3 present the DSPTs of the two configurations collected during the LO, UO, and LU simulations. The throughputs of the two configurations are presented in Section 5.3.1.4.

5.3.1.1 Interactions Involve Lookup-only Operations

Figure 5.3 shows the DSPTs for the CPLS-R and CPLS-L configurations when systems only involve data consumer interactions. Recall that when a data consumer at a specific processing node looks up the location information of the metadata of a patient (i.e. PID), the node server issues lookup requests to the directory server. The DSPTs presented in Figure 5.3 measure the CPLS-R and CPLS-L configurations when handling data consumers' lookup requests on the locations of patients' metadata (i.e. get-hosts-list requests). The DSPTs of the two configurations are collected during the LO simulations. In the figure, each value on the horizontal axis corresponds to the number of concurrent data consumers invoked at each processing node. The vertical axis shows the average

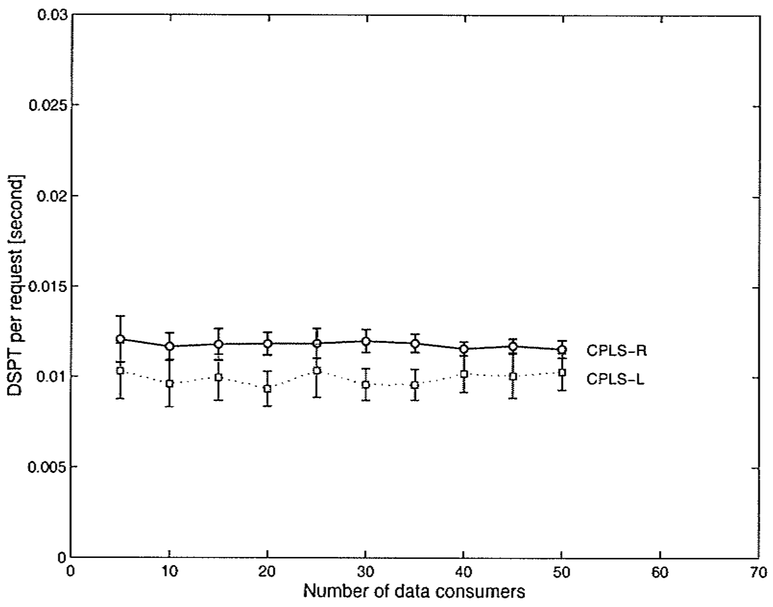


Figure 5.3: The DSPTs of the CPLS-R and CPLS-L for handling "get-hosts-list" requests in LO simulations

time[3] (measured in second) per request when a system handling data consumers' get-hosts-list requests. The vertical bars at each value in a figure represents the confidence interval that ranges from x-σ to x+σ.

The DSPTs shown in Figure 5.3 indicate the CPLS-L configuration offers a better performance than the CPLS-R when processing "get-hosts-list" lookup operations on the directory. Both configurations performed well and no significant performance degradation is observed as the number of concurrent data consumers increases in the system. In the range of 5 to 50 data consumers existing at each processing node, the mean DSPT of the CPLS-L is about 10 milliseconds per each get-hosts-list request with variance around 0.13 microseconds. In the same range, the mena DSPT of the CPLS-R is approximately

[3]Time is calculated by summing the times reported by all processing nodes divided by the total number of directory server existing in the system.

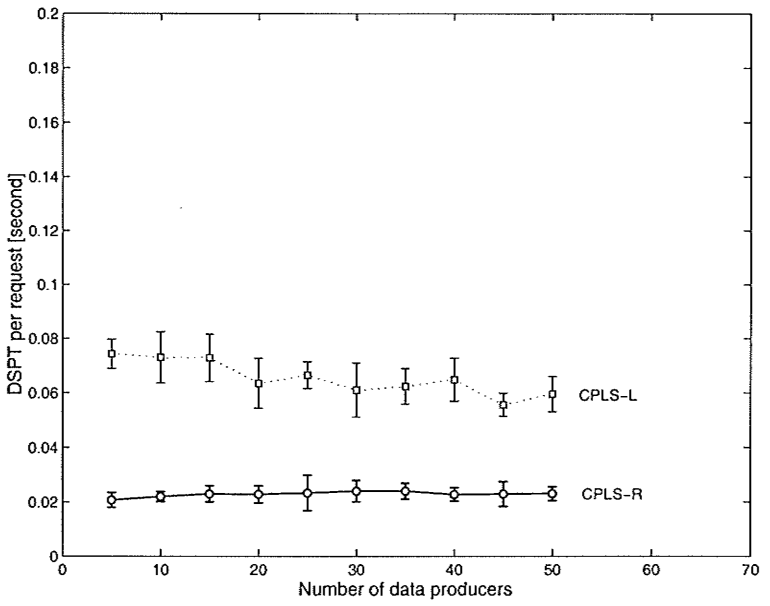


Figure 5.4: The DSPTs of the CPLS-R and CPLS-L for handling update requests in UO simulations

12 milliseconds per get-hosts-list request with variance around 0.03 microseconds. For 5 to 50 data consumers, the mean processing time of the CPLS-L is approximately 17% (per request) faster than the CPLS-R when searching data on the directory. Although this processing time difference seems small, the CPLS-L can perform 20% (i.e. 60000 get-hosts-list requests) more requests (per hour) than the CPLS-R provided all hardware and software configurations and system parameters remain the same. This result shows the hierarchical directory model offers an advantage on lookup operations over the relational directory model when a system only involves data consumer interactions.

5.3.1.2 Interactions Involve Update-only Operations

Figure 5.4 presents the DSPTs of the CPLS-R and CPLS-L configurations when the systems only involve data producer interactions. The DSPTs shown in Figure 5.4 measure the performance of the CPLS-R and CPLS-L when handling data producers' update requests (i.e. updates the metadata of patient records and the location information of those records). Recall that when a data producer updates the metadata of a patient (i.e. PID) at a specific processing node, the node server issues an update request to the directory server.

Figure 5.4 shows the performance of the two configurations diverges when performing updates to the directory. Figure 5.4 also shows that the performance of the CPLS-R is better than the CPLS-L when performing update operations to the directory. For the range of 5 to 50 data producers invoked at each processing node in the system, the mean DSPT of the CPLS-L and CPLS-R are about 65 and 23 milliseconds (per each update request), respectively. In this range, the overall DSPT of the CPLS-L is 2.8 times higher than the CPSL-R when updating data to the directory. This indicates the hierarchical directory model only offers an advantage on searching data but is poor on updating data.

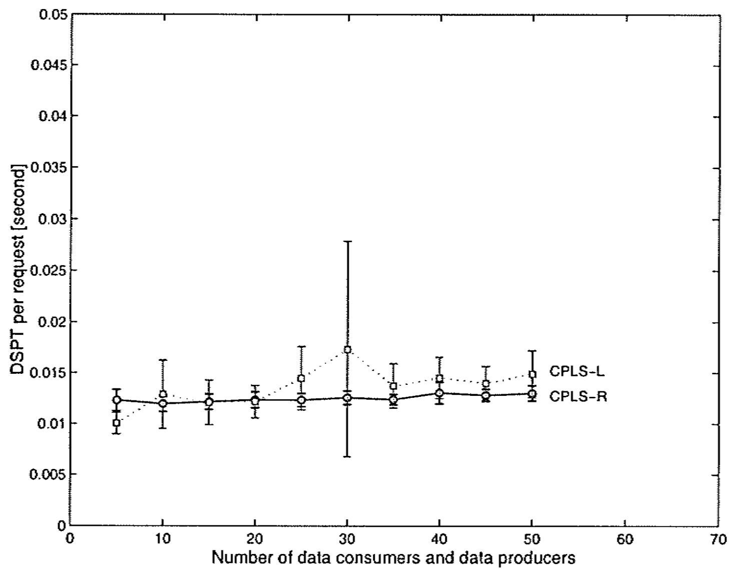
The reason this result occurred is the update method used in the CPLS-L is different from the one used in the CPLS-R. In both configurations, update to data (i.e. PID together with its location) is ignored if data already exists in the directory. In CPLS-L, updates to location information in the directory may involve modification operations if an entry (i.e. a PID) has already been inserted under the root entry and the data (i.e. the location) does not exist in that entry. In CPLS-R, a new row (i.e. a PID and its location) is inserted in the "Host" relational table if the PID together with the location information does not exist in the table. Therefore, the time difference between the two updates methods depend on the performance of the DB2® DBMS server and the OpenLDAP® Slapd server when performing updates to the directory. The reason for implementing the update method in different ways is the hierarchical data model supports multiple attributes in an entry, while the relational data model does not. This also indicates why the relational directory model requires more processing time on searching data than the hierarchical directory model. Thus, the processing time of a directory server depends on the size (i.e. number of entries) of the directory. Figure 5.4 also indicates the processing time needed for performing an update is more than the time needed for performing a lookup request. The implication of this result is that if the system has many more updates than lookups then the client of the CPLS-L may have to wait much longer time than the client of the CPLS-R configuration.

5.3.1.3 Interactions Involve Lookup and Update Operations

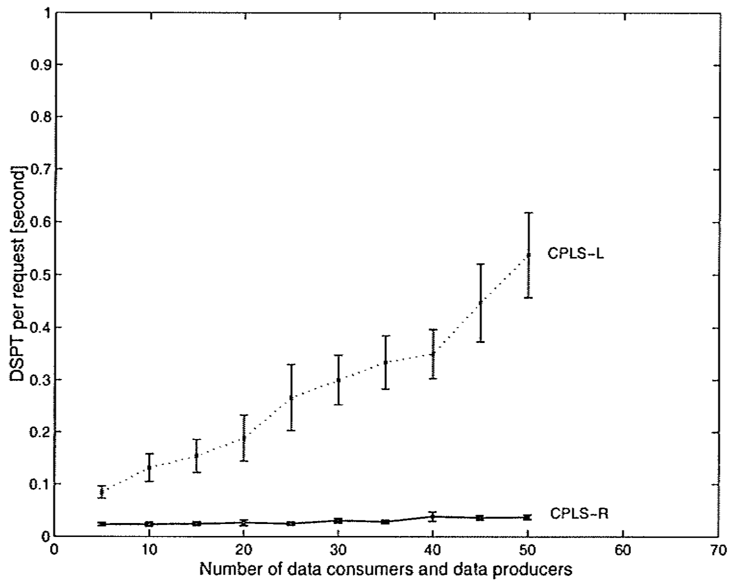
Figure 5.5(a) and 5.5(b) present the results of the CPLS-R and CPLS-L configurations collected during the LU simulations[4]. Figure 5.5(a) shows the DSPTs of the two configurations when handling lookup requests (i.e. get-hosts-list requests) in an environment where interactions involve both data consumers and producers. Figure 5.5(b) shows the DSPTs of the two configurations when handling data producers' update requests in the same environment.

Figure 5.5(a) shows the DSPTs (per get-hosts-list request) of the two configurations are approximately the same when a few clients (i.e. in the range of 10 to 20 clients per each type) participate in the system (e.g. approximately 12 milliseconds at 10). At 25 clients (per type), the DSPT of the CPLS-L is 17% (i.e. 2 milliseconds) higher than the DSPT of the CPLS-R when processing a get-hosts-list request. This result indicates the CPLS-L cannot maintain its performance advantage on lookup requests (see Figure 5.3) when a system involves both lookup and updates operations. When 5 to 50 clients (per

[4]In LU simulations, the horizontal axis in each figure indicates the same number of data consumers and producers are invoked at each processing node within a PLS. For example, a value of 50 indicates that 50 data consumers and 50 data producers are invoked at each processing node within a PLS.



(a)



(b)

Figure 5.5: The DSPTs of the CPLS-R and CPLS-L in LU simulations. (a) DSPTs for handling "get-hosts-list" lookup requests. (b) DSPTs for handling update requests.

type) participate at each processing node in the system, the overall mean DSPT (per get-hosts-list request) of the CPLS-R is approximately 12.5 milliseconds and the variance is around 0.14 microseconds. These results indicate the CPLS-R performs consistently well and maintains its performance on processing lookup requests even in a mixed environment.

Figure 5.5(b) shows the DSPTs of the two configurations when processing update requests in a mixed environment. The DSPT (per update request) of the CPLS-L increases with the number of concurrent clients. In the region of 5 to 40 data consumers and producers participating at each node in the system, the DSPT (per update request) of the CPLS-L increases about 314% (i.e. 265 milliseconds). At 50, the DSPT (per update request) of the CPLS-L is about 14 times higher than the CPLS-R. This indicates that a client of the CPLS-L in a mixed environment has to wait much longer time than the client of the CPLS-R when requesting update services from the directory server. This result clearly indicates the scalability problem of the CPLS-L configuration when handling both lookup and update operations in an environment where interactions involve data consumers and producers. By contrast, the CPLS-R performs consistently well and maintains its performance even in a mixed environment.

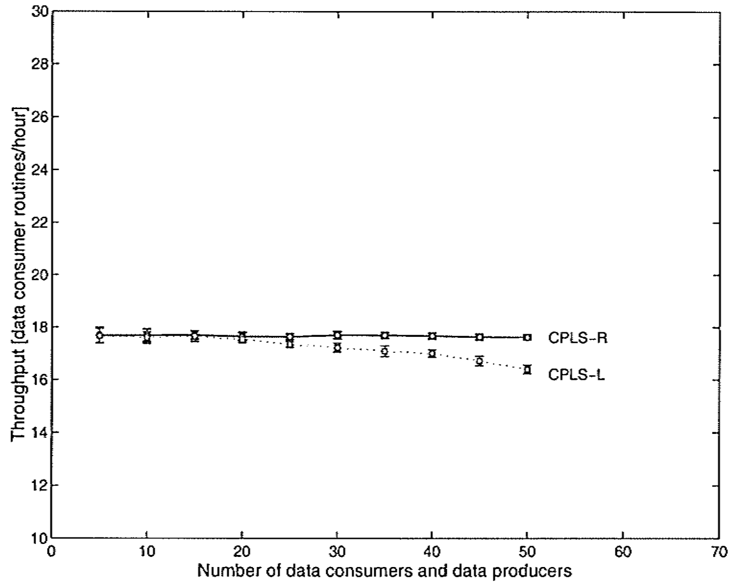
Figure 5.5(a) and 5.5(b) show the processing times of the CPLS-L in a mixed environment (i.e. involve both data consumers and data producers interactions) increases with the increasing number of concurrent clients, even though both systems have a centralized architecture, Figure 5.5(a) and 5.5(b) clearly indicate the performance differences between the two systems when using different data models for implementing the PLS directory. These results show a distinct advantage when using the relational directory data model for the PLS system. These results are very important because the PLS is meant for an environment where interactions involve both data consumers and producers.

5.3.1.4 The Throughput of CPLS-R and CPLS-L

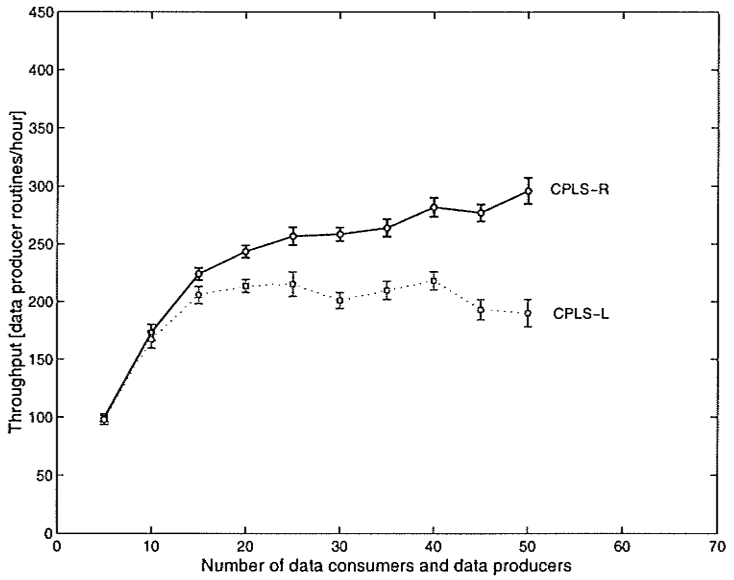
Figure 5.6(a) and 5.6(b) show the throughput of the CPLS-R and CPLS-L configurations captured during the LU simulations. The horizontal axis corresponds to the number of concurrent data consumers and producers invoked at each processing node. The vertical axis shows the throughput (i.e. data consumer or producer routines per hour) of both systems in a mixed environment. The throughput of a system is calculated by summing all the throughput reported from each processing node in the system.

The throughput reported by a node is measured by using the total number of clients (i.e. data consumers or producers) existing in a processing node divided by the total elapsed time of those clients spent on updating or looking up patient records. In this way, the total number of data consumer routines per hour of a PLS should maintain approximately the same level, if the system has maintained its performance. This is because the total task for each data consumer is approximately the same but differ in the "think" times chosen by each data consumer incorporated in its request routines. As described in Section 5.1.2, each client has a random sleep time in a range of 5 to 15 seconds between the completion of one request and the submission of the next. The sleep time chosen by each client is random, which may be different from one request to another request and/or one client to another client[5]. In addition to this, each data consumer randomly selects a PID from the list of available PIDs in the system before executes its request routine. Therefore, a data consumer may go to one or more nodes to obtain a list of

[5]Because the random sleep times selected by each client is varied so the throughput measured at one point may have a small difference at another point.



(a)



(b)

Figure 5.6: The throughput of the CPLS-R and CPLS-L in LU simulations. (a) Number of data consumer routines per hour. (b) Number of data producer routines per hour.

study UIDs and series UIDs, which depends on the chosen PIDs. Conversely, the total number of data producer routines per hour of a PLS should increase with the increasing number of data producers participated at each processing node. This occurs because the total PIDs distributed across the three nodes is fixed in each simulation so the total task assigned to each node is shared[6] among the total number of data producers existing in the node. Thus, the task assigned to a data producer becomes less as the total number of data producers increases at a node. The total number of data producer routines per hour will converge to a point when the task assigned to most of the data producers is approximately the same (i.e. one update request).

Figure 5.6(a) shows the throughput captured by a set of data consumers when executing lookup request routines in the CPLS-R and CPLS-L configurations. Figure 5.6(a) shows the highest throughput of the two configurations is obtained when a few clients (i.e. 5 data consumers and producers at each node) participated in the system and is around 18 data consumer routines per hour. However, the performance of CPLS-L starts to degrade as the number of clients (per type) increased in the system (e.g. starts degrade after 15). This result indicates the CPLS-L could have a potential scalability problem when a large number of lookup and update requests are involved in the system. At 50, the overall throughput of the CPLS-L is decreased about 7%. On the other hand, the overall throughput of the CPLS-R is maintained at an approximately the same level even when a large number of clients are added to the system. This result indicates the CPLS-R could maintain its quality of service and could scale well in an environment where lookups and updates are intensive. Figure 5.6(b) indicates the CPLS-R can handle many more data producer routines (per hour) than the CPLS-L. At 50, the CPLS-R can handle about

[6]The total task per node is not evenly assigned to the data producers existing in the node.

296 data producer routines per hour, which is about 55% higher than the CPLS-L in a mixed environment.

5.3.1.5 Summary of CPLS-R and CPLS-L Performance Study

The CPLS-R outperforms the CPLS-L configuration when an environment involves data consumer and producer interactions. The processing times shown in Figure 5.5 indicates the CPLS-R maintained its performance when operations involve both lookups and updates to the directory. The throughput shown in Figure 5.6 further indicates the CPLS-R could maintain its quality of service even in an environment where lookups and updates are intensive. With the same architecture, results shown in Figure 5.3 through 5.6 indicate that the relational directory model is more suitable than the hierarchical (i.e. LDAP-based tree structure) directory model for the implementation of the PLS directory.

5.3.2 A Performance Study on CPLS-R and DPLS-R Configurations

This section presents the performance comparison between the CPLS-R and DPLS-R configurations. Both configurations used the relational directory models to implementing the PLS directory but differ in the system architecture. Section 5.3.2.1 presents the performance comparison of these configurations when simulations were LO type (i.e. all clients in the PLS were data consumers). Section 5.3.2.2 shows the performance comparison of these configurations when simulations were UO type (i.e. all clients in the PLS were data producers). Section 5.3.2.3 presents the performance results of the two configurations, which are collected during the LU simulations (i.e. both data consumers and producers were participated in the PLS being test). The throughput of the two configurations is presented in Section 5.3.2.4.

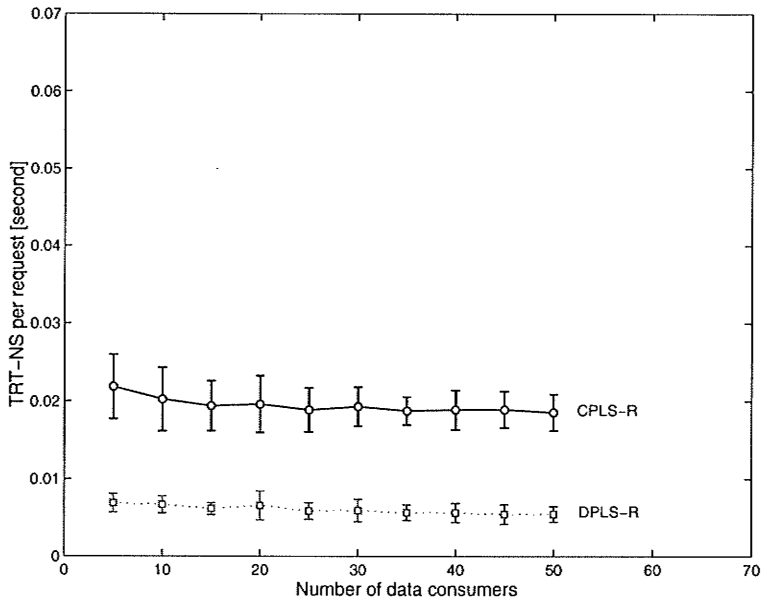


Figure 5.7: The TRTs-Ns of the CPLS-R and DPLS-R for handling "get-hosts-list" lookup requests in LO simulations

5.3.2.1 Interactions Involve Lookup-Only Operations

Figure 5.7 and Figure 5.8 show the performance results of the CPLS-R and DPLS-R configurations captured during the LO simulations[7]. The transaction response times (i.e. TRTs-NS) presented in Figure 5.7 shows the performance of the two configurations when performing lookup operations on the locations of patients' metadata (i.e. get-hosts-list requests). Figure 5.8 presents the DSPTs and SDCTs that constitute the TRTs-NS reported in Figure 5.7.

The TRTs-NS reported in Figure 5.7 indicate the DPLS-R can provide a significant faster response to its client's request compared to the CPLS-R. When 5 to 50 data consumers exist at each processing node, the mean TRT-NS (per directory server) of the

[7]Directory replication is not simulated in the LO simulations

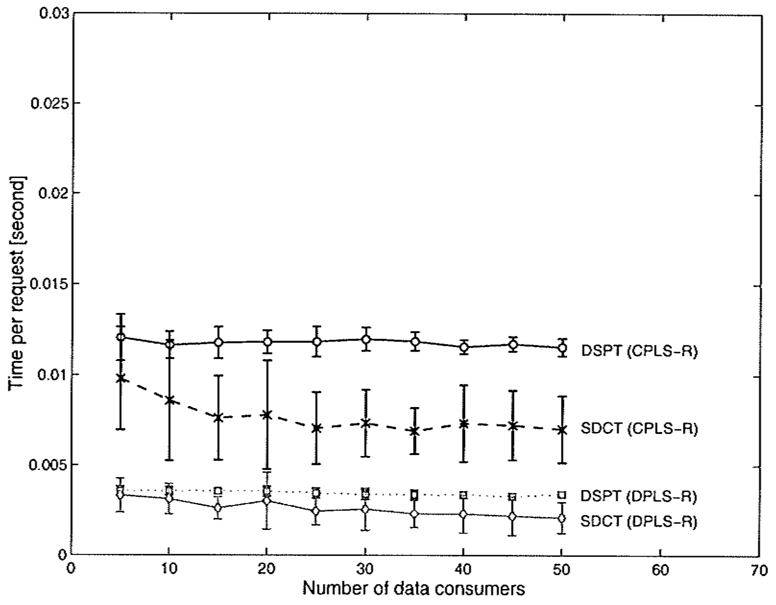


Figure 5.8: The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling "get-hosts-list" lookup requests in LO simulations

DPLS-R is about 6 milliseconds with variance around 0.3 microseconds when handling a get-hosts-list request. In the same range, the mean TRT-NS of the CPLS-R is around 19 milliseconds with variance approximately 1.0 microsecond per get-hosts-list request. This result shows that the directory server of the DPLS-R is about 3.2 times faster than the CPLS-R when performing location lookup operation. This result is expected because the directory service of the DPLS-R is shared among three directory servers. During the simulations, both configurations maintained their performance and no significant performance degradation is observed as the number of concurrent data consumers increases in the system.

Figure 5.8 presents the DSPTs and the SDCTs of the two configurations when processing "get-hosts-list" lookup operations on the directory. The DSPTs shown in Figure 5.8 indicate that both systems spend more time on directory processing than transmitting

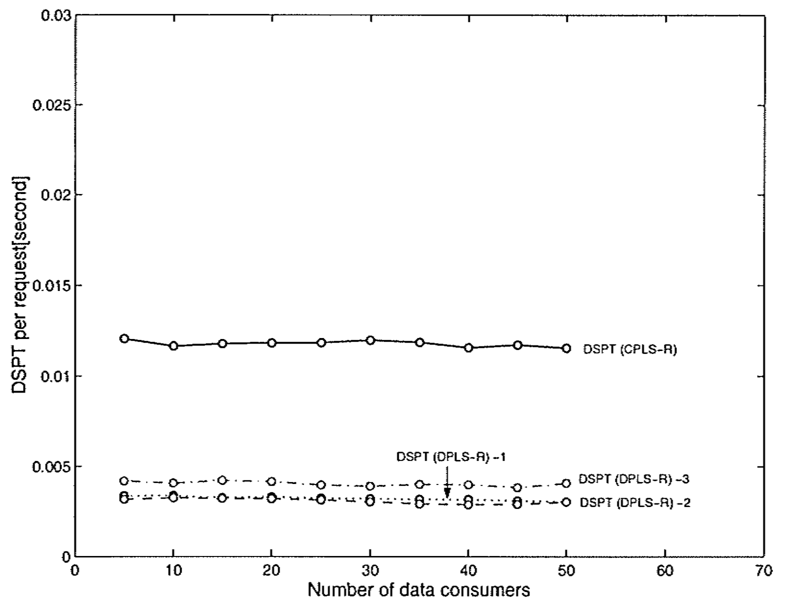


Figure 5.9: The DSPT reported by each processing node of the DPLS-R and the DSPT of CPLS-R configurations for handling "get-hosts-list" lookup requests in LO simulations

requests and responses between the node server and the directory server. In the DPLS-R configuration, the mean DSPT is approximately 33% higher than the mean SDCT of the system. However, the mean DSPT of the CPLS-R is about 54% higher than the mean SDCT of the system. Observe that the mean DSPT of the DPLS-R configuration is only about 2.4 times (e.g. less than 3 times) faster than the CPLS-R when performing lookup operations on the directory. The DSPT of the DPLS-R should at least 3 times faster than the CPLS-R when processing a lookup request. The reason is due to the simulator (i.e. the simulation GUI) being invoked at one of the processing nodes during the simulations for collecting results sent from other nodes. Figure 5.9 shows the DSPT reported by each node that constitutes the DSPT of the DPLS-R configuration[8]. Figure 5.9 shows the mean DSPT of processing node 3 is about 33% higher than the DSPT obtained from processing node 1. The mean DSPT of processing node 1 is about 4% higher than the

[8]For simplicity, confident interval is not shown in the figure.

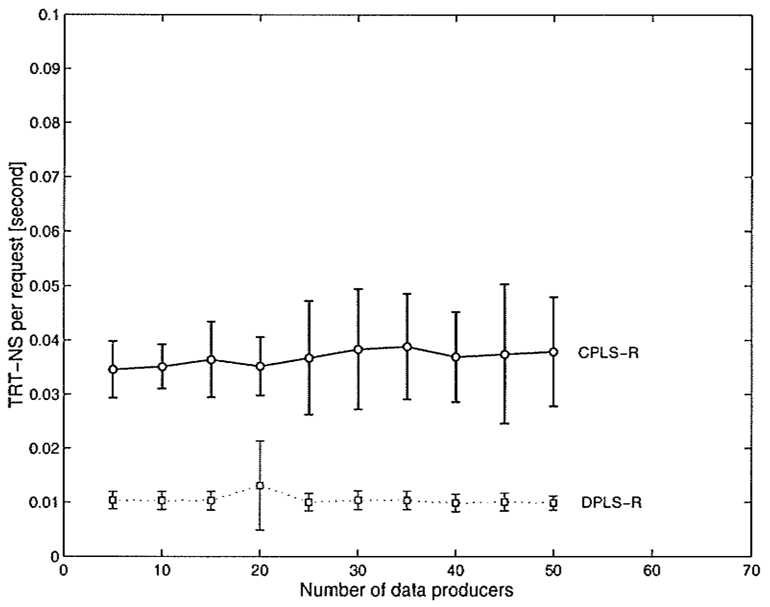


Figure 5.10: The TRTs-Ns of the CPLS-R and DPLS-R for handling update requests in UO simulations

DSPT obtained from processing node 2. These results indicate the simulator did affect the DSPT of the DPSL-R configuration.

5.3.2.2 Interactions Involve Update-Only Operations

This section reports on the performance study of the CPLS-R and DPLS-R configurations when the system only involves data producer interactions. In the DPLS-R, each directory server is also required to replicate its new updates to other directory servers every two minutes. Figure 5.10 shows the TRTs-NS of the two configurations when data producers update the metadata of patient records.

Again, the TRTs-NS shown in Figure 5.10 also demonstrates the advantage of the DPLS-R when the system only involves data producer interactions. When 5 to 50 data consumers exist at each processing node, the mean TRT-NS (per directory server) of the

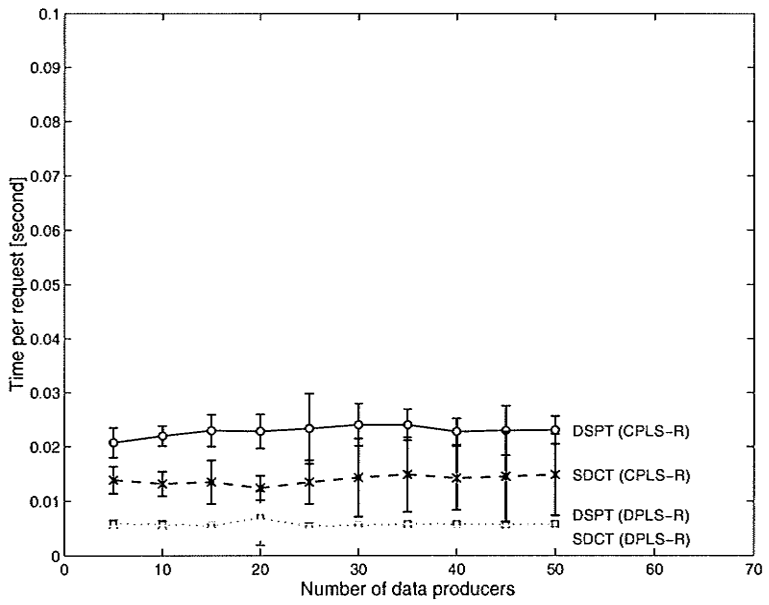


Figure 5.11: The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling update requests in UO simulations

DPLS-R is about 10 milliseconds with variance around 0.9 microseconds per update request. In the same range, the mean TRT-NS of the CPLS-R is about 37 milliseconds with variance approximately 2 microseconds. The performance gain of the DPSL-R system is 3.7 times faster than the CPLS-R when performing an update request.

Figure 5.11 presents the DSPTs and the SDCTs of the CPLS-R and DPLS-R configurations when processing update operations on the directory. It can be observed in Figure 5.11 that the DSPTs of the two systems are also higher than their corresponding SDCTs. In DPLS-R configuration, the mean DSPT of the system in the range of 5 to 50 data producers at each processing node is about 6 milliseconds with variance around 0.2 microseconds. The mean DSPT of the CPLS-R in the same range is approximately 23 milliseconds and the variance is around 0.9 microseconds per update request. Although each directory server in the DPLS-R has more responsibility (i.e. data replication) than the CPLS-R directory server, the mean DSPT of the DPLS-R configuration is about 3.8 times faster (per update request) than the CPLS-R when performing update operations on the directory. These results show that the DPLS-R performs better than the CPLS-R. Observe also that the DSPT of the CPLS-R is slightly increased with the number of data producers participate in the system. When comparing the DSPT at 50 with the DSPT at 5, the DSPT of the CPLS-R is increased by 11%. On the other hand, the DSPTs of the DPLS-R at these two values are approximately the same (i.e. about 6 milliseconds). Based on this observation, the DSPT of the CPLS-R would increase with additional clients. This could be further investigated by comparing the performance of the two systems in a mixed environment.

5.3.2.3 Interactions Involve Lookup and Update Operations

Figure 5.12 and 5.13 present the performance results of the CPLS-R and DPLS-R configurations when data consumers and producers simultaneously participate in the system. Figure 5.12 shows the DSPTs and SDCTs of the two systems when handling lookup requests (i.e. get-hosts-list requests). Figure 5.13 presents the DSPTs and the SDCTs of the two systems when handling data producers' update requests in the same environment. These results were collected during the LU simulations.

Figure 5.12 shows the DSPTs and SDCTs of the CPLS-R and DPLS-R configurations when handling "get-hosts-list" lookup requests in an environment where interactions involve data consumers and producers. The DSPTs of the two systems are higher than their SDCTs per lookup request. When 5 to 50 clients (per type) exist at each processing node, the mean DSPT (per directory server) of the DPLS-R is about 3.7 milliseconds with variance around 0.07 microseconds per get-hosts-list request. In the same range,

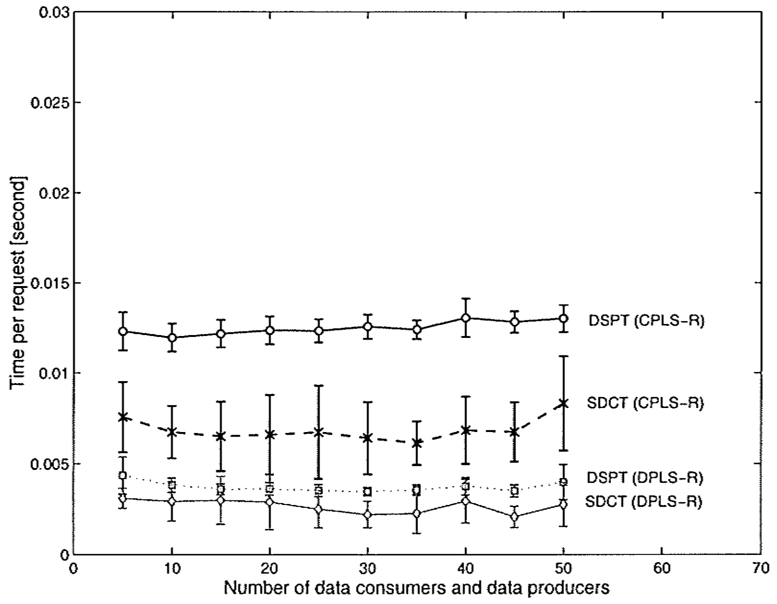


Figure 5.12: The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling "get-hosts-list" lookup requests in LU simulations

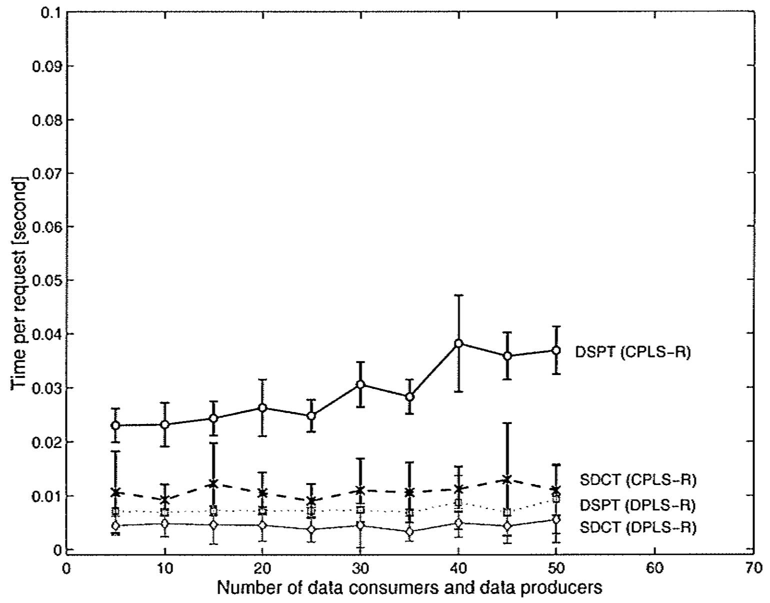


Figure 5.13: The DSPTs and SDCTs of the CPLS-R and DPLS-R for handling update requests in LU simulations

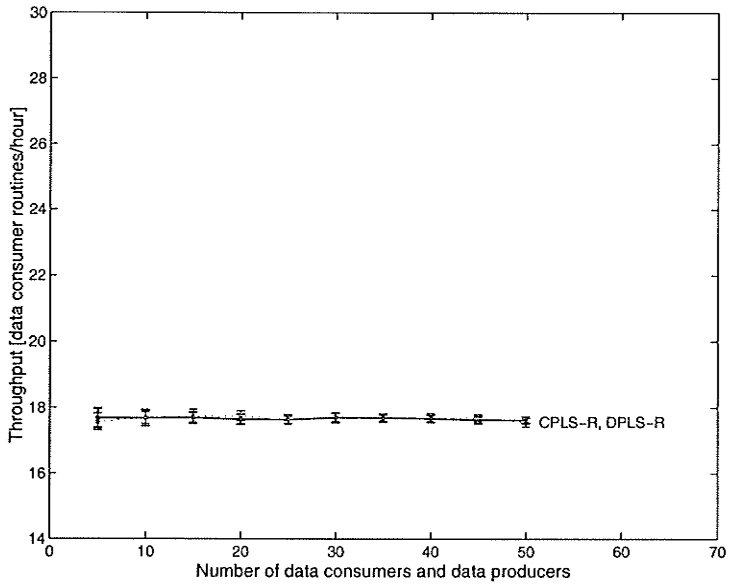
the mean DSPT of the CPLS-R is about 12.5 milliseconds with variance around 0.13 microseconds. The DSPT performance gain of the DPLS-R system is 3.4 times faster than the CPLS-R when performing a get-hosts-list request in a mixed environment. Both CPLS-R and DPLS-R perform consistently well and no performance degradation is observed during the LU simulations. When comparing the mean DSPT per get-hosts-list request obtained from the LO simulations with the mean DSPT obtained from the LU simulations, both systems maintain their performance even in a mixed environment (i.e. the mean DSPT of the DPLS-R is about 3.4 milliseconds and the mean DSPT of the CPLS-R is about 11.7 milliseconds during the LO simulations).

However, we observed that the DSPT of the CPLS-R per update request shown in Figure 5.13 is noticeably changed (i.e. from 20 to 50) as the number of concurrent clients participate in the system. In the range of 5 to 50 clients (per type at each processing node) participated in the system, the mean DSPT of the CPLS-R is about 29 milliseconds with variance around 34 microseconds per update request in a mixed environment. By contrast, the DPLS-R performs consistently well and maintains its performance even when the number of concurrent clients increases in the system. The mean DSPT of the DPLS-R is about 8 milliseconds with variance around 0.7 microseconds per update request. At point 25, the DSPT of the CPLS-R is about 25 milliseconds, which is approximately 3.4 times higher than the value obtained in the DPLS-R (i.e. the DSPT of the DPLS-R is about 7.2 milliseconds at 25). When the number of concurrent clients participating in each processing node increased from 25 to 50, the DSPT of the CPLS-R is increased about 48.7% (i.e. increased 12 milliseconds), while the DSPT of the DPLS-R is increased about 29% (i.e. increased 2 milliseconds) per update request. This result indicates the DSPT of the two systems increases as the number of clients increases in the system. However, the increasing rate of change of the DSPT in the CPLS-R is much faster than the DPLS-R per update request. Although the DSPT of the DPLS-R increases as the number of clients increases in the system, the change is small compared to the CPLS-R. The DSPT of the DPLS-R increases because of the need to support data replication, so more processing time is required. This result shows that the CPLS-L would potentially face the scalability problem with increasing number of clients in the system. This also confirms the observation from the last section (Section 5.3.2.2) when discussing the performance results of the two systems collected from the UO simulation.

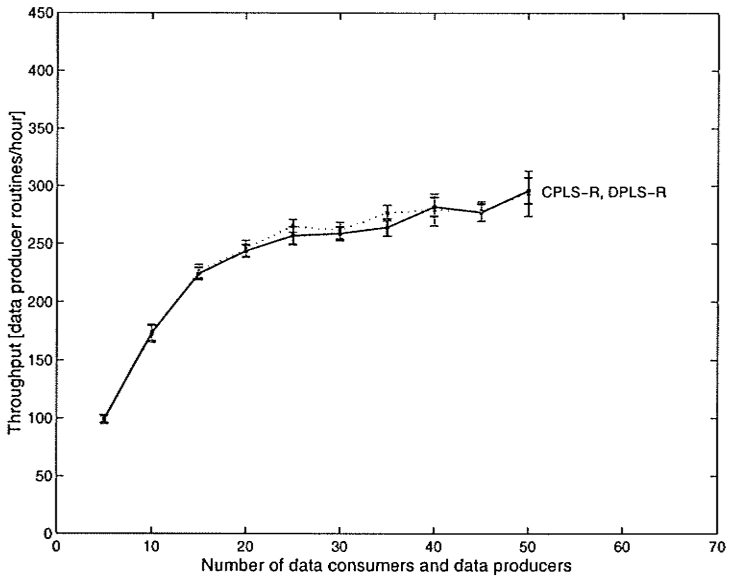
5.3.2.4 The Throughput of CPLS-R and DPLS-R

Figure 5.14(a) and 5.14(b) illustrates the throughput (i.e. data consumer or producer routines per hour) of the CPLS-R and DPLS-R configurations captured during the LU simulations. Again, the total number of data consumer routines per hour of a PLS should maintain approximately the same level if the system has maintained its performance during the simulations. The total number of data producer routines per hour of a PLS should increase as the number of data producers participating in each processing node increases. As shown in Figure 5.14(a) and 5.14(b), the throughput of the two systems are approximately the same. In the DPLS-R configuration, the node server layer is also implemented using a centralized approach and only the directory service is distributed so the overall throughput of a PLS also depends on the node server components of the system. It should be noted that the tasks performed by each node server are more than the tasks performed by each directory server at a node. As such, the performance gain of the DPLS-R obtained from the directory server layer is not noticeable when measuring the throughput of the system.

Figure 5.14(a) shows the throughput captured by a set of data consumers when exe-



(a)



(b)

Figure 5.14: The throughput of the CPLS-R and DPLS-R in LU simulations. (a) Number of data consumer routines per hour. (b) Number of data producer routines per hour.

cuting their lookup request routines in the CPLS-R and DPLS-R configurations. As shown in Figure 5.14(a), the overall throughput of the two configurations is about 18 data consumer routines per hour. Figure 5.14(b) shows the throughput captured by a set of data producers when executing their update request routines in the CPLS-R and DPLS-R configurations. The throughput of the two systems increased as the number of data producers participating in each node increase. The results show that both configurations perform consistently well without significant performance degradation is observed.

5.3.2.5 Summary of CPLS-R and DPLS-R Performance Study

The DPLS-R outperforms the CPLS-R configuration when measuring the transaction response time at the node server layer. In the layer between the node server and the directory server, the overall performance gain of the DPLS-R is at least 3 times faster than the CPLS-R when processing lookup and update requests. The results shown in Figure 5.13 indicate the processing time of the two systems increase as the number of clients increases in the system. However, the rate of change of the DSPT in the CPLS-R is much faster than the DPLS-R per update request, so the CPLS-R would face the scalability problem in a mixed environment when the total number of data consumers and producers increases in the system. The DPLS-R in the directory server layer is performed consistently well even through each directory server needs to support data replication. The throughput of the CPLS-R and the DPLS-R are approximately the same because the node server layer of the two systems is the same. The overall throughput of the DPLS-R would be higher than the CPLS-R if more than one node server is invoked at each node for processing clients' requests.

Chapter 6

Conclusions and Future Directions

The introduction of a PACS is to provide a better approach than the traditional film-based approach to acquisition, storage, delivery, retrieval, and management of patient medical images and data throughout the hospital. From the advantages provided by PACS, it has become increasingly important in the healthcare enterprise. PACS also directly and indirectly facilitates the workflow of various hospital components (e.g. radiology information system, health information system, *etc.*) in a number of different ways. Today, many hospitals make use of a PACS (or several PACSs) aimed to improve the efficiency of healthcare delivery as well as reduce the hospital operational costs due to the inefficient film-based management process. Although each of these PACSs worked effectively for each of the radiology departments and hospitals, patient medical data is not integrated with other PACSs residing at different departments of a hospital and/or remote hospitals. Patients often utilize the services of several hospitals so patient medical images and related data must be integrated and accessed significantly faster than the independent PACSs when needed. In response to this necessity, HDAS, a distributed computing system, is proposed and currently being developed at the SBRC. The main objective of the HDAS is to provide a virtual layer for the efficient integration, transmission, storage, and retrieval of medical images and patient data. In particular, data generated and stored at different PACSs residing at different hospitals inside or outside of a specific health region. Several challenges have been identified when developing the HDAS. These challenges include integrating and managing different PACSs across several hospitals and/or health regions, effectively locating and identifying the locations of medical data that might have been stored at multiple PACSs when needed, protecting data from unauthorized users, among others.

This research aims at providing a high performance patient data locating mechanism, PLS, to support the delivery of HDAS services. The PLS is a critical component of the HDAS. It provides information that is necessary and subsequently used by other components of HDAS in respond to user requests in a timely fashion. The core components of the PLS are its directory and directory services. The PLS directory is designed to store and manage the location information of patient's medical images and related data. The PLS directory service is designed to provide update and lookup operations on the directory.

In this thesis, three PLS configurations have been designed, prototyped, and tested to find the best configuration among them. These PLS configurations either differs in the database model (i.e. relational or hierarchical tree structure) used for implementing the directory or the architecture (i.e. centralized or distributed) used for providing the directory service. The PLS configurations have been designed, prototyped and tested in this thesis including the centralized PLS architecture with a relational directory (CPLS-R), the centralized PLS architecture with a LDAP-based (i.e. hierarchical tree structure) directory (CPLS-L), and the distributed PLS architecture with a relational directory (DPLS-R). This aim is to find the best PLS configuration among these three configurations by studying their performance in different simulated environment. A PLS performance is measured through system transaction response times and throughput collected from different simulating environment. Transaction response time indicates the time a user expects when requesting services from the HDAS component that uses the PLS services. It measures the time a PLS spent on transmitting and performing a user's request in a simulating environment. A PLS throughput measures how many routines per hour the PLS can handle in a simulating environment.

From one set of results obtained during various simulations, it is observed that for the same architecture (i.e. centralized architecture), the performance of a PLS is different when using relational or hierarchical tree database models to implement the PLS directory. The simulation results show that the PLS using the relational directory model (i.e. CPLS-R) outperforms the one using the LDAP-based directory model (i.e. CPLS-L) when an environment involve both update and lookup interactions. In the same environment, results also indicate the CPLS-L configuration has a significant performance degradation as the number of users increased in the system. This indicates the LDAP-based directory model has a scalability problem when using the same system architecture in an environment where both update and lookup operations are intensive. On the other hand, the CPLS-R configuration performed consistently well and maintained its performance even when the environment invole large number of lookup and update operations. Based on this observation, we conclude that the relational directory model is more suitable than the hierarchical directory model for the implementation of the PLS directory. However, a marked performance gain in the system transaction response time is observed when the relational directory model is applied to the distributed architecture (i.e. DPLS-R). The overall performance (i.e. TRTs measured at the node server layer) of the DPLS-R is better than the CPLS-R when processing lookup and update requests in any simulated environment. Therefore, the distributed architecture with a relational directory model is the best PLS configuration among all PLS configurations studied in this thesis. The performance study presented is important as the PLS is meant to be used by the HDAS to make patient data timely even when it might have been stored at different PACSs residing at remote hospitals. The PLS performance is important because accessing images in a timely way is a major concern to healthcare professionals that utilize the HDAS services. This thesis can also be used as a building block to develop a suitable database model for the storage and management of DICOM images and related patient data.

6.1 Ideas for Future Research

This work can be extended and further investigated in several ways. Some of the possible directions envisioned are listed here:

- In this thesis, each simulation is repreated 10 times. The reason for choosing 10 repeated experiments is to evaluate various PLS configurations as well as the methodology used to perform the PLS performance study. We anticipated that a further study is required to insure the degree of confidence of each experiment. In addition, the independent parameter (i.e. data consumers/producers) chosen for the PLS performance study is varied between 5 and 50 to simulate three major hospitals in Winnipeg health region. We expected that results could be similar to those presented in this thesis even with additional processing nodes and clients. In the future, a performance analysis can be conducted to evaluate the distributed PLS and validate the results presented when it is implemented and deployed in the regional healthcare data network.

- In the distributed PLS, we have used the push-based and lazy replication approach to propagate changes made on the data in each directory to other directories. This is accomplished to ensure replicated directories are globally consistent across the PLS. Gray [Gra96] describes a lazy replication approach that may cause global serialization and instability problems to the system. This is because changes on replicas are not immediately update at each replica node. Therefore, it may lead to or even further diverge the directory of each node from the other nodes if failure occur when updates are propagating through the network or when performing update operations. In addition, each node propagates (pushes) its update buffer to the other nodes almost simultaneously during the simulations. This update algorithm may lead to network congestion as the number of processing nodes grow. Therefore, an efficient data synchronization algorithm and replication algorithm are required to further research in the future and then apply to the distributed PLS. Once a new update algorithm and/or replication algorithm applied to the system, a performance study can also be conducted to compare the performance with the current system.

- In this thesis, the emphasis is on finding the best PLS configuration among all configurations by measuring their performance under a good operation environment (i.e. without any hardware/software or network partitioning occur) so issues related to recovery from failure are not addressed. However, hardware/software or network partitioning failure is uncommon in our daily practice so an intelligent recovery component is required to automatically recover from failures without seriously affecting the overall system performance. Hence, the PLS should maintain its performance to some extent even if failure occurs in some components of the system. Recovery from faults is an important issue in any distributed system so it requires further research in the future.

- Finally, an interceptor component could be added to the distributed PLS to monitor a directory server's activities at each processing site. When a directory server at a specific processing node is overloaded (i.e. beyond its workload threshold), the interceptor at that processing node can delivery some load to other idle directory servers to achieve load balancing across the distributed PLS.

References

[Bbv95] A. Beitz, M. Bearman, and A. Vogel, "Service Location in an Open Distributed Environment," *2nd International Workshop on Services in Distributed and Network Environment*, June 1995, pp. 28–40.

[CB01] S. Camorlinga and K. Barker, "HDAS: Health Distributed Archiving System," Technical Report: MITR200101A, St. Boniface General Hospital Research Centre, Winnipeg, MB, Canada, May 2001.

[Che01] E. Cheung, S. Camorlinga, K. Barker, and J. A. Rueda, "A Preliminary HDAS Global Database Architecture Designs," Technical Report: MITR200112A, St. Boniface General Hospital Research Centre, Winnipeg, MB, Canada, December 2001.

[Che02] E. Cheung, S. Camorlinga, K. Barker, and J. A. Rueda, "Distributed Technologies Analysis," Technical Report: MITR200203A, St. Boniface General Hospital Research Centre, Winnipeg, MB, Canada, March 2002.

[Fit97] S. Fitzgerald, I. Foster, C. Kesselman, G. V. Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations," *Proceedings of the 6th IEEE Symposium on High Performance Distributed Computing*, IEEE Computer Society Press, 1997, pp. 365–375.

[For96] D. W. Forslund, R. L. Phillips, D. G. Kilman, and J. L. Cook, "Experiences with a Distributed Virtual patient Record System," *Proceedings of the 1996 American Medical Informatics Association Annual Fall Symposium (Washington, D.C.)*, (J. Cimino, Ed. American Medical Inforamtics Assocf., Hanley & belfus), October 1996, pp. 483–487.

[For98] D. W. Forslund, J. E. George, E. M. Gavrilov, and T. E. Weymouth, "Distributed Telemedicine Using High-Performance Computing Over the Internet," High Performance Distributed Computing, 1998. (Available at: http://tmed.openemed.net/OpenEMed/TeleMed).

[Gra96] J. Gary, P. Helland, P. O'Neil, and D. Shasha, "The Dangers of Replication and a Solution," *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, June 1996, pp. 173–182.

[Gsm00] M. Grammatikou, F. Stamatelopoulos, and B. Maglaris, "Distributed Information System Architecture For Primary Health Care," *Proceedings of the MIE '2000 Conference*, (Germany), 2000.

[How95] T. A. Howes, "The Lightweight Directory Access Protocol: X.500 Lite," Center for Information Technology Integration, University of Michigan, July 1995.

[How96] T. A. Howes, "An X.500 and LDAP Database: Design and Implementation," 1996. (Available at: http://www.umich.edu/ dirsvcs/ldap/doc).

[Hua99] H. K. Huang, *PACS:Basic Principles and Applications*, Wiley-Liss, Inc., 605 Third Avenue, New York, USA, 1999.

[Inp00] *VisiBroker for Java – Programmer's Guide*, Inprise Corporation, 100 Enterprise Way Scotts Valley, CA 95066-3249, 2000.

[Kil97] D. G. Kilman and D. W. Forslund, "An International Collaborator Based on Virtual Patient Records," *Communications of the ACM*, Vol. 40, No. 8, August 1997, pp. 110–117.

[LS01] R. Lee and S. Seligman, *JNDI API Tutorial and Reference: Building Directory-Enabled Java Applications*, Addison-Wesley Publication Company, MA, USA, June 2000.

[Lei97] E. Leisch, S. Sartzetakis, M. Tsiknakis, and S.C. Orphanoudakis, "A Framework for the Integration of Distributed Autonomous Healthcare Information Systems," *Medical Informatics, Special Issues*, Vol. 22, No. 4, 1997, pp. 325–335.

[Lun99] N. Lundberg, "Impacts of PACS on Radiological Work," *Proceeding of the International ACM SIGGROUP Conference on Supporting Group Work*, 1999, pp. 168–178.

[Mor02] L. Moreau, "A Fault-Tolerant Directory Service for Mobile Agents Based on Forwarding Pointers," *Proceedings of the 17th ACM Symposium on Applied Computing (SAC'2002) – Track on Agents, Interactions, Mobility and Systems*, 2002, p. 93.

[Nov03] "Novell eDirectory Detailed View," Novell, Inc., 2003. (Available at: http://www.novell.com/products/edirectory/details.html).

[OV91] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems, Second Edition*, Prentice Hall, Inc., Upper Saddle River, New Jersey, USA, 1991.

[Oag02] The OpenLDAP Foundation, Redwood City, California, USA, *OpenLDAP 2.1 Administrator's Guide*, 2002.

[Ord93] J. Ordille and B.Miller, "Distributed Active Catalogs and Meta-Data Caching in Descriptive Name Services," *Processding of the 13th International Conference on Distributed Computing Systems*, 1993, pp. 120–129.

[Par00] J. E. Parham, "Find the Best PACS for Your Needs," CR & PACS Insights & Images, Spring 2000.

[Sai01] Y. Saito, "Consistency Management in Optimistic Replication Algorithms," June 2001. (Avaialble at: http://www.hpl.hp.com/personal).

[Shi00] S.S.B. Shi, E. Stokes, D. Byrne, C.F. Corn, D. Bachmann, and T. Jones, "An Enterprise Directory Solution with DB2," *IBM System Journal*, Vol. 39, No. 2, 2000, pp. 360–382.

[Sun95] "White Paper: Java Remote Method Invocation – Distributed Computing for Java," Sun Microsystems, Inc., 1995-2003. (Available at: http://java.sun.com/marketing/collateral/javarmi.html).

[Sut97] D. Sutherland, "RMI and Java Distributed Computing," JavaSoft, A Business Unit of Sun Microsystems, Inc., November 1997.

[TPC02] "TPC Benchmark C: Standard Specification, Revision 5.1," Transaction Processing Performance Council (TPC), December 2002.

[Tan02] A. S. Tanenbaum and M. V. Steen, *Distributed Systems Principles and Paradigms*, Prentice Hall, Inc., Upper Saddle River, New Jersey, USA, 2002.

[Tra94] G. Trayser, "Interactive System for Image Selection," Digital Imaging unit, Center of Medical Informatics, University Hospital of Geneva, 1994. (Available at: http://www.expasy.org/UIN/html1/projects/isis/isis.html).

[Tsi95] M. Tsiknakis, C. Chronaki, S. Kostomanolakis, and S.C. Orphanoudakis, "The Regional Health Telematics System of Crete," *Proceedings of the Health Telematics '95 Conference*, (Naples, Italy), July 1995, pp. 553–558.

[WL01] T. Wendler and C. Loef, "Workflow Management - Intergration Technology for Efficient Radiology," *Medica Mundi*, Vol. 45, No. 4, November 2001, pp. 41–48.

[Wal98] J. Waldo, "Remote Procedure Calls and Java Remote Method Invocation," *IEEE Concurrency*, July–September 1998, pp. 5–7.

[Wie00] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding Replication in Databases and Distributed Systems," *Proceedings of 20th International Conference on Distributed Computing Systems(ICDCS'2000)*, (Taipei, Taiwan, R.O.C.), IEEE Computer Society Technical Committee on Distributed Processing, 2000, pp. 264–274.

[Zis98] A. Zisman, *Information Discovery for Interoperable Autonomous Database Systems*, PhD thesis, Imperial College of Science, Technology and Medicine, University of London, London, UK, 1998.