

**SRE: Search and Retrieval Engine of the TerraScope Database Middleware
System**

By
Enna Z. Coronado-Pacora

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

MASTER OF SCIENCE
in
COMPUTER ENGINEERING
(Software Engineering)

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2003

Approved by:

Bienvenido Velez-Rivera, Ph.D.
Member, Graduate Committee

Date

Pedro Rivera-Vega, Ph.D.
Member, Graduate Committee

Date

Manuel Rodriguez-Martinez, Ph.D.
Chairperson, Graduate Committee

Date

Freya Toledo, M.S.
Representative of Graduate Studies

Date

Jose L. Cruz, Ph. D.
Chairperson of the Department

Date

Abstract

Our research emerged from the need to know how to recover and to visualize graphical/textual information of images that are stored in multiple data sources. We are focused on a system for recovering, merging, and displaying data, metadata, and images from heterogeneous distributed data sources for purposes of providing end-users with an unique entry point to those data sources. We present the *Search and Retrieval Engine* (SRE) developed as part of TerraScope, a Web-based Earth Science Database Middleware System. *SRE* is the execution engine based on Java Servlet technology and the Peer-to-Peer architecture that makes possible the transformation of the server into a client of data at any moment. *SRE* allows for the communication among others Web Servers, such that multiple data sources can be integrated into a coherent system. SRE also supports the execution of different queries that gather useful data from each distributed data source.

Resumen

Nuestra investigación emergió de la necesidad de saber cómo recuperar y visualizar información gráfica/textual de imágenes que se encuentran almacenadas en diversas fuentes de datos. Nos enfocamos en un sistema de recuperación, visualización de datos e imágenes que se encuentran distribuidas heterogéneamente en las diferentes fuentes de datos proveyendo al usuario con un único punto de acceso a dichas fuentes de datos. Presentamos *Search and Retrieval Engine* (SRE) desarrollado como parte de TerraScope, un Sistema de Base de Datos Distribuido de Ciencia de la Tierra basado en Web. *SRE* es el motor de ejecución basado en tecnología Java-Servlet y en Arquitectura Punto a Punto, donde el servidor puede convertirse en cliente en cualquier momento. *SRE* nos permite la comunicación entre otros Servidores Web, donde diversas fuentes de datos pueden ser integrados en un sistema coherente, y apoyar la ejecución de diferentes peticiones que comparten datos útiles.

Acknowledgements

There are many people to whom I must thank for their direct or indirect support they have given to me and which have enabled me to carry out this thesis.

I would like to thank infinitely my advisor Dr. Manuel Rodriguez-Martinez for giving me the opportunity to work with him, for his continuous support, dedication and help in this project and for his friendship during all these years.

Thanks to my dissertation committee members for their comments and suggestions on my research work: Dr. Bienvenido Velez-Rivera, Dr. Pedro Rivera-Vega, and Professor Freya Toledo.

I also want to thank all family; my gratitude to all for their understanding, and for their sacrifices, especially when they wanted me to be with them and I was far from home. I must thank my parents, Wilma and Eduardo, for all their love, understanding, advice, and for always believe in me; without them maybe I would not be able to go on. Also I want to thank my sisters Zaira Karina and Zulman Omaira, my brother Paul Eduardo, and my nephew Jean Pierre, for their confidence in me, for the strength and words of courage that they give me every day.

My complete and deepest feeling of gratitude I dedicate to Jairo, my husband, whose love has been always the sustenance of all my work. His support and infinite patience has been the key for this work to become a reality.

I must extend my gratitude to all my friends of the Electrical and Computer Engineering Department that at many moments encourage me to follow ahead this thesis and to make my stay enjoyable here, at the University of Puerto Rico at Mayagüez; Carlos Acuña,

Hillary Caituiro, Carlos Huallparimachi and Idalides Vergara. To all of them, I give my thanks for their sincerity and amiability towards with me.

Finally, I would like to thank the Department of Electrical and Computer Engineering at University of Puerto Rico-Mayagüez Campus for this great opportunity to follow graduate studies in this prestigious university. I would like to thank the project CenSSIS (NSF award number EEC-9986821) for their financial support during this project.

Table of Contents

List of Tables	viii
List of Figures.....	ix
Introduction.....	1
1.1 The Problem	1
1.2 The Solution	3
1.3 Research Objectives	4
1.4 Summary of Contributions	5
1.5 Structure of this Thesis	6
Survey of Related Work	7
2.1 Distributed Database Systems and Database Middleware Systems	7
2.2 Peer – to – Peer (P2P).....	8
2.3 Spatial and Object Relational Databases - PostgreSQL	9
2.3.1 Index R – Tree Access Methods Spatial	9
Overview of the SRE Prototype System.....	11
3.1 Introduction	11
3.2 Overview of the Architecture of SRE.....	11
3.2.1 Client Application	13
3.2.2 Web Server.....	13
3.2.3 Database System	17
3.3 Components of SRE	20
3.3.1 Scenario of Communication.....	21
3.3.2 Messaging Communication.....	24
Peer – to – Peer Architecture.....	27
4.1 Overview of what is Peer – to – Peer (P2P) System.....	27
4.2 Comparison between Conventional Distributed Systems and Peer – to – Peer (P2P) .	29
4.3 SRE’s Peer – to – Peer Architecture.....	29

Experiments and Results.....	39
5.1 Introduction	39
5.2 Methodology.....	39
5.2.1 Equipment	39
5.2.2 Tools Used	41
5.3 Development of the Experiments	42
5.3.1 Centralized Architecture	42
5.3.2 Decentralized Architecture.....	45
Conclusions.....	59
Future Work.....	60

List of Tables

Table 5-1 Result obtained using a Centralized Architecture	43
Table 5-2 Result obtained using Decentralized Architecture	46
Table 5-3 Result obtained using a second structure of Decentralized Architecture	51
Table 5-4 Result obtained using a third structure of Decentralized Architecture	54
Table 5-5 Result obtained using a fourth structure of Decentralized Architecture.....	57

List of Figures

Figure 1-1 Problem 1: How to extract and to visualize graphical/textual information of images? .	2
Figure 1-2 Centralized Architecture	2
Figure 1-3 Excessive Data Movement and Centralization.....	3
Figure 3-1: Communications with a Servlet	12
Figure 3-2 Document of Configuration Terrascope.xml	14
Figure 3-3 Example of a Document of Configuration	15
Figure 3-4 Document of Configuration Broker.xml	16
Figure 3-5 Example of a Broker.xml	16
Figure 3-6 Overlap relationship	18
Figure 3-7 Architecture SRE - Schema Experimental Design.....	20
Figure 3-8 Scenery of Communication.....	21
Figure 3-9 Communication between Client Access Servlet and Data Broker Servlet.....	22
Figure 3-10 Communication between Data Broker Servlets of the others Web - Server (Peers)..	23
Figure 3-11 Communication between Data Broker Servlet and Information Gateway Servlet.....	24
Figure 3-12 Schema of the Request in XML format	25
Figure 3-13 Schema of the Response in XML format	26
Figure 4-1 Client – Server Model	28
Figure 4-2 Peer to Peer Architecture.....	30
Figure 4-3 IdGenerator Class.....	31
Figure 4-4 Setting Class.....	32
Figure 4-5 Id_Query process	33
Figure 4-6 Times - to - Life (ttl)	35
Figure 4-7 Data Broker Servlet Class	36
Figure 4-8 Information Gateway Servlet Class.....	37
Figure 5-1 Schema to process request in Centralized Architecture	42
Figure 5-2 Results of a Centralized Architecture.....	44
Figure 5-3 Schema to process request in Decentralized Architecture Peer – to - Peer.....	45
Figure 5-4 Results of our Decentralized Architecture Peer-to-Peer.....	47
Figure 5-5 Results obtains between both Architectures.....	48
Figure 5-6 Second structure of Decentralized Architecture.....	50

Figure 5-7 Result obtained with the second structure of a Decentralized Architecture.....	52
Figure 5-8 Third structure of a Decentralized Architecture.....	53
Figure 5-9 Result obtained with the third structure of a Decentralized Architecture	55
Figure 5-10 Fourth structure of Decentralized Architecture.....	56
Figure 5-11 Result obtained with the fourth structure of a Decentralized Architecture	58

Chapter One

Introduction

1.1 The Problem

In today's world we have the necessity of obtaining information in a quick, precise and instantaneous way, regardless of the geographical localization of the given information. Distributed Database systems that already exist have experienced an important growth in the volume of data to process.

The use of Distributed Database Systems, allows sharing and accessing information stored on local or remote databases in an easy and transparent way for the user, the easy management of temporal and spatial information, and offer an intuitive query language for data manipulation.

Currently, there exist many institutions, research centers, universities and government's agencies that allow us to access some data sources, obtain data and metadata of images that are stored in different data centers (**Figure 1-1**). Generally, they have a Web site to allow us to review some types of images, their spatial data and metadata. However, these solutions face scalability problems when heterogeneous data and computational resources need to be integrated to support applications that carry out an analysis of the different types of data sources stored by research centers.

We have found such occurrence in data centers such as TCESS (Tropical Center for Earth and Space Studies), CENSSIS (Center for Subsurface Sensing and Imaging Systems), or some sensors such as MODIS, Landsat, RadarSat among others.

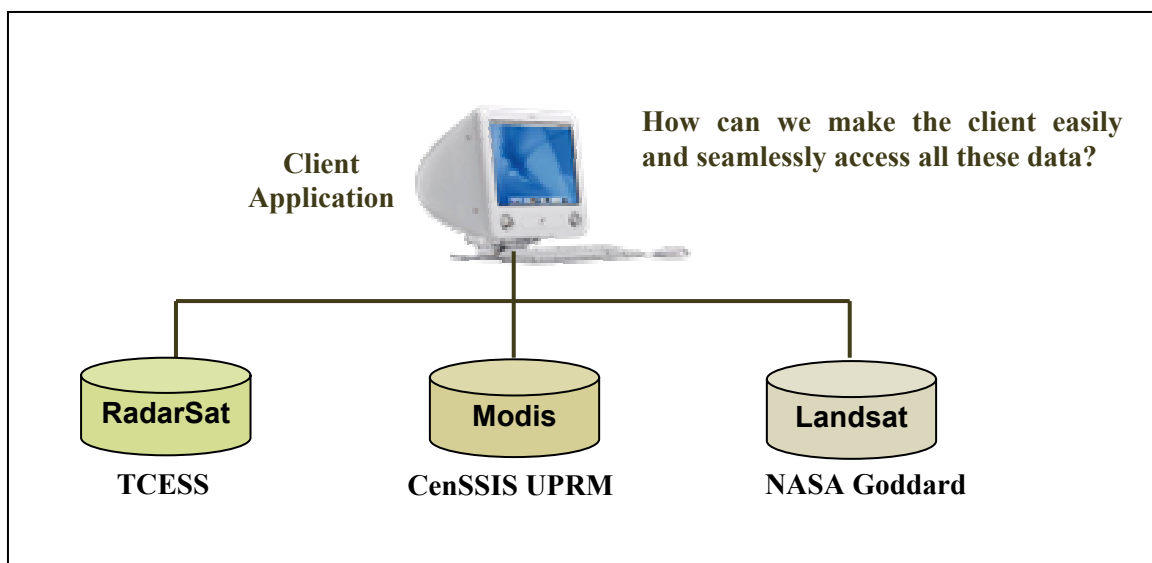


Figure 1-1 Problem 1: How to extract and to visualize graphical/textual information of images?

Also, our research is motivated by the drawbacks of the existing solution where there are many distributed systems with a centralized architecture. This architecture (**Figure 1-2**) has an integration server with a catalog that has the information and address of the others remote servers that are connected to this.

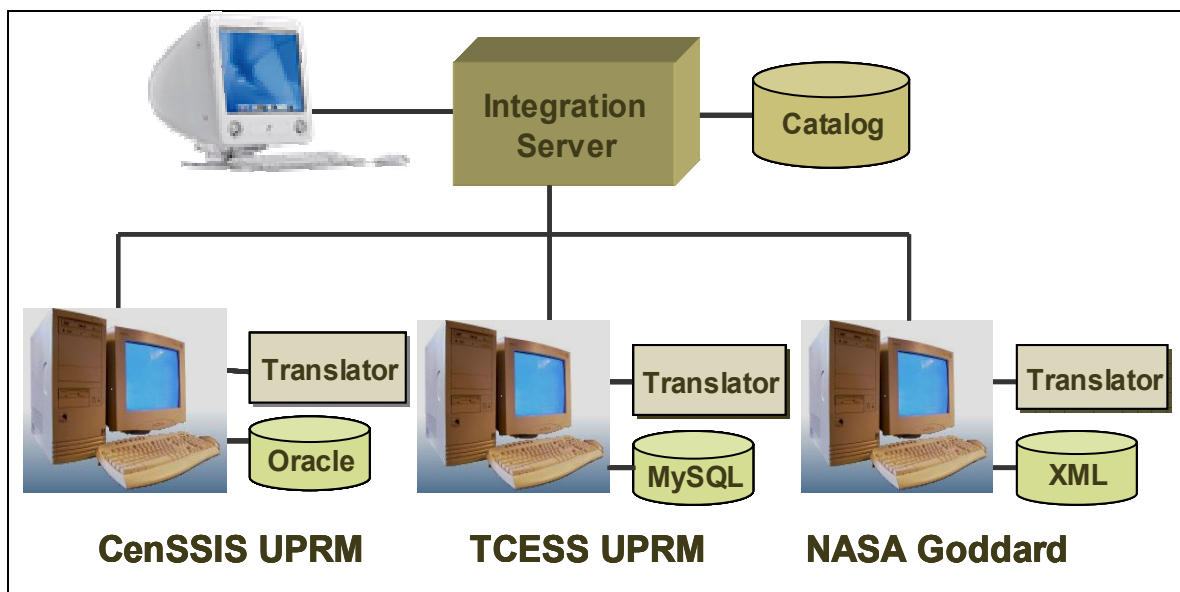


Figure 1-2 Centralized Architecture

For this reason, this type architecture has several disadvantages (**Figure 1-3**). First, there exists too much data movement exist over the network making query processing very slowly, because the integration server does most of the processing. Second, there is a reliability problem; if the main server fails the entire system fails, thus this server becomes a single point of failure.

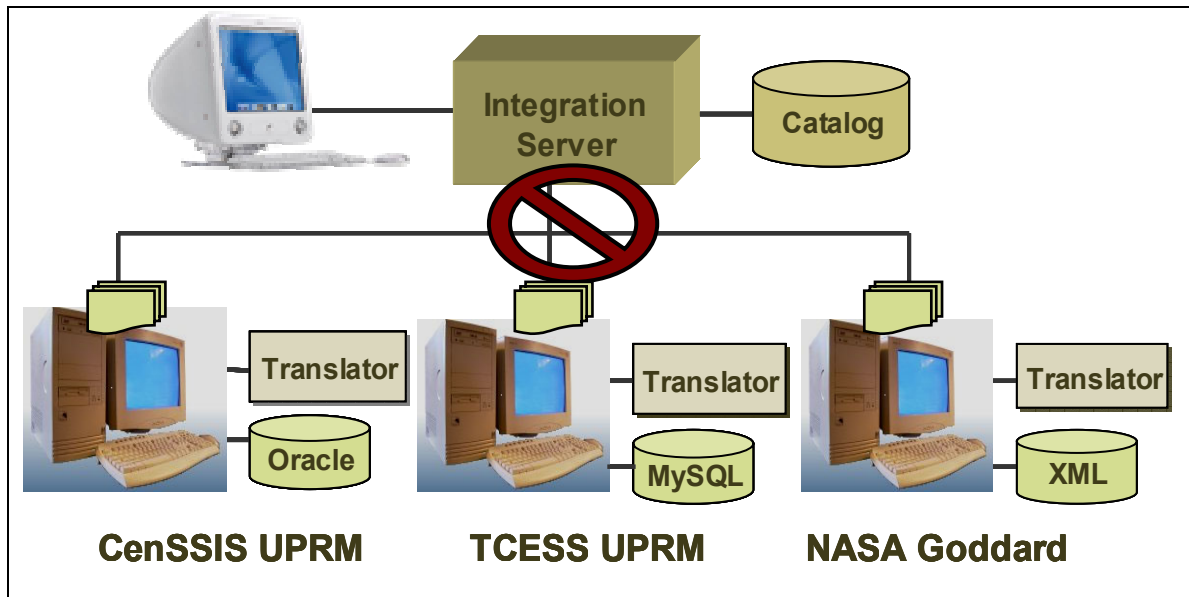


Figure 1-3 Excessive Data Movement and Centralization

1.2 The Solution

The proposed solution of this research is to develop a system that allows us to obtain information about images and their metadata that are distributed and stored in different data sources. For this reason we present the *Search and Retrieval Engine (SRE)* as part of *TerraScope* that is an *Earth Science Middleware Database System* based on the Web.

SRE is a spatial distributed database system and it has a distributed query execution engine based on the *Java Servlet Technology* that can respond quickly to each query request. It has methods that make it possible for the user to search the wanted information based on temporal data, geographical area, data sources or sensor type.

Also, SRE is based on a scalable and decentralized *Peer-to-Peer architecture* where multiple data sources can be integrated in a coherent system. It receives requests from the users, and accesses each database that is distributed in different data source. It can support the execution of different queries that make it possible to gather information that are stored in each data center where the users of TerraScope can visually manipulate spatial data local and remotely stored.

In short, SRE is designed to support spatial indexes, distributed data recovery and visualization of images. It does not require existing data sources to be removed, and their data re-ingested into a new storage server.

The system SRE is composed of:

- Web Server - responds to HTTP requests submitted by the clients, and where SRE will be installed as part of the Servlets deployment infrastructure.
- Database System - where the data and metadata corresponding to each of the images are stored.

1.3 Research Objectives

The main objective is to develop a Peer-to-Peer Web-based middleware engine that can handle search operations, spatial queries, and other operations on data sets stored on different data sources. The system shall have a decentralized architecture that minimizes the likelihood that a single component can become a system bottleneck or cause a major system failure.

The specific objectives of the *Search and Retrieval Engine* are listed as follows:

- To develop a prototype server with capabilities for search and recovery of data and/or metadata from different sources, for instance, TCESS, CENSSIS, NASA and others.

- Peer-to-Peer architecture to retrieve Remote Sensing Images. By using peer-to-peer architecture, there are simple access points to the system, which will allow us to add other sites and metadata accordingly as the system grows.
- To increase the performance in the recovery of spatial objects with the use of spatial R-tree indexes, which provide the most efficient access to spatial data.
- To have a portable system, using Java Servlet Technology, where the ingestion component transforms the metadata of several data sources into a single XML document. The XML language is ideal to describe data based on ASCII because a lot of metadata of earth science are in an ASCII format that can be easily handled.

1.4 Summary of Contributions

The major contributions of this thesis can be summarized as follows:

1. Development of the SRE system to enable a Web server to become a database middleware engine for processing data and metadata related with satellite images. The SRE is implemented with Java Servlets Technology, and our work serves as evidence for the feasibility of this approach.
2. The development of a the SRE system based on a Decentralized Peer-to-Peer Architecture, that allow us have access to different data sources that are located in different locations and can obtain the data and metadata referent to images satellites. The system allows access to any available data source from a single point of entry.
3. A performance evaluation that show that our architecture is more efficient than the centralized architectures used in previous systems.

4. Finally, other of my major contributions was the publication and presentation of a paper wrote to the Conference of **IASTED CST** (*Computer Science and Technology*) – 2003, realized in Cancun, Mexico in May 19 to 22 at present year.

1.5 Structure of this Thesis

The reminder of this thesis is organized as follows, Chapter 2 presents a survey of related arranged in three subtopics: i) Database Distributed System and Database Middleware Systems; ii) Peer – to – Peer Architecture; and iii) PostgreSQL R-Tree Index. Chapter 3 presents an overview of the SRE Architecture, discussing its main components. Chapter 4 presents the Peer – to – Peer Architecture of our SRE, describing and analyzing its operational behavior. Chapter 5 presents the performance experiments and the results obtained. Finally, Chapter 6 presents the final conclusions and future work.

Chapter Two

Survey of Related Work

We present a review of the research literature most relevant to the work done for this thesis. We only consider here the work upon which this thesis is based.

2.1 Distributed Database Systems and Database Middleware Systems

Researchers in the area of *Distributed Database Systems* have concentrated on the problems of heterogeneous data integration [12, 13], distributed query processing [14] distributed transaction processing [15], and data dissemination systems. *Database Middleware Systems* are used to integrate collections of data sources distributed over a computer network [1].

The *MARIPOSA* distributed database management system is a research prototype developed at the University of California at Berkeley [3, 4]. *Mariposa* addresses fundamental problems in the standard approach to distributed data management. *Mariposa* allows DBMSs which are far apart and under different administrative domains to work together to process queries, and provides powerful mechanisms for specifying the behavior of each site. However, *Mariposa* require data to be removed from their existing server applications, and re-ingested into the federated database engine.

Database Middleware differs from Network Middleware such as CORBA, .NET and RMI, since the latter is used as an infrastructure layer to provide applications access to the network. *Database Middleware* is at a higher layer, and can leverage on the Network Middleware for connectivity purposes. *Database Middleware Systems* follow an architecture centered on a *data integration server*, which provides client applications with a uniform view and a uniform access mechanism to the data available in each source.

MOCHA is a novel database middleware system designed to interconnect hundreds of data sources distributed over a wide area network. The purpose of *MOCHA* is to integrate collections of data sources distributed over wide-area computer networks such as the Internet [1]. *MOCHA* is a database middleware system that provides client applications with a uniform view and access mechanism to the data collections available in each source. It provided the ability to ship Java code implementing user-defined types and operations to remote data sites, but have a problem of scalability as new sites are added to the system and a single site must manage system-wide interactions. In contrast our **Search and Retrieval Engine of TerraScope** is based in peer to peer architecture that ameliorates this problem.

TERRASERVER is the largest public repository of aerial, satellite and topographic data based on a simple client - server model. *TerraServer* needs the data of interest to be fetched from remote sites into the client site, where most of the processing occurs. The data processing options are limited to the client or the server site. In contrast, the **Search and Retrieval Engine** has more options for data processing and the data can be moved to sites appropriately.

MERCURY is based on a simple client-server model that requires the data of interest to be fetched from remote sites into the client site, where most of the processing occurs. *Mercury* only gives access to the metadata for data products. The actual image data must be obtained by other means such as FTP or HTTP. In contrast, SRE provides a common interface to access both data and metadata available in some image repository.

2.2 Peer – to – Peer (P2P)

The use of P2P architectures opens up new dimensions of handling and managing the information facilitating the exchange of the most recently created and highly distributed information. The most frequently used applications include file sharing systems, such as Napster, Gnutella, among others. It has a decentralized control able to cope with challenges such as metadata, cost sharing and security.

The decentralized P2P system gives them the potential to be robust to faults or intentional attacks, making them ideal for long-term storage.

2.3 Spatial and Object Relational Databases - PostgreSQL

PostgreSQL is a traditional relational database management system (DBMSs) that support a data model consisting of a collection of named relations, containing attributes of a specific type. In current commercial systems, possible types include floating point numbers, integers, character strings, money, and dates. The relational model successfully replaced previous models in part because of its "Spartan simplicity". However, as mentioned, this simplicity often makes the implementation of certain applications very difficult.

Although PostgreSQL has some object-oriented features such as object data-types, it is firmly in the relational database world. In fact, some commercial databases have recently incorporated features pioneered by Postgres.

2.3.1 Index R – Tree Access Methods Spatial

A Spatial Database System is a database System able to manage spatial information such points, lines, and polygons in 2D, 3D, or any other type of n-dimensional Euclidean space. These systems also offer a query language for the manipulation of these spatial entities, particularly operators to perform operations such overlaps, intersection and containment queries. The objective of a spatial database management system is providing efficient access methods and algorithms for the processing of queries over the spatial attributes of the data sets [Güt94].

The most traditional methods of this type were not devised to be used for three (3) dimensions and only recently structures thought for a high dimensionality have arisen. For that reason, the spatial access methods are going to be those build mainly to store geometric objects.

The pioneering structure is *R-Tree* [Gut84] that has served as a base for many other methods that arose by trying to improve some aspects of the basic R-tree. An R-tree is a multidimensional tree balanced in height that represents one hierarchy of rectangles representing regions of the minimal bounding box (MBRs – Minimum Bounding Rectangle) that cover some object X in the target Euclidean space.

In order to increase the performance of the query processing system, access methods have been developed [GaG98]. The one-dimensional index (data structure with unique key) are B-Tree [BaM72] and its derivatives B+-Tree, B*-Tree, etc. A group of derived index from B-Tree to manage multidimensional index is known as R-Trees (trees data structures based on minimal rectangles (MBRs)). The MBRs in R-trees are defined by two points, forming a rectangle that encloses the object to be indexed. There are many variations of the original R-Tree: R+-Tree, R*-Tree, and X-Tree. Each of these variations uses a different algorithm to partition the space, split tree nodes, or merge tree nodes in order to keep the tree balanced.

Chapter Three

Overview of the SRE Prototype System

3.1 Introduction

Search and Retrieval Engine (SRE) is a spatial database middleware system that is composed for a group of servlets developed with *Java Servlets Technology*, capable of communication between them through XML syntax. Each Servlets has a different function to help process each request sent by the client through the *Client Application* obtaining a final result from the diverse data sources that our server has access to. The servlets implement a distributed data processing engine that will enable operations that extract and process data from multiple sites. We can install our SRE as part of the Web – Server infrastructure available at a given site that is holding an image repository.

To enable access a data source that provides access to image data and metadata, we have worked and created the databases using *PostgreSQL* because it allows us to manage spatial indexes that facilitate the search. In PostgreSQL, we have created databases where the data and metadata about images are stored correspondingly.

3.2 Overview of the Architecture of SRE

SRE is based on Java Servlet Technology and on Decentralized Peer-to-Peer architecture, where the clients can customize their own view of the system to define the remote data and computational services they wish to export. They can control which of their services and data products they are willing to share with others.

The *Servlet* are programs in Java that are executed on a Web Server, and that receive and respond to request from the clients. The client can invoke the Http Protocol (**Figure 3-1**). Any given servlet can communicate with others servlets to help in its work or facilitate the access to a database. A Servlet is secure and portable because of its

execution on the java virtual machine, the mechanism for exception handling, and the use of the java security infrastructure.

The API of the servlets used for servlet programming includes two packages: i) `javax.servlet` which defines the interface for all servlet and the implementation a `GenericServlet` class. Here are we find defined the methods of the servlet Life cycle, and also the interfaces `ServletRequest` and `ServletResponse`, that define a client's request and response; ii) `javax.servlet.http`, which includes servlet interfaces and classes for use with the protocol http. This package defines the class `HttpServlet` that is one extension of `GenericServlet`; also include the interfaces `HttpServletRequest` and `HttpServletResponse`, which are extensions of `ServletRequest` and `ServletResponse`.

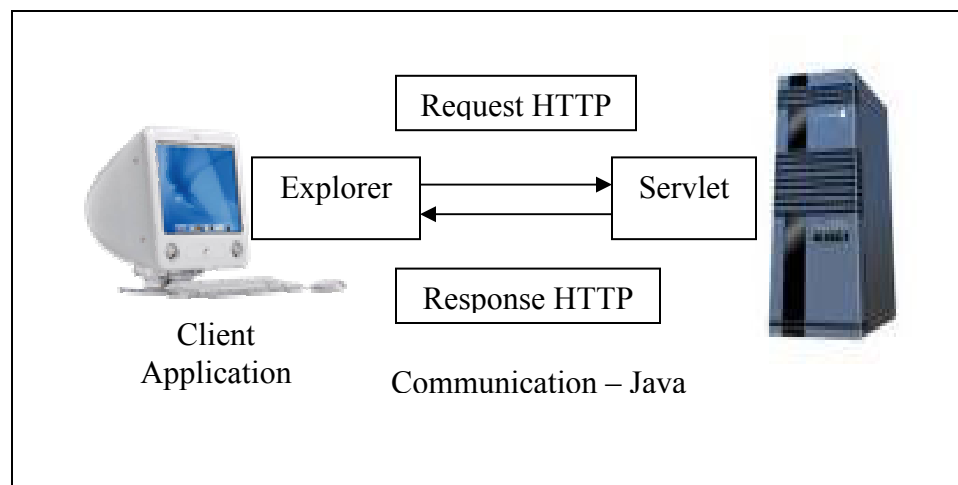


Figure 3-1: Communications with a Servlet

The reason behind our decision to work with the *Java Servlet Technology* is that it allows us run our SRE in different platforms; our execution is multithread where each request creates a thread instance that is executed independently. Also, a servlet allow us invoke to other servlets located locally or on a remote site. Our system is organized in three tiers: a) Client Application, b) Web Servers, and c) Database System. We now give more details on these elements.

3.2.1 Client Application

The client application is the graphical user interface where there are multiple options for doing the search of images that are stored in different data sources. This application is done in Macromedia Flash that was realized for other student of the group of research.





3.2.2 Web Server

The *Web Server* is the most important part in any Internet site, and it constitutes the next step in the evolution of the distributed object-oriented technology, in charge of new distributed architectures such as peer-to-peer. The Web Server takes care of responding to HTTP requests submitted by the clients, and facilitating the growth of the system by adding new sites and their metadata, without having to change the structure of the system. Our SRE system is installed as part of each Web Server that are running on the different data sources.

The Web server that we used to support Java Servlets is *Tomcat*, which is a Servlet Container. We used the *Java Web Service Developer Pack* (JWS DP) that have the Tomcat Server that allows us run our servlets. JWS DP is easy to obtain, free cost and offers us the best ratio of ease-of-use/communications cost. An applet runs in a web browser, performing actions it requests through a specific interface. A servlet does the same type action, but the servlet is running on the web server. The requests from clients are sent through a graphical client (GUI) encoded with XML format, and sent to the web server via a URL.

Each SRE is installed as part of the web-server and has three main components: i) *Client Access Servlet* - that interacts directly with the petitions requested by the clients; ii) *Data Broker Servlet* - its main functions are processing, controlling and coordinating the communication process between all the others Data Broker Servlets that are located or distributed in the others web-servers, independently of the level of connectivity (remote or local); and finally iii) *Information Gateway Servlet* - that allows us to access and retrieve data requests from its associate database.

For our *SRE* to work correctly we have to follow some steps:

- First, we install SRE in the WEB-INF subdirectory of the jwsdp directory. This is:
 -  jwsdp
 -  webapps
 -  ROOT
 -  WEB-INF
- Second, to configure two (2) main documents that are in the ROOT directory, and that are part of our SRE.
 - a) *Terrascope.xml*. (See **Figure 3–2**). This document is a configuration document own by each Web server where SRE will run. It has some specifications such as: hostname and port where SRE is running, the hostname where SRE is installed, the controlling number of hops (called ttl), JDBC database driver name and address of the database, the user name and password of person what have access to the database.

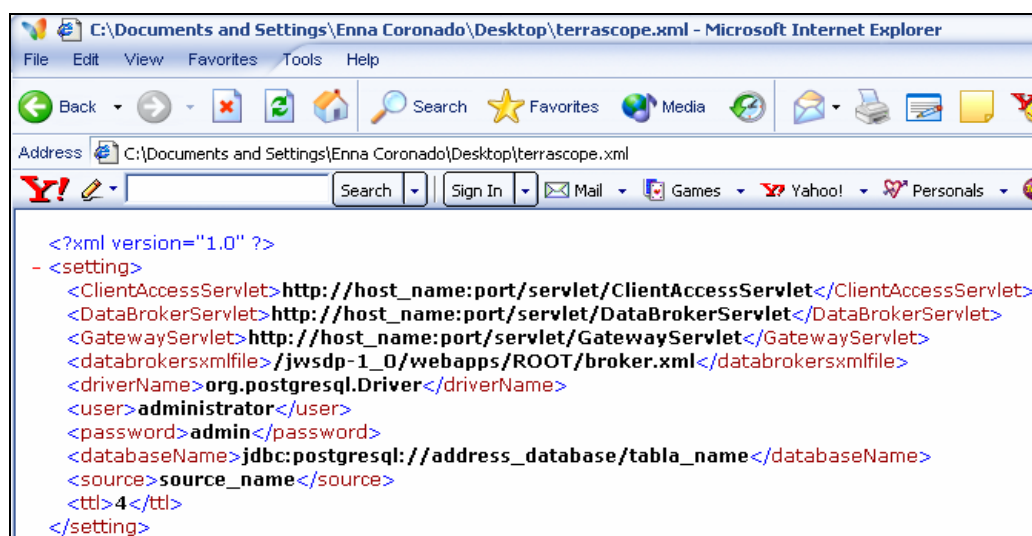


Figure 3-2 Document of Configuration Terrascope.xml

For a major understanding, we show in **Figure 3–3** how to configure this document for this case. In this case, the hostname where we installed our SRE is called **icarus.ece.uprm.edu** and the port where it will run is the **9090**, then the addresses of our three main servlet are shown below. Since the database belongs to PostgreSQL then the driver name also belongs to `org.postgresql.Driver`.

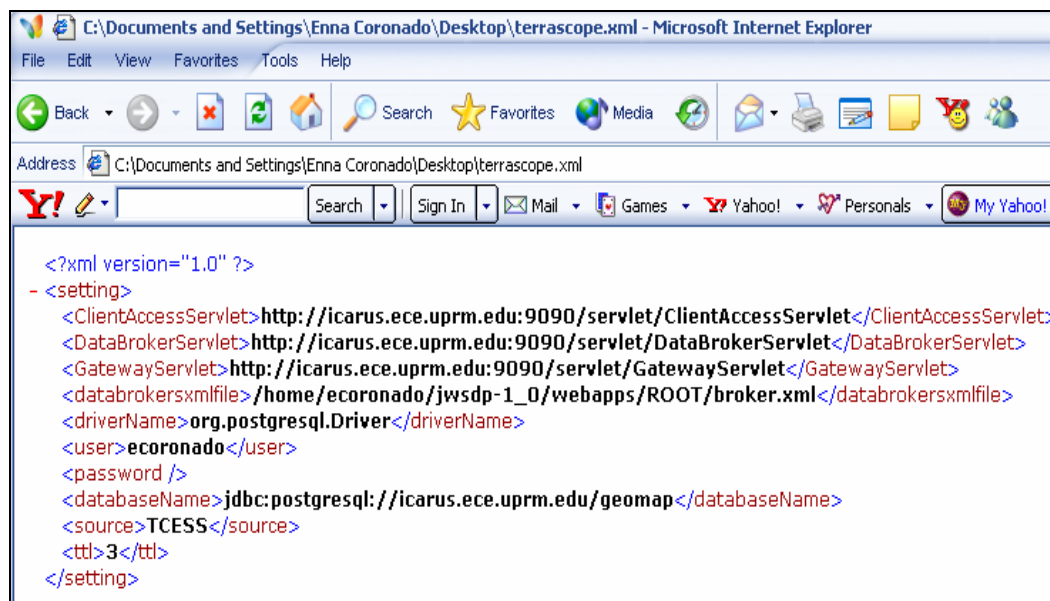


Figure 3–3 Example of a Document of Configuration

We can see that each tag has a different function. The three first tags belong to addresses located in our three main servlets. The next tag is the address where the document **broker.xml** is located in the ROOT. Then, the tags such as `driverName`, `user`, `password`, `databaseName` are referring to handling and access operations of the database.

- b) *Broker.xml*. As shown in **Figure 3–4**, Broker.xml contains only the addresses of the other Data Brokers (tag `Name_Source`) which are part of the Peer – to – Peer system. Using this list one peer had the ability to forward information about of the request sent by the client.

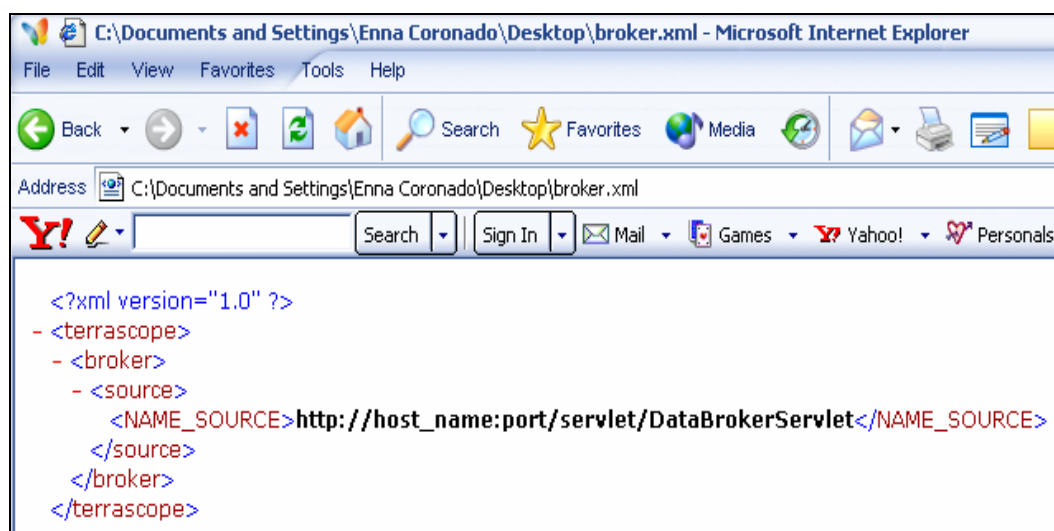


Figure 3-4 Document of Configuration Broker.xml

For example, in the **Figure 3-5** we can see that our server recognized three peers (friends) namely: TCESS, CenSSIS and NASA. In order to request information from them, our server must add them into the Broker.xml document as it is shown in the figure below.

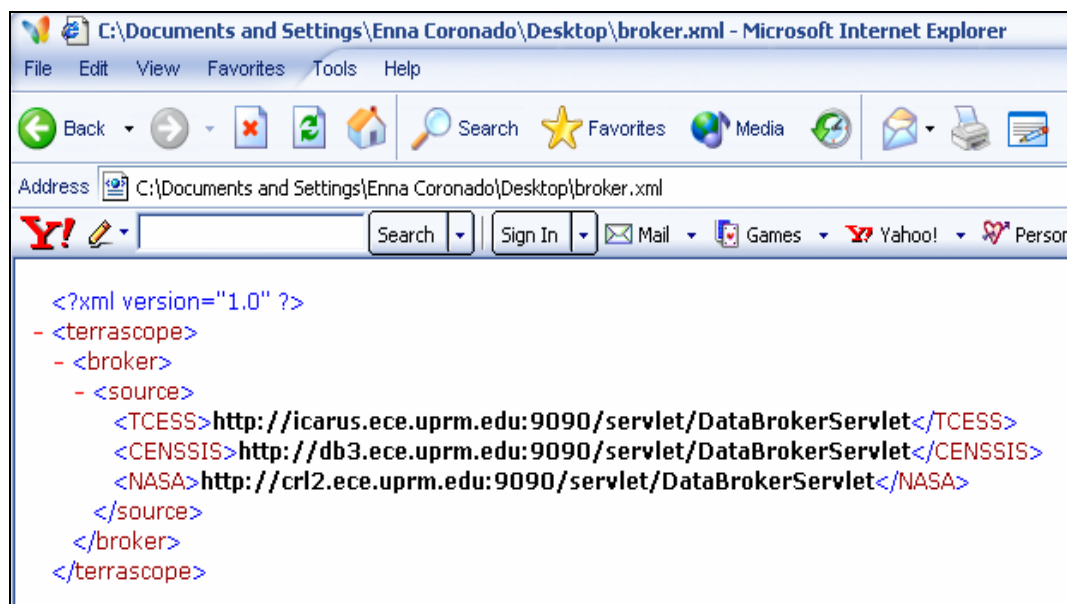


Figure 3-5 Example of a Broker.xml

3.2.3 Database System

Our spatial databases were created for storing the data and metadata for each one of the images. It is formed of various tables where information that corresponds to each of images will be stored. Our current implementation is based on the PostgreSQL database system because it allows us to manipulate and create spatial indexes, particularly R-tree indexes.

3.2.3.1 Spatial Access Methods

The *R - Trees* is a tree data structure, balanced in height, similar to the B - Tree; however, instead of value ranges, R-trees are associated with Minimum Bounding regions (MBRs) with each node of the tree. Each MBR encloses either an individual object in the tree, or a collection of two or more MBRs of smaller size. Each leaf tree node corresponds to a data page. In the nodes leaf, an identifier for each object and the MBR that contains it is stored and the internal nodes represent a succession of minimal rectangle regions, each one of which it controls a node in the inferior level. The regions of the same level can be overlapped and its union not necessarily covers the complete universe of spatial positions.

One of the methods that allow us to handle overlaps regions is the R-Tree and PostgreSQL allow us create this index type. PostgreSQL allows us to use a data type called *polygon* where each one is represented internally as a collection of points. Our databases allow us to build B-trees as well as R-tree indexes in order to do spatial searches to obtain an efficient answer to this type of requirement. For this reason, we used the method of *R-Tree* index because it can cover areas in multidimensional spaces that are not well represented by locations points.

A spatial database consists of a collection of tuples (records) representing spatial objects where each tuple (record) has a unique identifier that can be used to retrieve its leaf nodes R-Tree contains index record entries of the form (I, id_tuple) where “I” is an

n-dimensional rectangle which is the bounding box of the spatial object indexed and “id_tuple” is a tuple in the database.

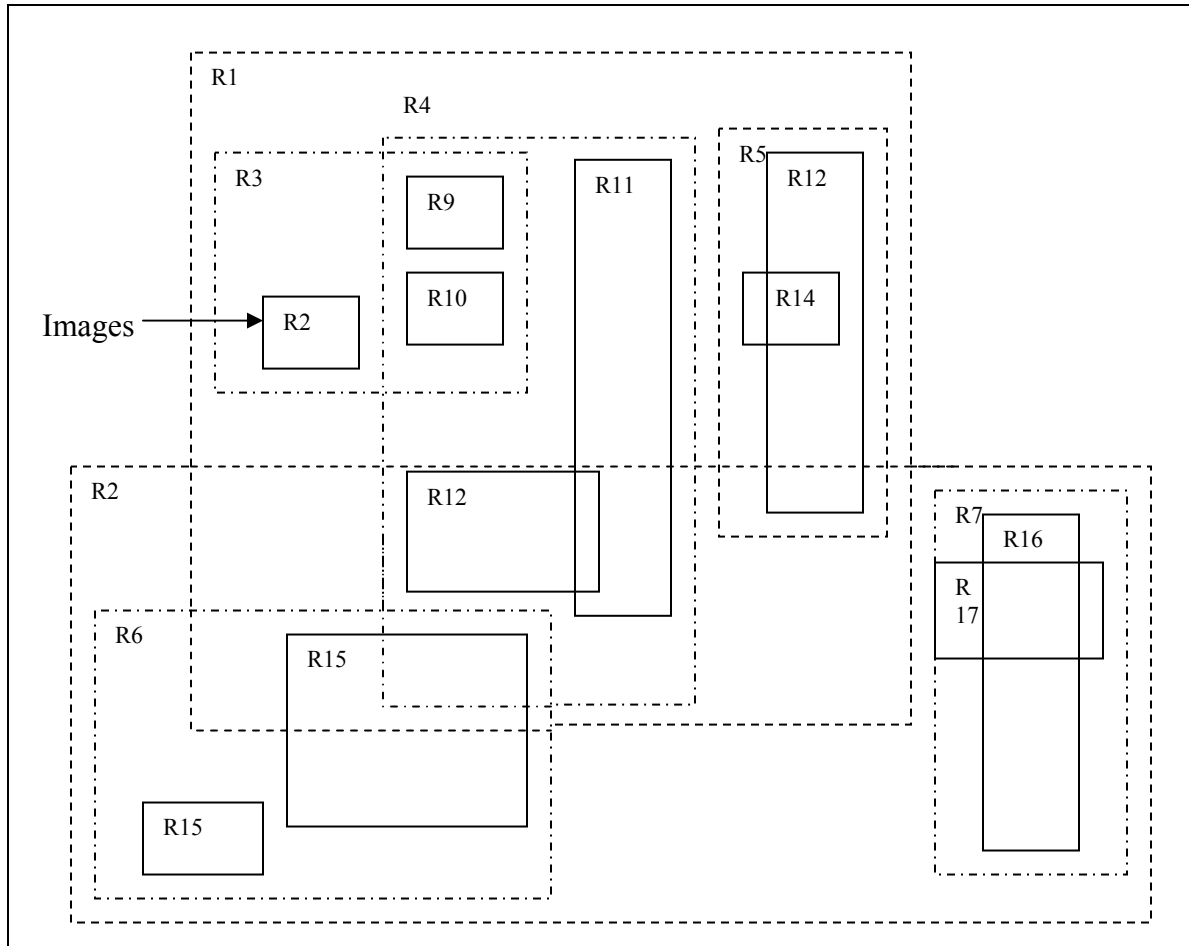


Figure 3-6 Overlap relationship

In **Figure 3-6** we show overlapping relationships that can exist between spatial searches. Spatial searches are represented in picture 3-6 as a rectangle. Spatial searches are achieved using spatial indexes or polygons that overlap in a given range. Those queries require the use of a spatial index because it will result extremely difficult in a system with a single support B-tree. The use of the operator "&&" in a query formulated with the Structured Query Language (SQL) will allow us to bring all images that are overlapped inside a selected range, and finally recursive queries allows us to bring the images belonging to a certain geographical point.

3.2.3.2 Connection with the Database

SRE use the *JDBC (Java DataBase Connectivity)* Standard to access the Database. The JDBC API makes the access to database knowing which server to access (Oracle, PostgreSQL, Informix and others). It allows us the connection with a database, to send a SQL query and to process the results. JDBC's main advantage is that an application (in Java) can interoperate with multiple data sources, and connect to different databases with a similar set of operations.

3.2.3.3 Supported Queries

SRE has some methods that will make it possible to the user to search information from any image, obtaining its respective data and metadata, provide a quality of service semantics to the data sites involved in query execution and to assure resource sharing, reliability, performance and security. The main characteristic of SRE is the capacity of processing spatial data that are stored in multiple data sources. SRE provides for the efficient management of those data in response to related queries with spatial properties.

The queries can be expressed in terms of:

- *Spatial Data*. It is an exact query that looks for spatial objects determined by their location. For this, we use the R-tree indexing tool available with PostgreSQL.
- *Sensor type* or some other characteristics.
- *Temporal Data*
- *Data Sources*.

The final result of the query is send to the user by means of an XML result set that contains the data and metadata with its respective URL where the corresponding images are stored. The *client application* receives the XML results and presents the data to the end user.

3.3 Components of SRE

Search and Retrieval Engine (SRE) is the main search engine and it is conformed of a group of servlets that communicate between them, where each servlet has a different function to process each request sent by the client obtaining a final result from the diverse data sources that our server has access to. The Servlets implement a distributed data processing engine that will enable operations that extract and process data from multiple sites.

In the **Figure 3-7** we show our implementation design scheme of our research. We can see that each Web Server contains a group of the main servlets that cooperate between them, forming the SRE.

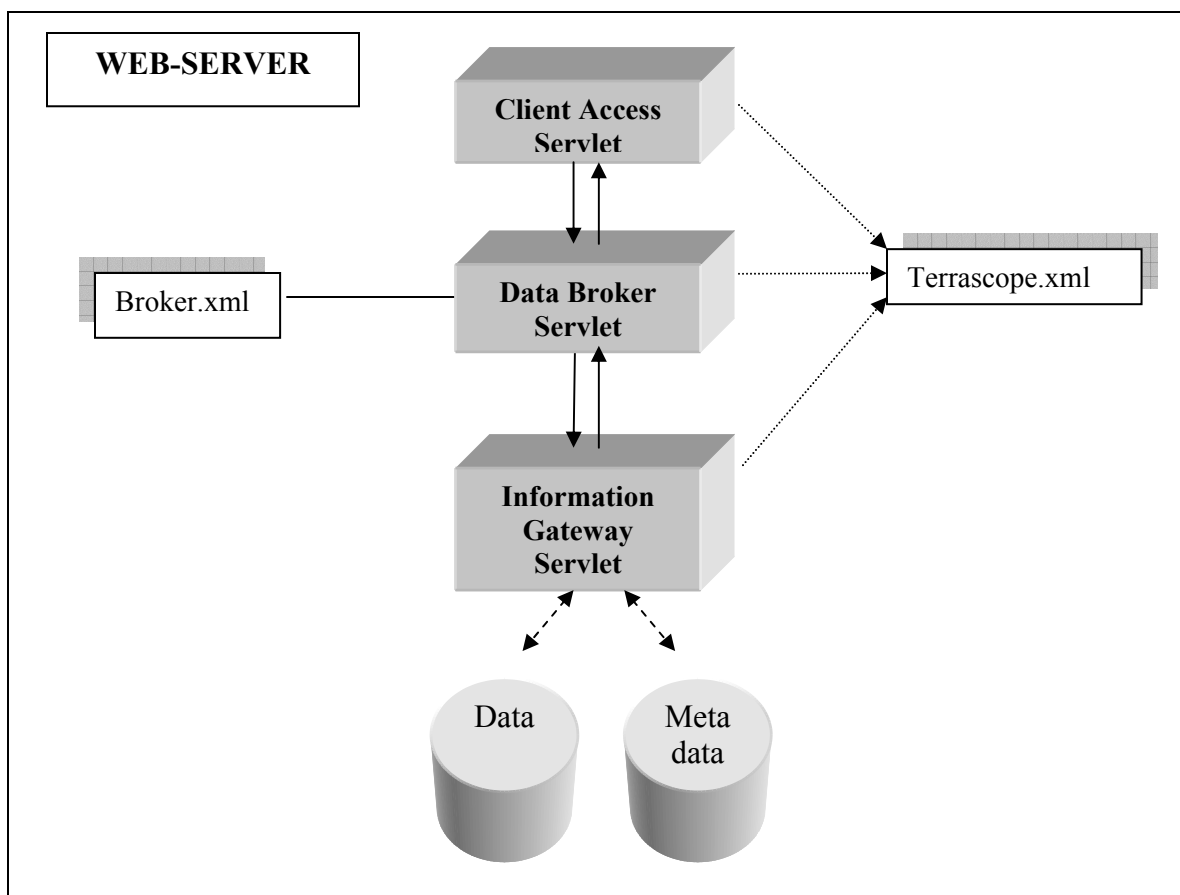


Figure 3-7 Architecture SRE - Schema Experimental Design

3.3.1 Scenario of Communication

Consider for example, as represented in **Figure 3-8**, what happens when an user requests the following query through to the *client application*: “Get all the Radarsat Images of CENSSIS acquired between 03/16/2001 and 03/20/2001”. Our SRE system will act as a data and resource broker for the application, where the client application will send this request to the Web - Server.

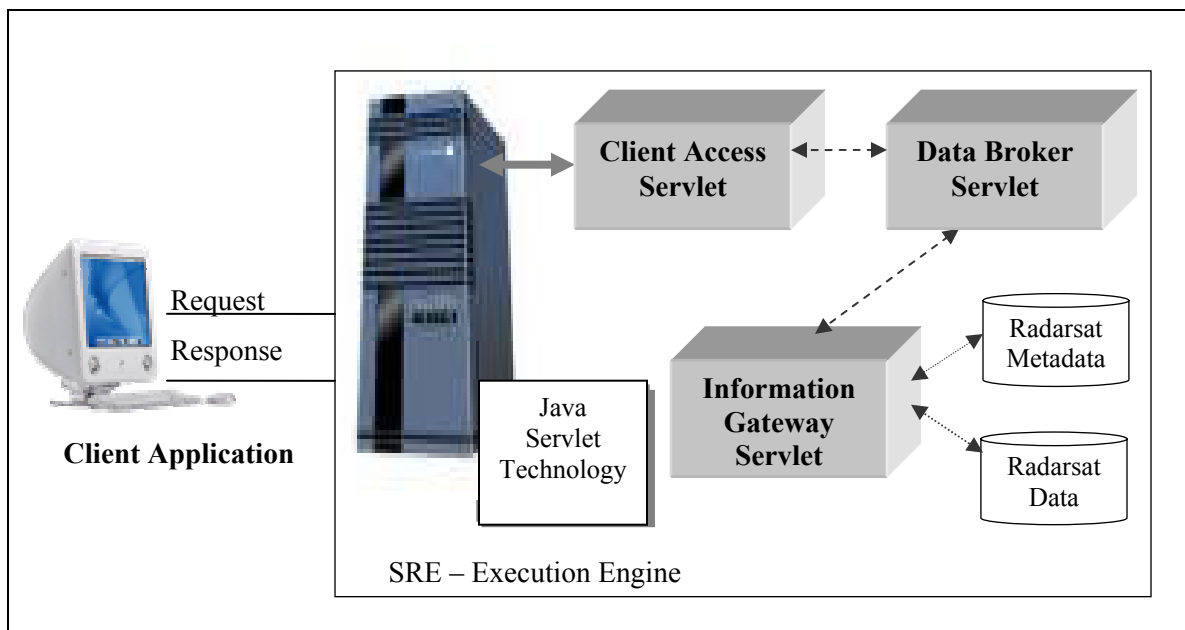


Figure 3-8 Scenery of Communication

Each one of the components behaves as follows:

- a. The ***Client Access Servlet*** (CAS) is in charge of interacting directly with the requests made by the client, as shown in **Figure 3-9**. The CAS receives the requests sent by the client. The requests are encoded using a XML format which is interpreted and validated by the *CAS* using a proper parsing function. This servlet verifies the query type that the user has chosen and determines if the query results are already stored in a memory cache data structure, or if the query is a new query not in the cache. If the results are stored in memory, CAS will send the response to the end-user without making

further effort; otherwise, CAS will make the connection with the next servlet, the Data Broker, and forward to it the request.

CAS extends the `HttpServlet` class provided by the *Java Web Service Developer Pack* (JWS DP) that receives the requests of multiple clients in and prevents the use of a port that a Firewall could block.

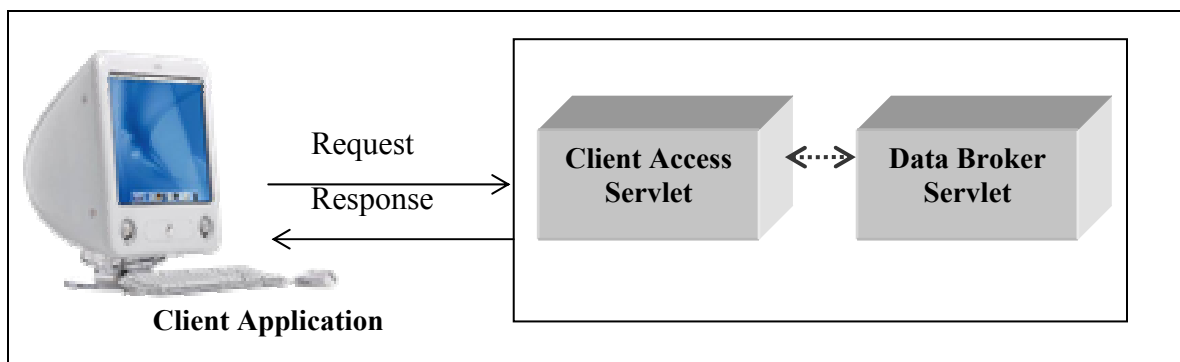


Figure 3-9 Communication between Client Access Servlet and Data Broker Servlet

b. The **Data Broker Servlet** (DBS) receives the client's requests from the CAS in a more precise and detailed way, as shown in **Figure 3-10**. DBS processes the information, and also controls and coordinates the communication that exists between the others Data Broker Servlets that are installed in each one of the others peer sites. DBS determines a strategy to solve the queries in a quick and precise way; it verifies what data sources have been invoked by the clients, consulting in a catalog stored in XML.

The broker also determines the remote peer sites where the query request will be sent to extract the data and metadata of a group of images. Once the DBS has found groups of peer sites, it will negotiate with them for access rights, and then it will send the query request to be resolved. Otherwise, if no specific data sources are found in the local catalogs, then the data broker will request to others peers information leading to solve the

query at hand. In this latter case, we assume that some peer site might have knowledge of the target data sources, and provide the required images.

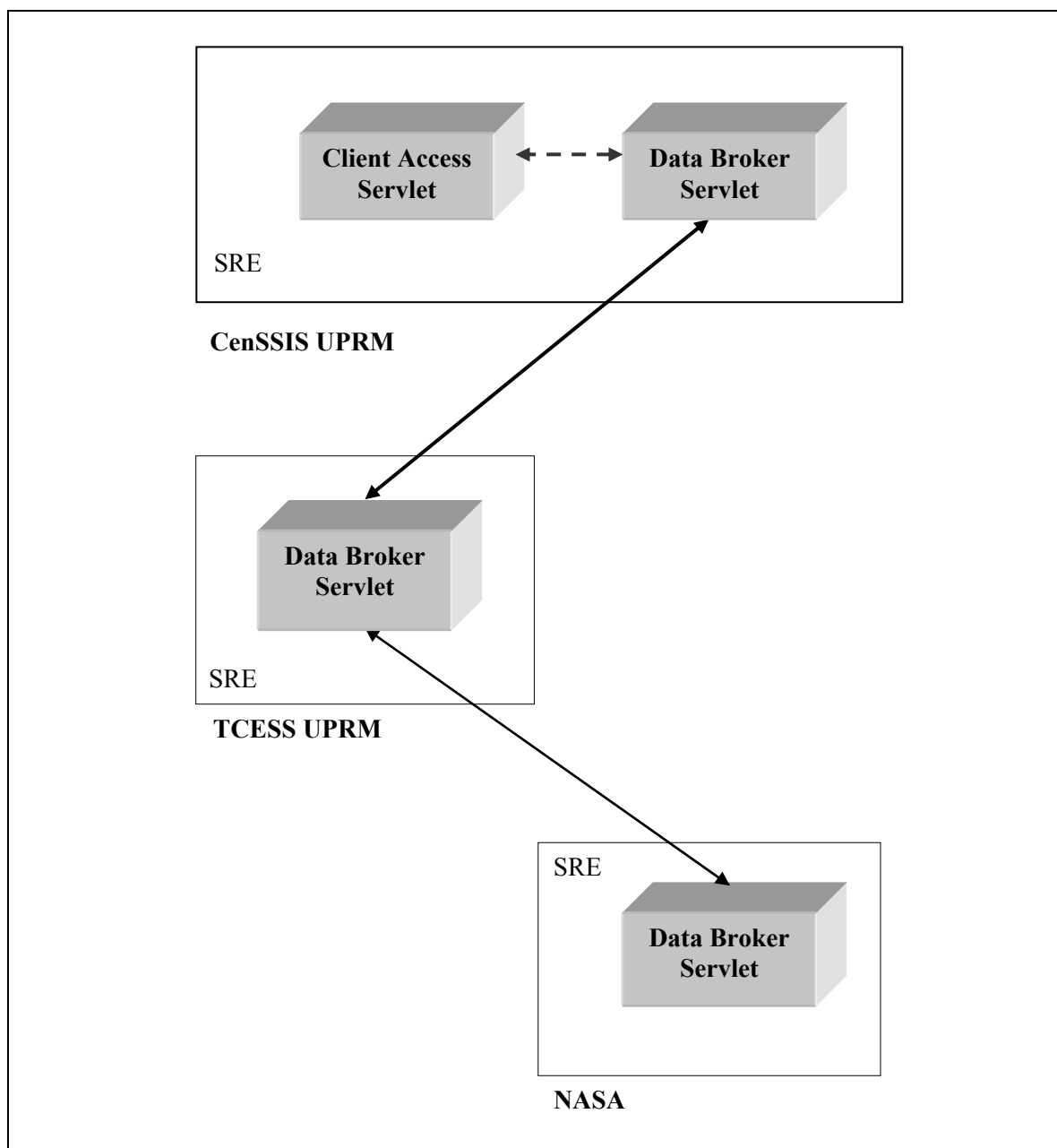


Figure 3-10 Communication between Data Broker Servlets of the others Web - Server (Peers)

c. The **Information Gateway Servlet** (IGS) will receive the query request that the Client sends through the CAS and DBS, as shown in **Figure 3-11**. IGS is the elements that control access to the database, and extracts the significant information according to the queries received, creating a new XML document with the final result containing the respective data and metadata. Thus, IGS provides access to other members of the system to the data sets maintained by a given source site.

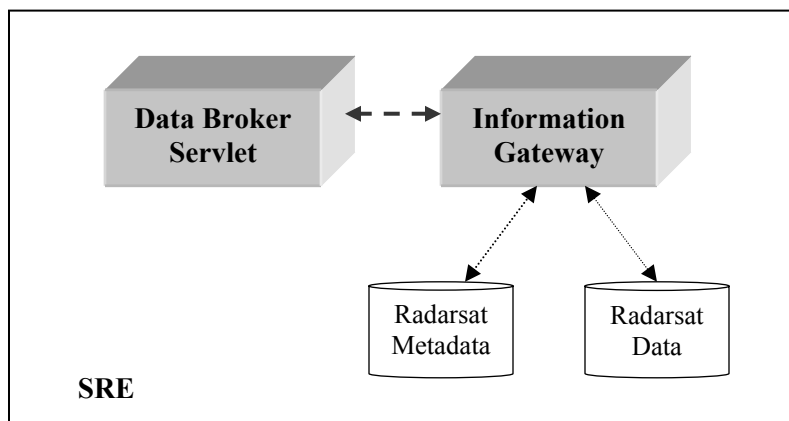


Figure 3-11 Communication between Data Broker Servlet and Information Gateway Servlet

3.3.2 Messaging Communication

The communication between the servlets is performed through XML messages, which are a new technology for web applications. XML is a World Wide Web Consortium standard that lets you create your own tags to enclosed data that will be exchanged between sites. Thu, XML can be used a language for communications can is understood and processed by the server applications (namely the Servlets). Hence, XML provides both data and semantics for the data, which can be exploited by the applications since they can find the data of interest by looking for the tags that enclosed the data. XML applications provide many advantages. XML's strongest point is its ability to do data interchanges and make it easy to send structured data across the web so that nothing gets lost in translation.

3.3.2.1 Schema of the Query Request in XML

This schema shown in **Figure 3-12** is an XML document containing the request placed through the Client Application. The corpus of the XML document is composed by detailed information such as data sources name, range of searching coordinates, and dates. This format will be used by the client to send a message, through a URL to the Web Server where our Search and Retrieval Engine (SRE) is running, and then this request is received and parsed by our first Servlet, the Client Access Servlet.

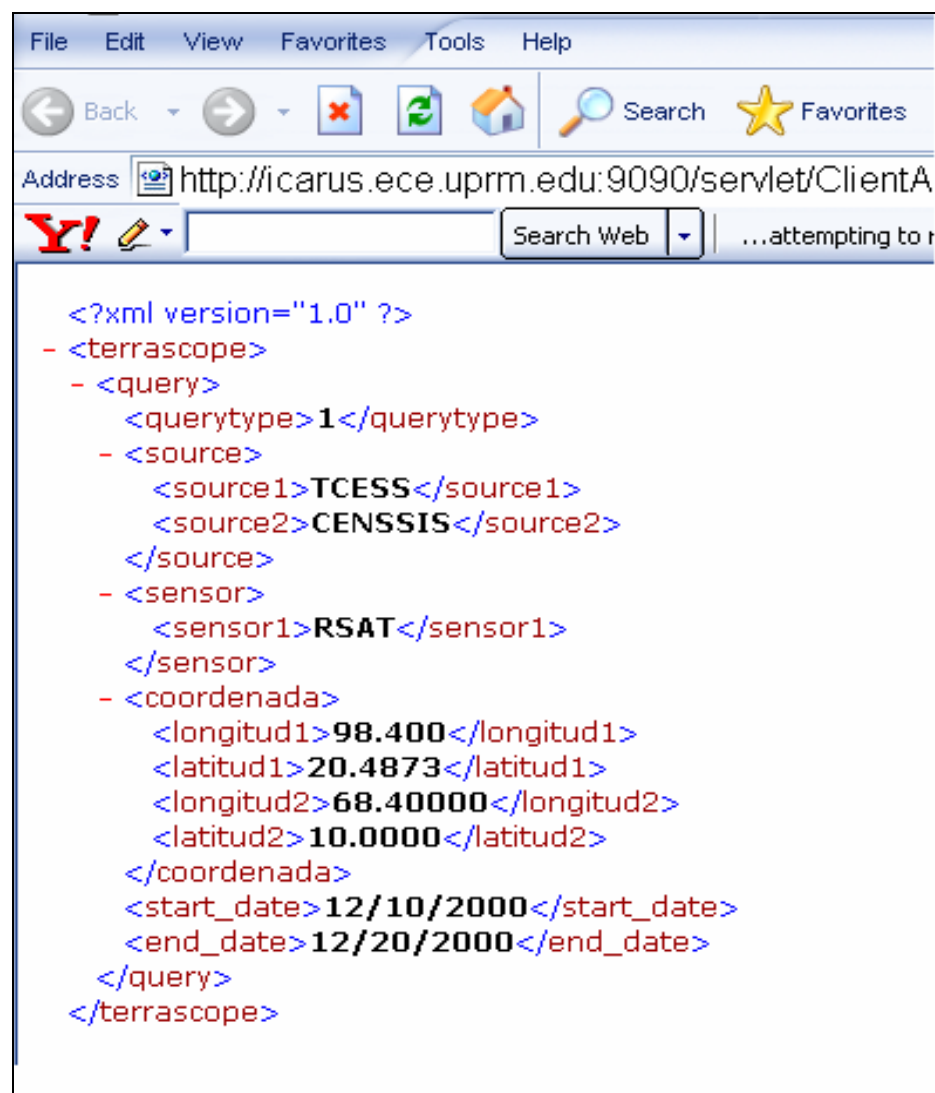


Figure 3-12 Schema of the Request in XML format

3.3.2.2 Schema of the Response in XML

Figure 3-12 shows a format that we have developed for responses to queries. This format is a XML document containing all the query results found in either local or remote databases.

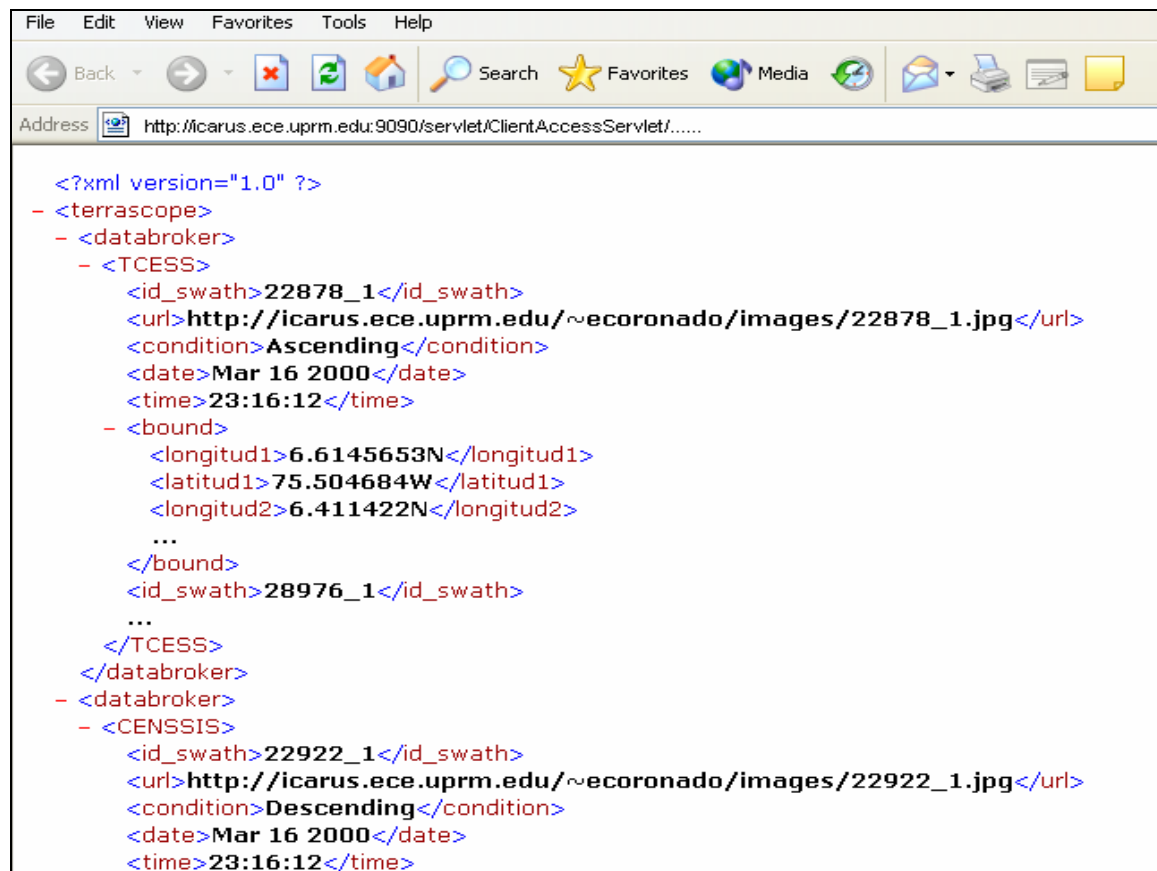


Figure 3-13 Schema of the Response in XML format

The body of this XML document is composed of important data for the user such as a url where the images are stored, satellite status (ascending or descending), creation date, time, images coordinates, source, and other data that will be sent to the Client Application who will parser this format, extract the data and metadata, and show these results to the user.

Chapter Four

Peer – to - Peer Architecture

This chapter presents the peer-to-peer architecture used by our SRE information system. First we have an overview about of what is a Peer – to – Peer architecture, a comparison with the conventional systems, and finally we presents our Decentralized Peer – to – Peer Architecture.

4.1 Overview of what is Peer – to – Peer (P2P) System

Peer-to-Peer (P2P) is a model that promotes a global system that allows the direct collaboration between computers with emphasis in the capitalization and show of resources. For defining *P2P* architecture is important to first comment what is *client/server* model (**Figure 4 – 1**). It is the more common model for communication in Internet, where there exist a client that knows how to realize a request of information, and how to put this request on a server. On the other hand, the server knows how to perform some service (e.g. provide a weather forecasts), and how to give the results to the client.

This model has two entity of execution:

- Server, it offers some service X.
- Client, it uses the service X.

The interaction occurs under the form of exchange of two messages: i) Request, specification of the requirement services, parameters, others; ii) Response, results or indicator of some error.

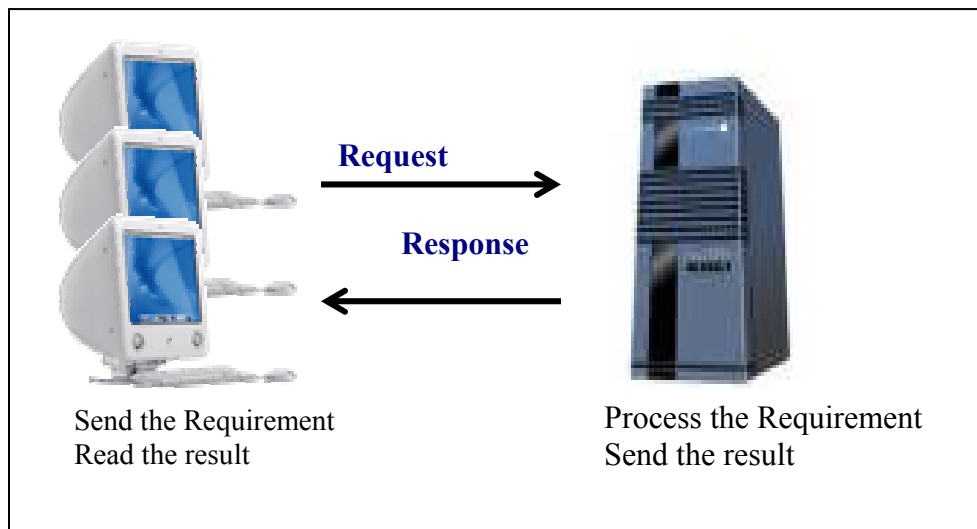


Figure 4-1 Client - Server Model

When we talk of Peer - to - Peer system we refers to a type of communications exchange in which any given applications can act as: a) client that uses a service form other server, or b) a server that provides some service. Thus, each application has a dual role as client or server, and can switch from these roles during normal operating conditions. For a given application X, its peers are the other applications that provide service to X. In turn, peers of X can receive services from X itself. This is what characterizes peer-to-peer systems. Peer-to-Peer systems are decentralized by nature, since every peer to start a data processing operation. There is no central authority that controls query execution. The Internet Domain Name Service (DNS) is a Peer-to-Peer architecture that has demonstrated a capacity for auto-structure and scalability.

The use of this architecture is important because, it enables a computer to behave as a client or server in any moment and establish a direct connection with one or more peers. All nodes in this model can be client or server of services at same time.

4.2 Comparison between Conventional Distributed Systems and Peer – to – Peer (P2P)

The conventional distributed systems have one centralized and dedicated infrastructure, central administration. The clients rely on this central site to consume services; making it difficult to extend the system, because of the possibility of restriction by the central administration.

A distributed system featuring ease-of-evolution was desired, where all the responsibility is not deposited on a centralized infrastructure. There is where the necessity to implement a P2P system arises. P2P provides users with a decentralized infrastructure, devoid of problems with a central administration.

A P2P application is different to client – server model because in the latter, the roles of client and server are fixed in the application's logic. In contrasts, in P2P systems the application can act as a client or server, it have capacity to ask information to others peer (servers), and also have the ability of acting like a server to service the requests of the others clients.

4.3 SRE's Peer – to – Peer Architecture

Currently, there exist different systems that search information about of the satellite images that are stored in different data sources. But these systems are complex because they are not in an integrated system. Moreover, most of these previous systems work with a centralized architecture which also represents a performance bottleneck due to the excessive movement of data to the central site.

The architecture that we proposed in this thesis is a decentralized *Peer - to - Peer Architecture*, that allow us to obtain information from local or/and remote databases at the same time. Our SRE is built around this architecture. In our system the clients can customize their own view of the system to define the remote data and computational

services they wish to access. In the **Figure 4-2**, we show a decentralized peer-to-peer architecture that enables SRE to integrate four different data sources: CenSSIS UPRM, CenSSIS RPI, NASA and TCESS, all of which are located in different sites.

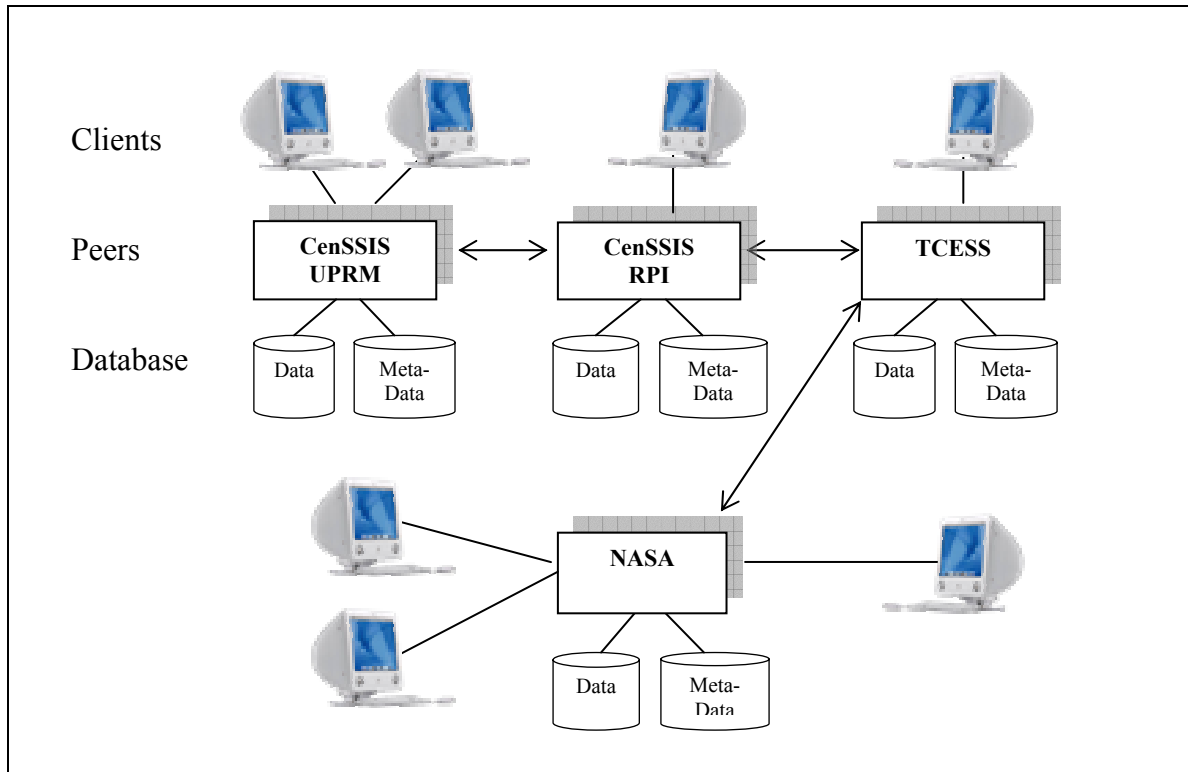


Figure 4-2 Peer to Peer Architecture

Each one of those peers has installed our SRE as part of its web-server, and each peer acts like client or server when searching information about some images. Ours peers are capable to work like clients when sending a request to others peers and also can be capable of working like a server when processing the requirement and sending the results.

In our architecture there are some important issues such as:

i) That a query request “X” can be received by a peer site more than once and we do not want to reply more than once; Our solution to this issue is that each peer gives an *ID*

to query request. Each peer needs to keep track of these IDs, and reject request with IDs already seen. In our scheme, we call this ID the *Id_Query*.

The *Id_Query* contains the URL or IP address of the CAS servlet that originally received the query from the client, plus a sequence number assigned for each request received. The *Id_Query* is stored in a table created in each database site running the SRE. For this we have created an *IdGenerator* class (**Figure 4-3**) that contains five (5) different methods that allows us opens the database, create a new *Id_Query*, and save this in the database.

```
package edu.uprm.adm.imagegrid.server;
public abstract class IDGenerator
{
    public static Database openDB(HttpServlet sl)
    public static String nextID (HttpServlet sl)
    public static boolean containsID( HttpServlet sl, String id ) throws
        Exception

    public static void saveID (HttpServlet sl, String id)
    public static void purge(HttpServlet sl)
}
```

Figure 4-3 IdGenerator Class

For a better understanding of our scheme for managing *Id_Query*, we provide a scenario of communication in our SRE:

First, the client or user sent a request to our system. After that client establish a connection with the first servlet, *Client Access* (CAS), and sent its request, CAS assign a new *Id_Query* number for this request that allow us to have a better control of all the request that enters, avoiding executing the same request more than two times, and send

forward with the same request plus the *Id_Query* assigned to next servlet, Data Broker Servlet (DBS).

To realize this communication, we had created another class called *Setting*, shown in **Figure 4 – 4**, inside a package *server* that contains the method that allows us to extract of each of the tag of the address that corresponds to the next servlet as follows:

```
package edu.uprm.adm.imagegrid.server;

public abstract class Settings
{
    public static final String filename = "/terrascope.xml" ;

    public static String get (HttpServletRequest sl , String tag)
    {
        String path = sl.getServletContext().getRealPath(filename);

        return TSParser.parse( path , tag );
    }
}
```

Figure 4-4 Setting Class

Next, the DBS receives the user's request, it will solve the query with local and distributed data; but first, it verifies if this request was attended before by comparing if the *Id_Query* that it have received is already stored in its database. If this *Id_Query* exist, then it will send a message that indicates that this request was attended before and it will not attend this again. Otherwise, if this query was not attended before then it save this new *Id_Query* in its own database, and send the request forward to next servlet, the Information Gateway Servlet (IGS) in order to extract the local data from the local database.

In the **Figure 4-5**, we show where TCESS receive a new request sent by the user.

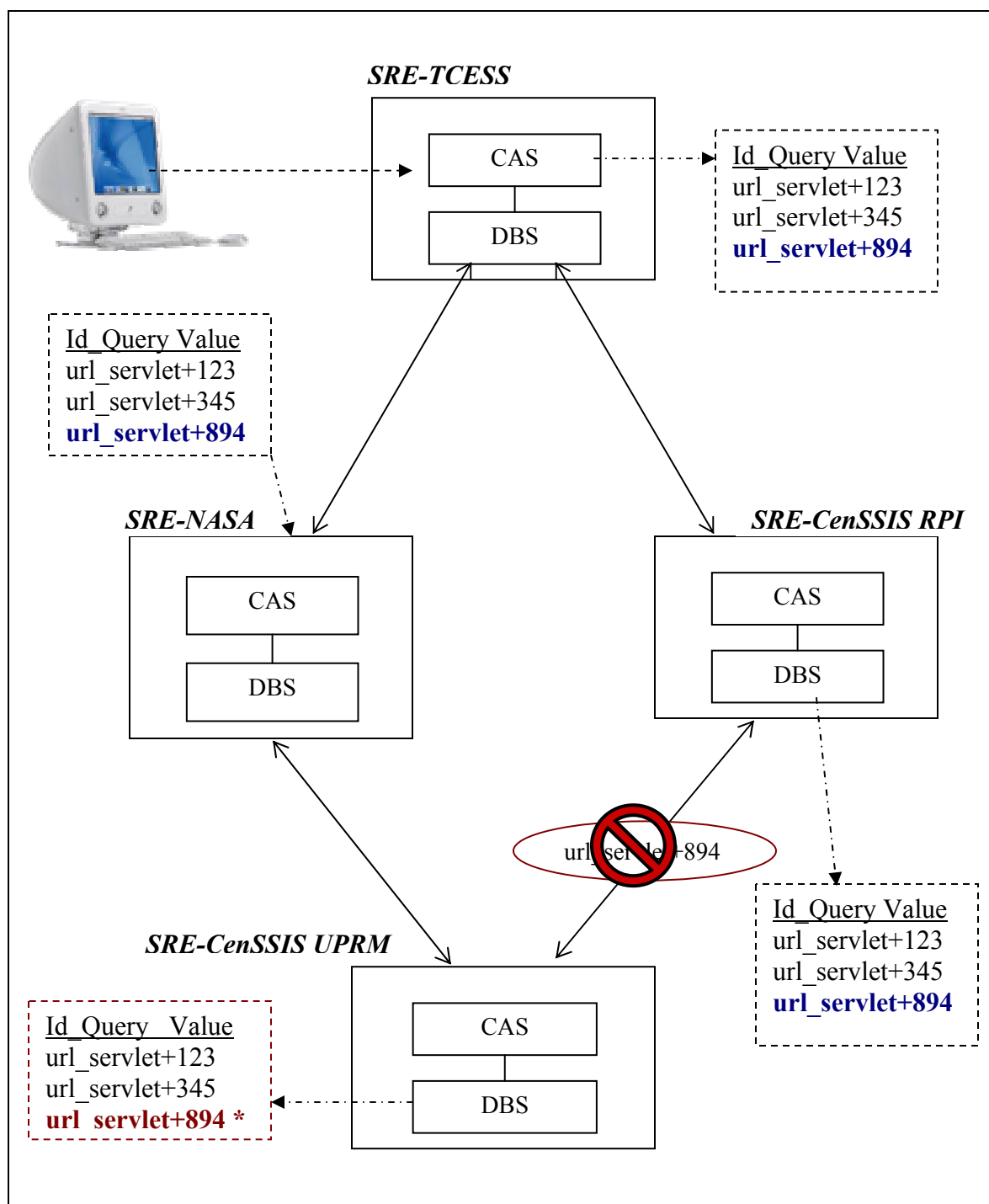


Figure 4-5 Id_Query process

Then, CAS creates and save a new *Id_Query* in its database and DBS forward the same request to others peers with the *Id_Query* generated. Each peer attends the request and store and mark *Id_Query* as attended.

But for example, TCESS receive a request of the client; TCESS' CAS verify if this request is new or not, like is new then assign a new *Id_Query* for this request and TCESS forward the same request plus *Id_Query* to others peers (friends) that this know. In this case, TCESS send the request to NASA and CenSSIS RPI, each search in its database if this *Id_Query* was attended before, if not was attended then each stored this new *Id_Query* and attend the request, and marked it as attended.

The same happens when NASA forwards the request to CenSSIS UPRM, but when CenSSIS RPI send the same request again to CenSSIS UPRM then it is not attended again because this request was marked as attended.

ii) Other issue is that we need to control the number of hops that is the number of peers that we will sent the same request. Our solution is to keep a counter on the message indicating how many times it can be forwarded. For this we created a counter called *Time-to-Life* (ttl). Each server (peer) has to check if its *time - to - life* (ttl) is not expired (i.e. reached 0); if *ttl* is not expired, then it attends the request and decrement *ttl* by one ($ttl = ttl - 1$) and it will connect to the next servlet, *Information Gateway Servlet* (IGS), to search information and it will forward the request to the others peers. After that, the Data Broker Servlet will forward the query to its peers in order to get results from them. The results from the remote sites are merged with the results from the local database.

The *ttl* is refereed to the number hops that a server has to search in others peers. In other words, if we said that the first server has a *ttl* of three then it means that the server will search in three levels, not three peers or servers. In this case (shown in **Figure 4-6**), it will only search until level three. This *ttl* can change according to the configuration type that the administrator assigns.

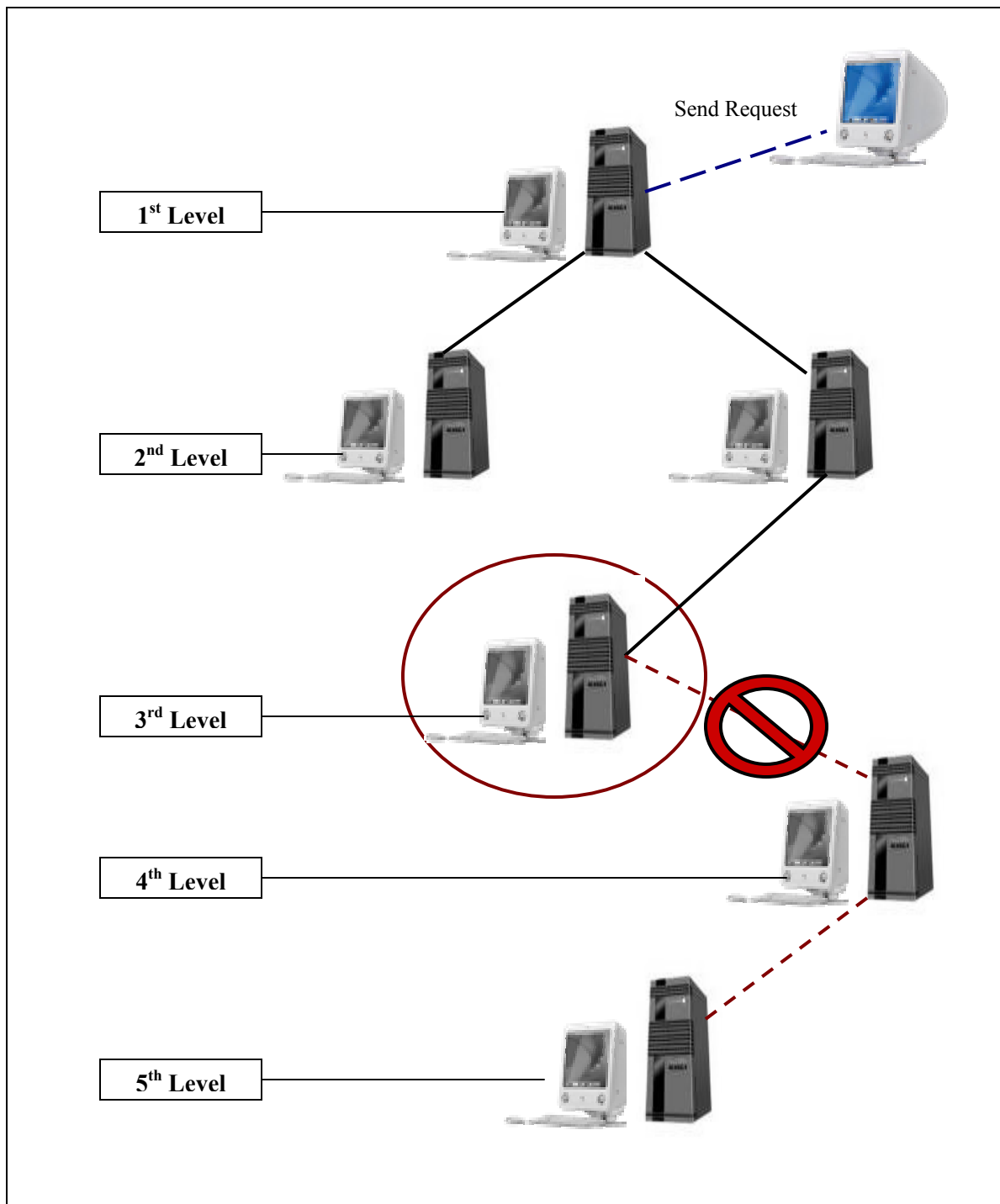


Figure 4-6 Times - to - Life (ttl)

To make this possible, our Data Broker Servlet Class has a group of methods that are show in the **Figure 4-7**.

```

package edu.uprm.adm.imagegrid.server;

public class DataBrokerServlet extends HttpServlet
{
    public void init( ServletConfig config ) throws ServletException
    public void destroy()
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, java.io.IOException
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, java.io.IOException
    protected void processRequest( HttpServletRequest request , HttpServletResponse response )
        throws ServletException, java.io.IOException
    private boolean alreadyAttended( TSQuery tsq )
    private String acceptRequest( TSQuery tsq )
    private String denyRequest( TSQuery tsq )
    private void markAsAttended( TSQuery tsq )
    private String getRequestFromMyGW( TSQuery tsq )
    public URLConnection getURLConnection( String source ) throws Exception
    public Vector getRequestFromDataBrokers( TSQuery tsq )
    public String getRequestFromDataBroker( TSQuery query , String source )
    public void sendResponseToClientAccess( HttpServletResponse response , String tsrl )
}

```

Figure 4-7 Data Broker Servlet Class

Finally we have the next servlet; *Information Gateway Servlet* (IGS) who receive the request and extract the data that contributes to the solution of the query. It allows us to access its database locally to search the information required by the user. In the **Figure 4-8**, we show methods that contain Information Gateway Servlet and that make this search possible.

```

Package edu.uprm.adm.imagegrid.server;

public class GatewayServlet extends HttpServlet
{
    public void init( ServletConfig config ) throws ServletException
    public static Database openDB(HttpServlet sl)
    protected void doPost (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, java.io.IOException
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, java.io.IOException
    protected void processRequest(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, java.io.IOException
    public TSQuery getRequestFromDataBroker(HttpServletRequest request)
    public TSResultSet getRequestFromDatabase(TSQuery tsquery) throws Exception
    public void sendResponseToDataBroker( HttpServletResponse response, TSResultSet
        resultset)
    private TSResultSet queryToResultList(Query query,TSQuery tsquery) throws Exception
    private TSResultSet parsePolygon (String bounds)
    private TSResultSet queryToResult( Query query , TSQuery tsquery ) throws Exception
}

```

Figure 4-8 Information Gateway Servlet Class

When the query is solved, it packs the result and sends this to its local DBS who verifies the results and merges these local results with all the results obtained from the others peers.

Each DBS that received a request will connect to other remote peer DBS to ask them for this data. Each remote DBS will do the same independently: ask its local IGS to get data and also talk to other peer DBS.

Our P2P solution has the following goals:

- It is possible add new peers to the system without central control. Each peer can communicate with others peers or servers to search information.
- To easily share content with other partners. It can share information after it has been discovered.

The benefits of the P2P Architecture are to be a single point of access to the system where the growth system can be adding sites and metadata.

Chapter Five

Experiments and Results

5.1 Introduction

We conducted some experiments to show the benefits of our Peer-to-Peer SRE system versus previous centralized systems. To demonstrate that the speed to process a query in a decentralized architecture is better and faster, we had executed some tests where we run the same queries on both types of architectures. Our experiments show that SRE provides a more efficient architecture than previous solutions.

5.2 Methodology

The methodology used in the development of the project SRE is divided in equipment and tools used.

5.2.1 Equipment

We used four servers: three (3) that run under Linux and other that run under Windows XP where we installed our SRE. The descriptions of our equipment are:

- Three servers (3) under Linux:
 - **Host Name:** **icarus.ece.uprm.edu**

<i>MachType</i>	i686-pc-linux-gnu
<i>OSType</i>	Linux-gnu
<i>Host Type</i>	i686
<i>Memory</i>	512 MHz
<i>Speed_CPU</i>	930 MHz

- **Host Name:** **crl2.ece.uprm.edu**

<i>MachType</i>	sparc-sun-solaris2.9
<i>OSType</i>	Solaris 2.9
<i>Host Type</i>	sparc
<i>Memory</i>	1GB
<i>Speed_CPU</i>	440 MHz

- **Host Name:** **db3.ece.uprm.edu**

<i>MachType</i>	i686-pc-linux-gnu
<i>OSType</i>	Linux-gnu
<i>Memory</i>	128
<i>Speed_CPU</i>	500 MHZ

- 1 PC – DELL

- **Host Name:** **llws04.ece.uprm.edu**

<i>RAM</i>	1.0 GB
<i>Processor</i>	Pentium 4
<i>OSType</i>	Windows XP
<i>Speed_CPU</i>	2.4 GHz

- A TCP/IP communication network of 100 Mb/s
- Clients Applications (12) - To realize our test we used twelve workstations installed in the laboratory Amadeus of the Electrical and Computer Engineering Department that are identify with different Caciques' name just as: Dehostos, Betances, Cofresi, Lascasas, among others. Each one of the workstations has the following features:

- PC – DELL

<i>RAM</i>	2.0GB
<i>Hard disk</i>	20GB
<i>Processor</i>	Intel Pentium 4
<i>Speed</i>	2.4GHz
<i>OSType</i>	OS Windows XP.

5.2.2 Tools Used

Among the tools used for realized our SRE project we had considered the follows:

- Sun ONE Studio 4 EC
- Java Version 1.4.0
- Java TM Web Services Developer Pack 1.0.
- JDK 1.2 or bigger, to be able to compile the classes and to execute the server.
- PostgreSQL 7.3.2 for UNIX and vs.7.2.1 Windows XP.
- PgAdminII vs. 1.4.2. It is a graphic tools to work with PostgreSQL under Windows and easy used and manage.

Our project was developed in Java Servlet because they are more efficient, easier to user, more powerful, more portable, safer and cheaper than traditional CGI and many alternative CGI like technology.

- Efficient, because with the use of the servlet, the java virtual machine stays running and handles each request with a light weight java thread. With servlet there would be N threads but only a single copy of the servlet class would be loaded.
- Convenient, because this technology makes for more reliable and reusable code than does Visual Basic, C++ among others.
- Powerful, because it can to talk directly to the web-server.
- Portable, because it is written in the java programming language.

5.3 Development of the Experiments

To continue we show the realization of our experiments that was based in both architecture types: centralized and decentralized peer-to-peer architecture in order to measure the *execution time* to process a request.

5.3.1 Centralized Architecture

In the **Figure 5-1** we have TCESS as a central server, and which only this server has a catalog with the location of others peers such as: CenSSIS UPRM, NASA, CenSSIS RPI; the others server do not know themselves.

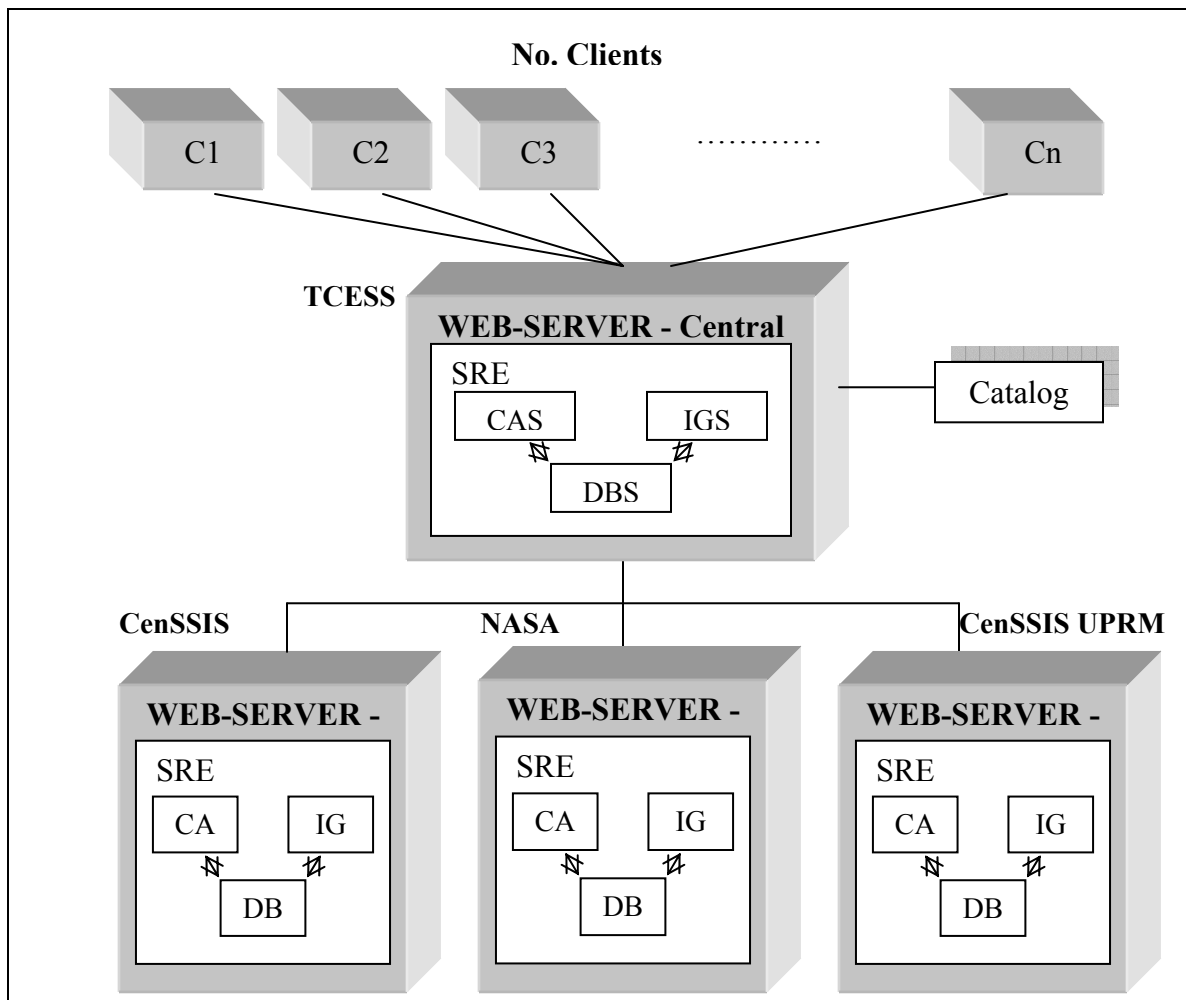


Figure 5-1 Schema to process request in Centralized Architecture

Cn is the clients' number of each client being used. For this architecture we had done some test where we connected twelve clients to TCESS central server. Each client sends a query to system, and we measure the time it takes to get the answer. We repeated this experiment four times and we show average values. These results are shown in the following table: (**Table 5–1**)

C=1	C=2	C=3	C=4	C=5	C=6
2216	3820.00	4481.33	6920.25	7546.20	7363.83
2135	3650.00	4235.67	5913.25	5967.40	7169.00
2419	3529.00	3940.67	5791.25	5278.40	8416.83
2035	3844.50	3833.33	6001.00	5930.20	8146.50
2,201.25	3,710.88	4,122.75	6,156.44	6,180.55	7,774.04

C=7	C=8	C=9	C=10	C=11	C=12
8707.43	10592.25	11478.00	11196.90	13464.82	13728.50
10278.43	9927.75	10651.33	12853.10	12591.18	17135.17
8792.14	9214.88	11622.11	13179.60	14421.55	15800.75
12168.14	11548.63	13813.44	14845.50	14792.91	15857.67
9,986.54	10,320.88	11,891.22	13,018.78	13,817.62	15,630.52

Table 5-1 Result obtained using a Centralized Architecture

It can be seen that when added a new client the execution time to process the same query is increases; for example the time average that takes one client to search information about of a query determined was 2,201.25 millisecond that equivalent to 2 seconds; but if we have twelve clients concurrent making the same query, the time average increase to 15,630.52 milliseconds that equivalent to 15 seconds. To continue, we shows a plot in the **Figure 5–2** that corresponds to the previous table where each one of the results obtained is showed.

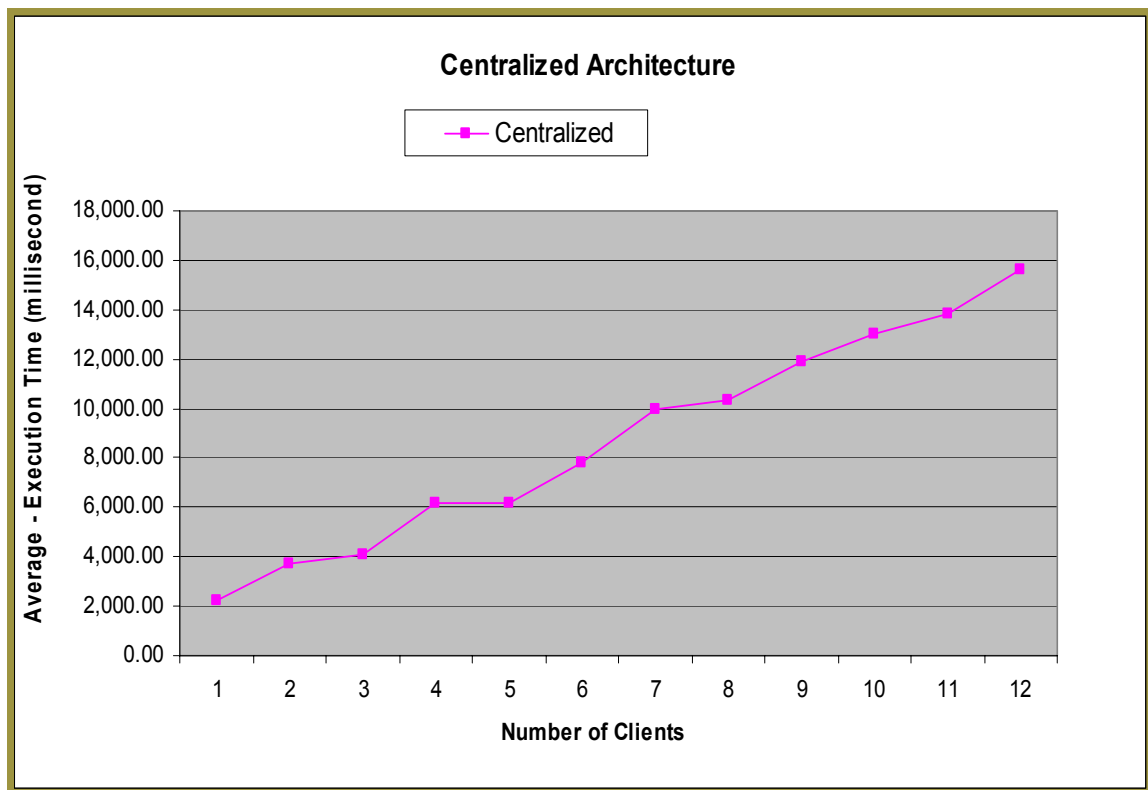


Figure 5-2 Results of a Centralized Architecture

We can see that we obtained an almost lineal graphic, where the average time to process a query executed with twelve clients concurrent was near of 16 seconds approximately. We can see that each time that we increment a number of the clients the average time to process a query increases linearly.

Features of Centralized Architecture

- There exist only one main server that contain a catalog with all the address of others server.
- There exists only one central point to access the entire system.
- Each request is sent by the central server to each of the remote database servers, causing that the transfer of information but in a very inefficient way.

5.3.2 Decentralized Architecture

We used various peer configurations in our decentralized architecture with the same servers that are located in different places, to demonstrate that the results obtained are, in general, better than the results obtained in a centralized architecture. Each server is capable of receiving queries, processing them, connecting with other peers they know to get more information, and collaborating to give the final results to the client.

In the **Figure 5-3** our Decentralized Peer-to-Peer Architecture has the same peers and the same number of concurrent clients, and the structure in this case is ring type. This architecture allows us to have many concurrent clients connected to different peers.

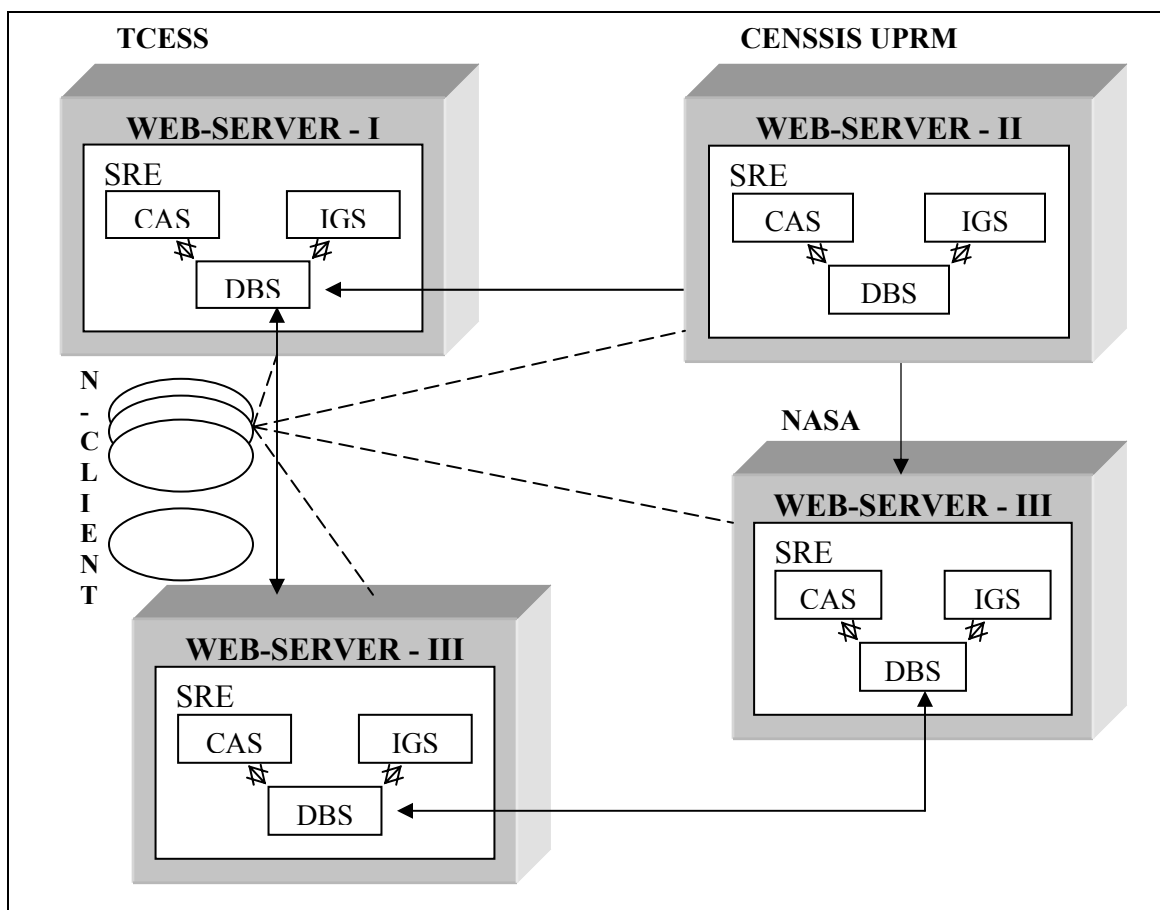


Figure 5-3 Schema to process request in Decentralized Architecture Peer - to - Peer

In this schema we have the same clients connect to different servers, where:

- The same request is send to each server where is connect.
- The Execution Time that each server takes for resolve the request is measured.
- Each experiment was repeated four times, and the results shown are averages of these values.

In the **Table 5-2** we show the query execution result obtained in this test, and we can see that the results are better that those obtained in the centralized architecture.

For example, when one client is connect to a peer the execution time was of 2,047.50 that equivalent to 2 seconds, but when we used more clients (twelve) in an architecture like this, then we get an average time to obtain the information of 6,911.06 milliseconds that is equivalent to about 7 seconds.

C=1	C=2	C=3	C=4	C=5	C=6
1921	2407.50	3369.33	2931.25	3341.20	4300.33
2275	2314.50	3289.33	3155.50	3876.40	5025.55
1979	2372.00	3506.00	3156.25	3866.00	4378.00
2015	2882.00	3081.33	2904.75	3709.80	4151.17
2,047.50	2,494.00	3,311.50	3,036.94	3,698.35	4,463.76

C=7	C=8	C=9	C=10	C=11	C=12
5023.71	5610.88	4991.11	5589.00	6700.64	7263.67
4519.00	4850.50	5444.67	6221.10	6489.82	6575.08
5173.71	4929.75	5293.56	6142.40	6432.09	6645.08
5216.00	4858.00	5171.22	6036.50	6212.64	7160.42
4,983.11	5,062.28	5,225.14	5,997.25	6,458.80	6,911.06

Table 5-2 Result obtained using Decentralized Architecture

For a major understanding we show the **Figure 5-4** where each one of the results obtains in this architecture are presented in a graph.

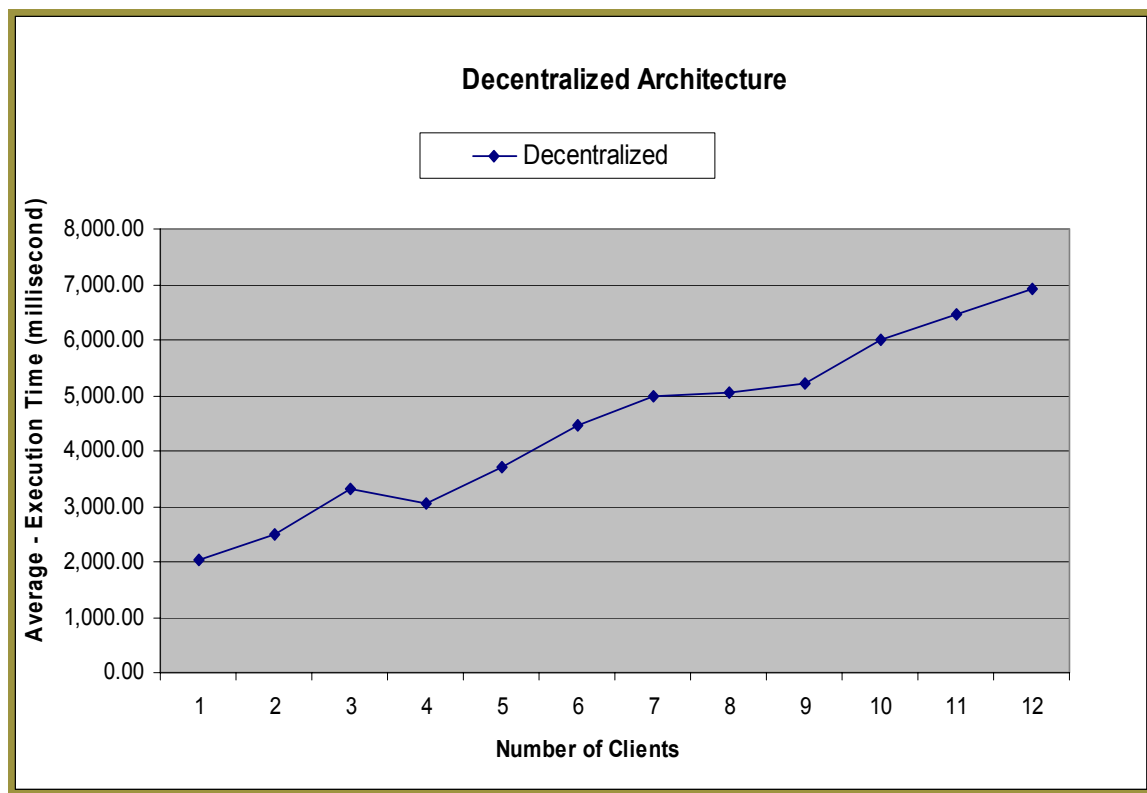


Figure 5-4 Results of our Decentralized Architecture Peer-to-Peer

We can see that the Execution Time in our Decentralized Peer – to – Peer Architecture was smaller (hence better) than the one obtained in a Centralized Architecture, using the same query in both cases and the same number of clients.

When the request is sent for one client the execution time for this process is approximately of 2 seconds; but when there are connected twelve clients to the different servers the execution time for this process is greater but minor with respect to the centralized architecture.

To continue, we show the **Figure 5-5** where we make a comparison between both architectures. We can see that the time of execution differ totally, and the P2P decentralized system solves the queries in less time than the centralized one.

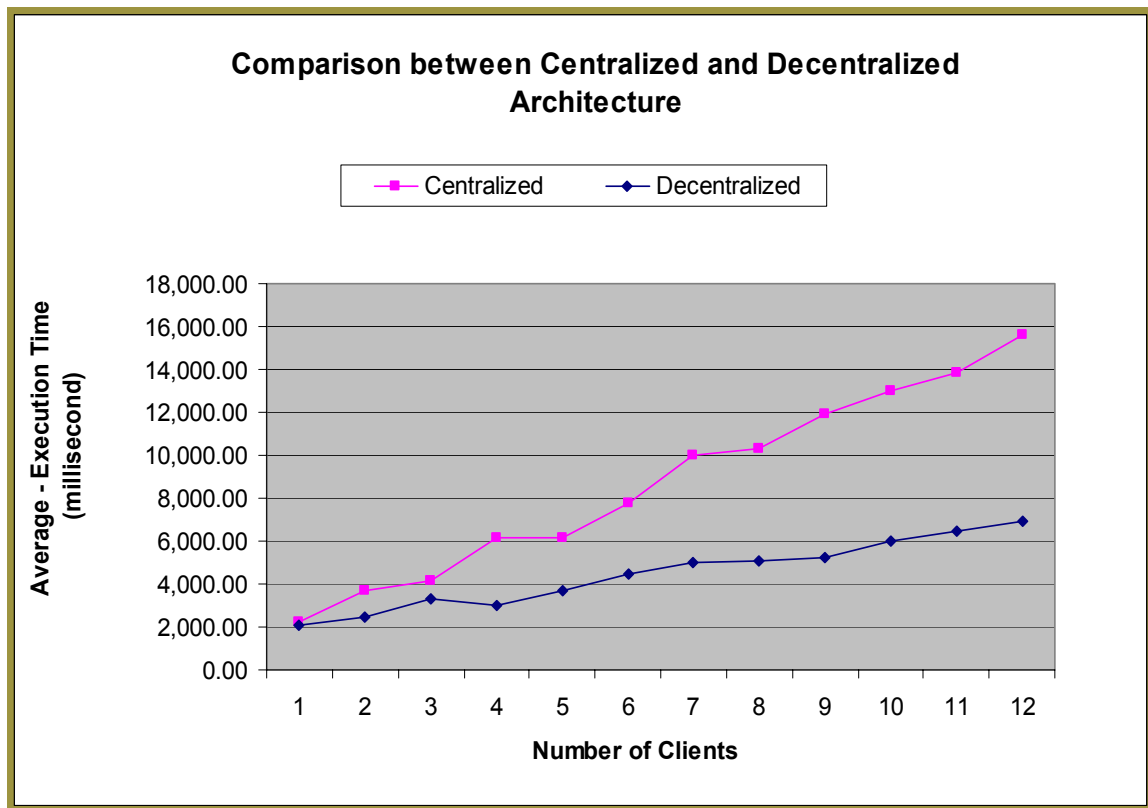


Figure 5-5 Results obtains between both Architectures

When we compare the results from the **Tables 5-1** and **5-2**, we can see that the average execution time between both architecture differed in nearly of 44%. This is demonstrated with the follow equation:

$$\frac{\text{AVG Max (decentralized)}}{\text{AVG Max (centralized)}} \times 100\% = X$$

Then, we have that in a centralized architecture the execution time that took was of 15,630.52 milliseconds (about 16 seconds) and our decentralized architecture the execution time was 6,911.06 milliseconds (about 7 seconds). It is to say:

$$\begin{array}{lcl} \text{If} & 16 \text{ sec} & \text{—————} 100\% \\ & 7 \text{ sec} & \text{—————} X \end{array}$$

$$\text{Then, } X = \frac{(7 * 100)}{16}$$

$$16$$

$$\boxed{X = 43.75 \%}$$

Then we can say that our architecture was near of 44 % more efficient and faster in process a request, which the average time took in the first architecture.

Also, we can see that in the range of six (6) to twelve (12) clients connects concurrently, the time to process one request incremented near of 8 seconds more in a centralized architecture; but we can see that in our decentralized architecture the time to process the same request in the same range, incremented approximately 2 seconds.

Next, we show in the **Figure 5-6** other structure or peer configuration for our decentralized architecture that indicate us that we can configure or distribute the SRE peers in different manners. We realized the same experiments, with location different where demonstrate us that the results obtained are better that to use a Centralized Architecture. Thus, SRE can be configured in different ways, and can get excellent performance.

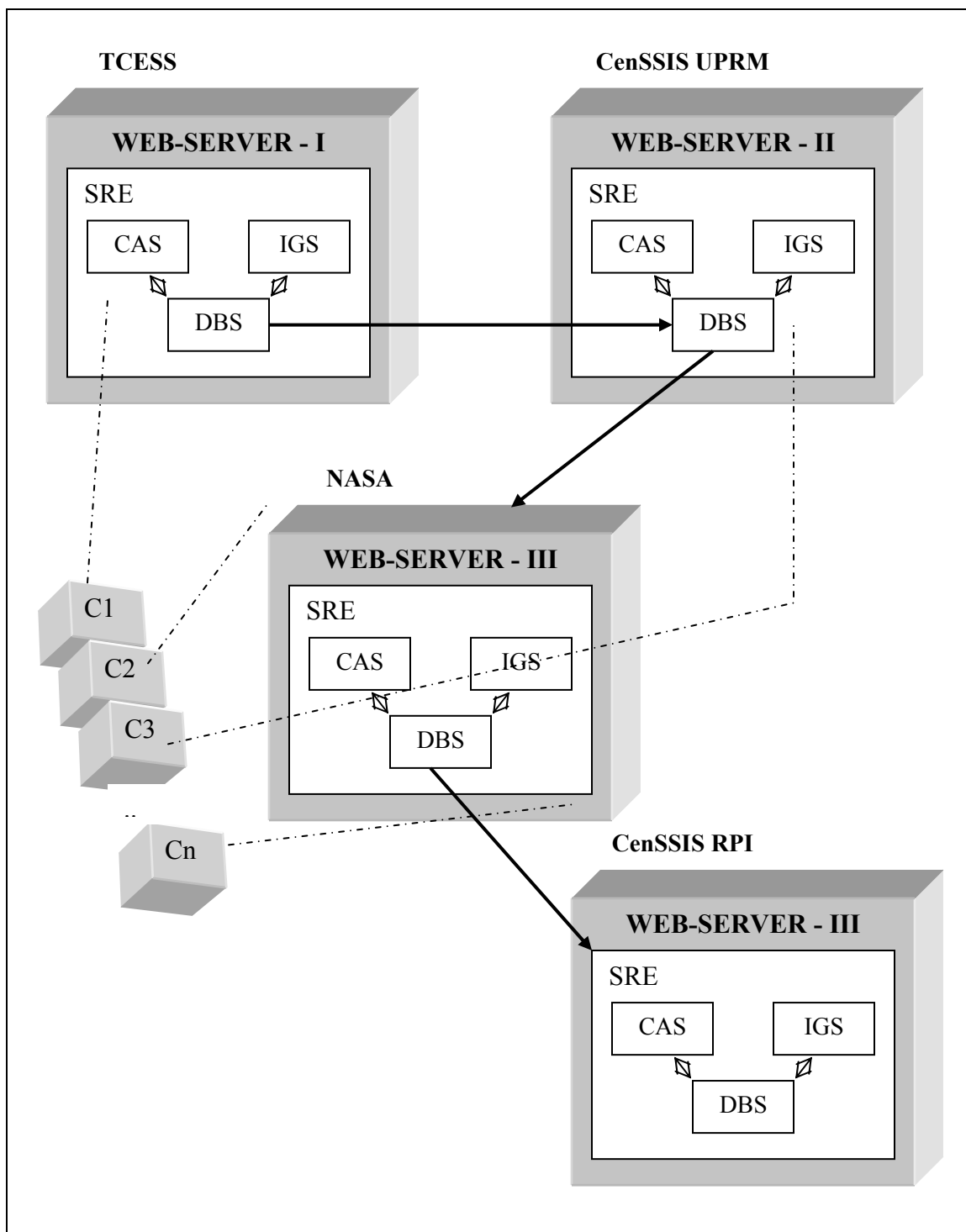


Figure 5-6 Second structure of Decentralized Architecture

In this structure we have the same clients; with the same servers but each server have a different distribution. The connection is not ring type like the first case. With this structure, we realized the same test where obtained the follows results:

C=1	C=2	C=3	C=4	C=5	C=6
2,232.00	3,077.50	3,202.00	3,823.75	4,692.60	5,731.33
2,404.00	3,085.50	3,036.67	3,891.25	4,540.40	5,404.33
2,348.00	3,097.50	3,248.67	4,131.25	4,363.80	5,123.67
2,308.00	3,135.00	3,255.33	4,286.75	4,449.60	5,634.00
2,323.00	3,098.88	3,185.67	4,033.25	4,511.60	5,473.33

C=7	C=8	C=9	C=10	C=11	C=12
6,116.43	6,680.38	7,658.44	8,370.90	8,524.64	8,686.42
6,148.29	6,563.25	7,851.56	8,038.00	8,922.82	9,304.58
6,128.57	6,764.38	7,473.67	8,176.50	8,530.00	8,913.83
6,466.14	6,391.25	7,095.78	8,379.20	8,237.91	9,034.92
6,214.86	6,599.82	7,519.86	8,241.15	8,553.84	8,984.94

Table 5-3 Result obtained using a second structure of Decentralized Architecture

We can compare the above results with the first structure and can see that the results are different, but also better than the Centralized Architecture. To continue we show in the **Figure 5-7** the graphic obtain that corresponding to **Table 5-3**.

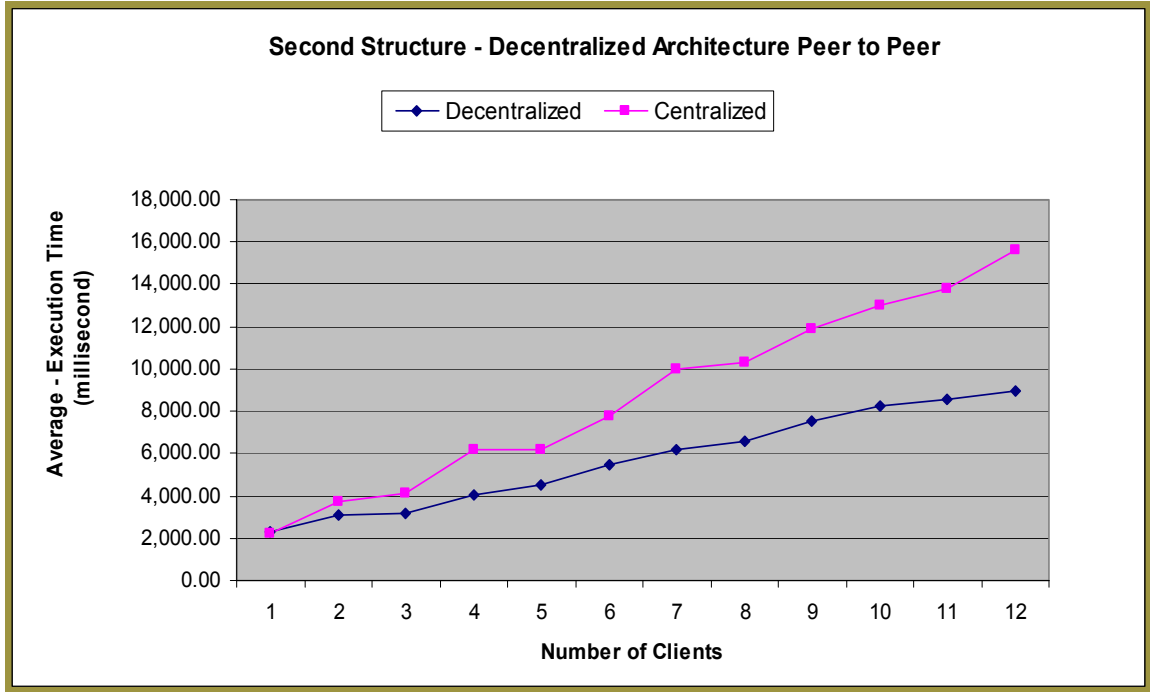


Figure 5-7 Result obtained with the second structure of a Decentralized Architecture

We can see that the Execution Time obtained in this second structure of a Decentralized Peer-to-Peer Architecture was faster than the one obtained in a Centralized Architecture. However, it used a little more of time that the first structure because in this structure each peer know the existence only one other peer. Notice than in this structure we used the same query with the same number of clients as in previous experiments.

To continue, we present a third structure (**Figure 5-8**) for the scenario, where we used the same servers but with other distributing of the peer sites.

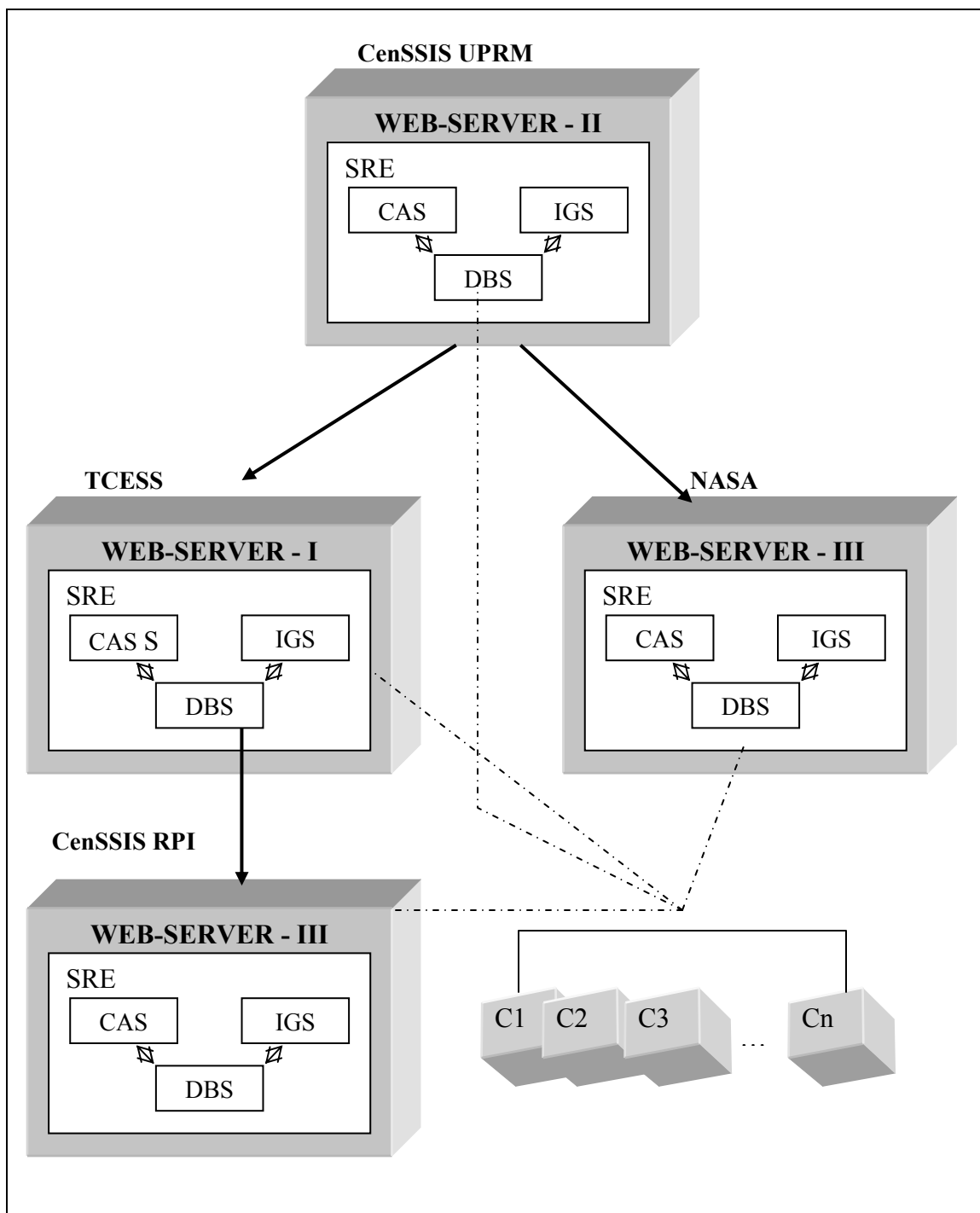


Figure 5-8 Third structure of a Decentralized Architecture

In this figure we can see that scheme changed; now there exist one server that know two peers, other two that only know the existence of one peer. For this structure we also realized the same tests with the same workstations or clients, and the same query as before. The query execution times results obtained were as follows:

C=1	C=2	C=3	C=4	C=5	C=6
2,322.00	2,379.00	2,958.00	3,096.00	3,177.00	3,680.00
2,125.00	2,177.00	2,472.33	3,551.50	3,701.20	3,126.00
2,410.00	2,020.50	2,810.67	3,409.50	3,456.40	3,783.83
2,200.00	2,155.50	2,713.00	2,741.50	3,682.00	3,909.83
2,264.25	2,183.00	2,738.50	3,199.63	3,504.15	3,624.92

C=7	C=8	C=9	C=10	C=11	C=12
4,307.14	4,158.75	4,673.44	4,546.40	5,479.73	5,586.42
3,980.57	3,748.88	4,770.89	4,435.40	4,999.36	5,555.92
4,196.43	4,158.75	4,323.78	4,949.70	4,845.27	5,169.00
3,804.57	4,288.25	4,052.33	4,681.70	5,325.91	5,438.42
4,072.18	4,088.66	4,455.11	4,653.30	5,162.57	5,437.44

Table 5-4 Result obtained using a third structure of Decentralized Architecture

We can compare the above results with the first structure and can see that the results also are different, but also is better that to use a Centralize Architecture. The time to response the same request was lower that the obtained in the first and second decentralized architecture. This expressed in percent was near of 31.25%. To continue we show in the **Figure 5-9** the graphic obtain that corresponding to **Table 5-4**.

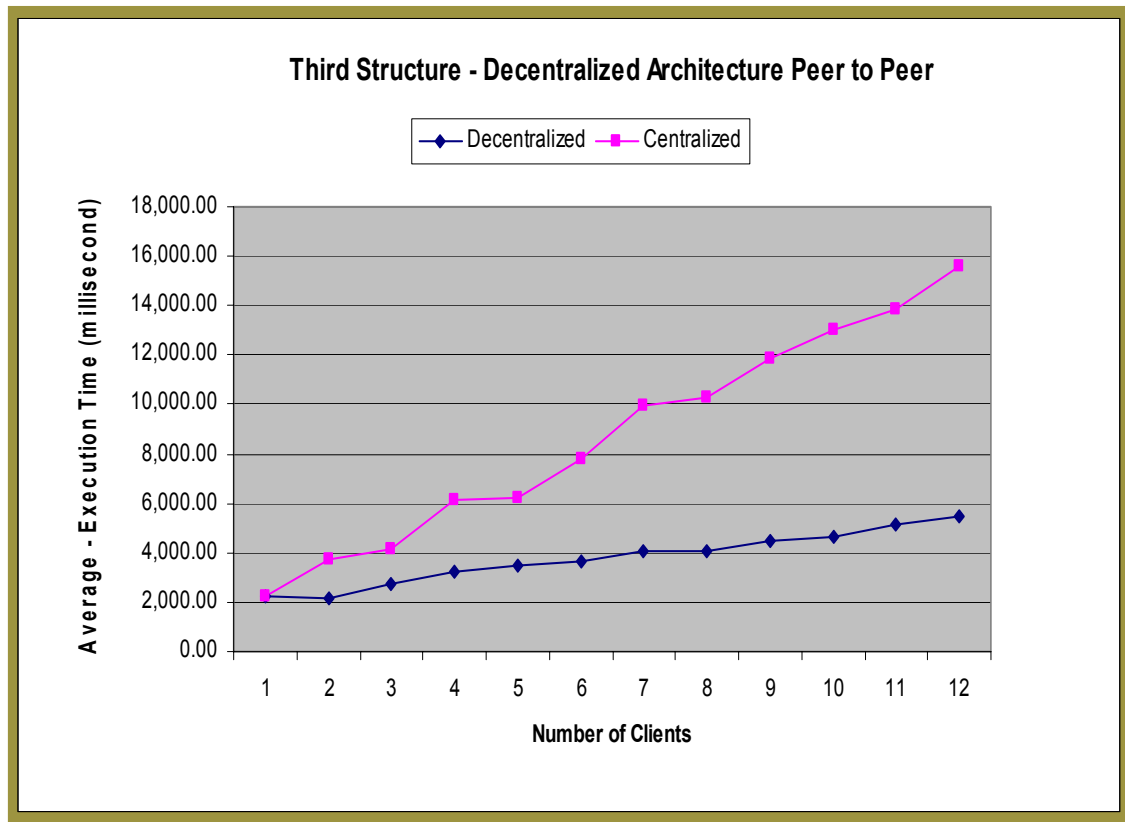


Figure 5-9 Result obtained with the third structure of a Decentralized Architecture

Also we can see that the Execution Time obtained in this third structure of a Decentralized Peer – to – Peer Architecture was faster than the obtained in a Centralized Architecture. Also the time of the response was faster than the first and second decentralized architecture.

Finally, we realized other fourth structure with a design of distributed similar to third structure. This is shown in the **Figure 5-10**.

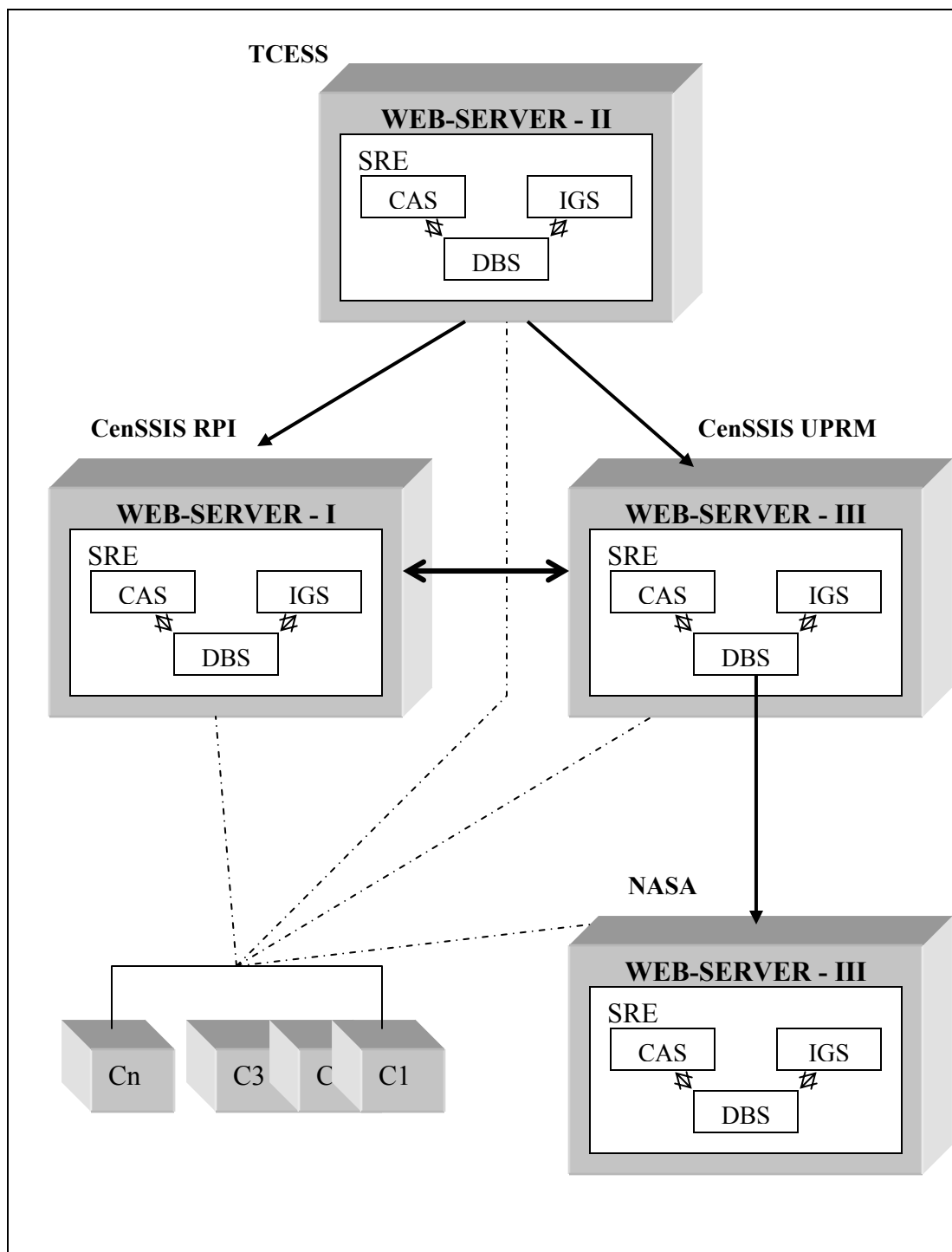


Figure 5-10 Fourth structure of Decentralized Architecture

In this figure we have the same servers, the same workstations or clients connects, but with the different structure. Here, also we realized the same test where obtained the follows results:

C=1	C=2	C=3	C=4	C=5	C=6
2,988.00	2,888.00	3,492.00	4,178.00	4,761.20	4,537.67
2,650.00	2,902.50	3,748.67	4,202.00	4,057.40	5,177.33
2,730.00	2,903.50	3,685.67	4,014.50	5,143.40	5,690.33
2,728.00	3,356.00	3,392.00	4,140.50	4,290.60	4,978.50
2,774.00	3,012.50	3,579.59	4,133.75	4,563.15	5,095.96

C=7	C=8	C=9	C=10	C=11	C=12
5,918.57	6,144.63	7,365.67	8,572.20	8,271.45	9,041.25
5,861.14	6,981.38	7,147.22	8,728.20	9,197.82	9,646.25
5,580.86	5,918.00	7,555.78	7,902.50	9,164.45	9,864.42
5,918.57	6,952.00	7,074.11	7,909.40	8,911.91	9,032.08
5,819.79	6,499.00	7,285.70	8,278.08	8,886.41	9,396.00

Table 5-5 Result obtained using a fourth structure of Decentralized Architecture

In this **Table 5-5** we can follow comparing that the results also were different that the obtained in the centralized architecture and that the results obtained using the first, second and third structure. Also we can see that this structure is similar to the third structure, but the times of response were totally different. For a better understanding to continue we show in the **Figure 5-11** the graphic corresponds to **Table 5-5**.

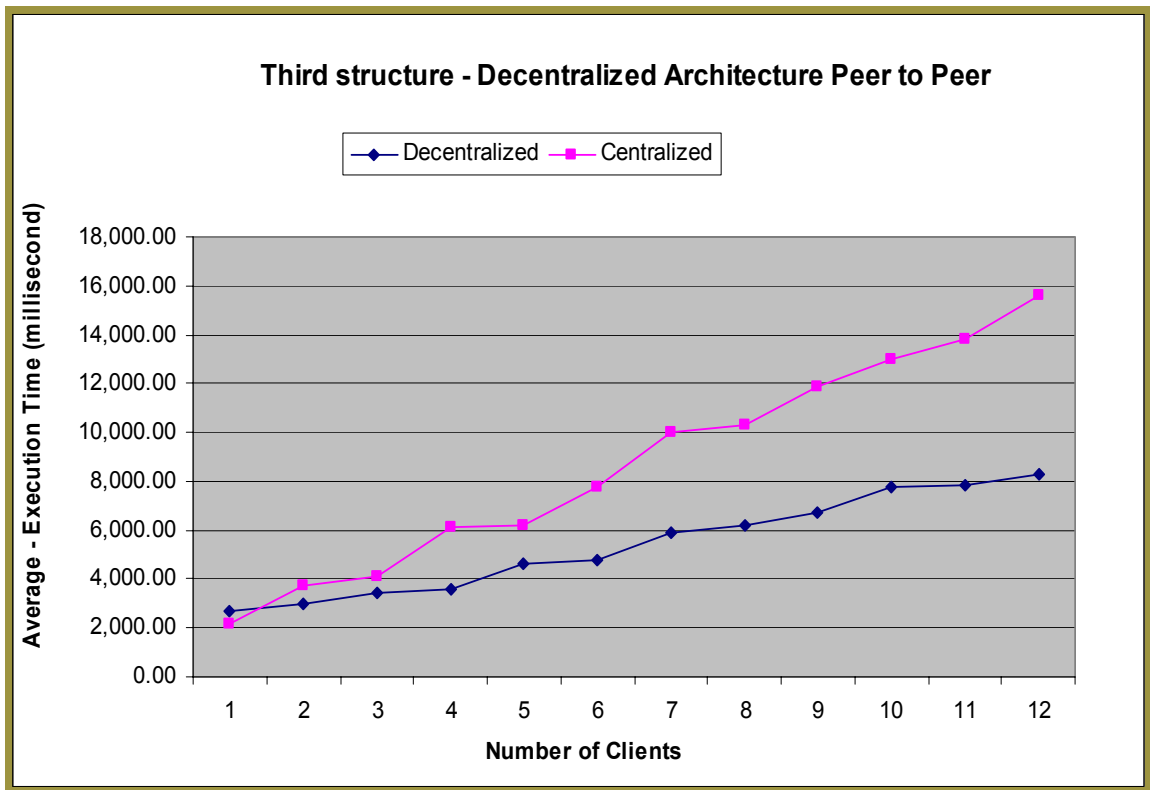


Figure 5-11 Result obtained with the fourth structure of a Decentralized Architecture

We can see that the Execution Time obtained in this fourth structure of a Decentralized Peer-to-Peer Architecture was faster than the one obtained in a Centralized Architecture. But, when comparing with the time of the responses was slowest than the first, second and third decentralized architectures shown before.

In summary, our experiments show a trend that indicates the decentralized architecture of SRE is more efficient than the centralized schemes in the previous systems.

Chapter Six

Conclusions

Our research was designed to support spatial indexing, hypertext based image visualization, and distributed data retrieval and query processing.

In this research our major contribution is the development of the Search and Retrieval Engine (SRE), a Peer-to-Peer Middleware System for data and metadata manipulation for satellite images. The benefits of SRE are: 1) can be run as part of the Web Server infrastructure available at a research site; 2) reduces infrastructure costs; 3) reduces the cost of application development; 4) reduces the effort in deployment and maintenance since it leverages on the widespread use of the Web Technology and the wealth of experience already acquired by many enterprises. The administrators simply need to *install Search and Retrieval Engine (SRE)* into their Web servers and update the two documents of the configurations.

SRE is implemented with the Java Servlets Technology and it is based on a scalable and decentralized Peer-to-Peer (P2P) Architecture where individual sites dynamically select specific data and computational resources to share, enabling a more user-oriented operation and removing the burden of centralized management. SRE will enable cooperative sites to exchange system metadata and incrementally learn about the services available on a given federation of satellite image databases deployed over the Internet.

We conducted various experiments and tests using both a centralized and a P2P decentralized Architecture, to demonstrate that the use a Decentralized Peer-to-Peer Architecture is a better option, it is faster and more efficient than using a Centralized Architecture. To demonstrate this, we used several different structures, clients and peer configurations of SRE and a centralized schema, and we compared each one these.

In our first experiment we used a centralized architecture, where the average time to execute a request was of 16 seconds proximally. In our second experiment we used a decentralized peer-to-peer architecture, where the average time of execute the same was near 7 seconds approximately. Then we had a comparison between both centralized and decentralized architectures where we can see that our decentralized architecture is more efficient and faster than a centralized architecture by a factor of 44%.

Also, we conducted several others experiments using different kinds of peer configurations. In these experiments, the results yield a better performance when using a decentralized peer-to-peer architecture.

The results obtained vary depending of the type of configuration used to arrange the peers. Depending of the type of configuration, the average time of execution for a given query, compared with the results for the same query in a centralized architecture, will be reduced between by a factor of 31% to 56% approximately.

In summary, our experiments show a trend that indicates the decentralized architecture of SRE is more efficient than the centralized schemes in the previous systems.

Future Work

We have some future work to realize in order to improve the quality of the system:

- i) We must develop a cost model that allows a site to choose its peers in a dynamic fashion. This cost model must use factors such computing power, network speed, and quality of available data to enable a peer X to decide whether or not to have site Y as its peer.

- ii) Allow synchronization and recycling of the Id_Query. With synchronization we could assure that in the case that two requests reach the same peer at the very same time, the peer will be able to process only one of them at the time. If recycling could be applied, we could reuse id numbers after pre-defined times have passed.
- iii) Implement fault tolerance into our system. This will allow an aborted query to be restarted automatically without user intervention. This feature will be required in environment where queries than take hours to complete.
- iv) To execute further experiments where can use more server, more clients and a Wide-Area environment to give stronger evidence about the benefits of SRE over centralized approaches.

Bibliography

- [1] M. Rodríguez and N. Roussopoulos. “MOCHA: A Self - Expandable Database Middleware System for Distributed Source Dates,” in Technical Report UMIACS-TR 2000-05, CS-TR4105, University of Maryland, January 2000.
- [2] M. Stonebraker and J. Dozier. “THE SEQUOIA 2000: Storage Benchmark,” in proceedings ACM SIGMOD Conference, Washington, D.C. 1993
- [3] M. Stonebraker, P. Aoki, R. Devine, W. Litwin and M. Olson. “Mariposa: A New Architecture for Distributed Data,” University of California Berkeley, California.
- [4] M. Stonebraker, P. Aoki, A. Pfeffer, A. Sah, J. Sidell, C. Staelin and A. Yu. “Mariposa: A Wide-Area Distributed Database System,” in VLDB Journal, 1996
- [5] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah and C. Staelin. “An Economic Paradigm for Query Processing and Data Migration in Mariposa,” University of California Berkeley, California.
- [6] J. Sidell, P. Aoki, A. Sah, C. Staelin, C. Stonebraker and A. Yu. “A Data Replication in Mariposa,” in proceeding 12th ICDE Conference, New Orleans, Louisiana, 1996.
- [7] R. Yang, F. Kafatos and X. Wang. “Managing Scientific Metadata Using XML,” July – August 2002.
- [8] M. Echevarria-Martines. “La Infraestructura de Datos Espaciales. Experiencias en su Implementación,” September – October 2001.

- [9] A. Coral-Liria. "Tesis Doctoral: Algoritmos para el Rendimiento de Consultas Espaciales utilizando R-Tree. La Consulta de los Pares más Cercanos y su Aplicación en Base de Datos Espaciales," Almería January – 2002.
- [10] D. Dewitt, N. Kabra, J. Luo, J. Patel and J. Yu. "Client Server PARADISE," University Wisconsin, Madison.
- [11] Microsoft's TerraServer
[Online] Available at: <http://terraserver-usa.com/>
- [12] R. Ahmed and E. Al. "The Pegasus Heterogeneous Multi Database System," in IEEE Computer, 19-27, December 1991.
- [13] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman and J. Widom. "The TSIMMIS Project: Integration of Heterogeneous Information Sources," in proceeding of IPSJ Conference, Tokyo, Japan, 1994.
- [14] M. Franklin, B. Jonsson and D. Kossmann. "Performance Tradeoffs for Client-Server Query Processing," in proceeding ACM SIGMOD Conference, pp. 149-160, Montreal, Quebec, Canada, 1996.
- [15] C. Mohan, B. Lindsay and R. Obermarck. "Transaction Management in the R* Distributed Database Management System," TODS. 11(4), 378-396.