A FRAMEWORK FOR A WEB BASED TRANSACTION COORDINATOR SWITCH

By

Juan A. Correa Colón

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

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

University of Puerto Rico
Mayagüez Campus
2006

Approved by:

Bienvenido Vélez Rivera, Ph.D. Member, Graduate Committee	Date
Pedro Rivera Vega, Ph.D. Member, Graduate Committee	Date
Manuel Rodríguez Martínez, Ph.D. President, Graduate Committee	Date
Cristina Pomales García, Ph.D. Representative of Graduate Studies	Date
Isidoro Couvertier, Ph.D. Chairperson of the Department	Date

ABSTRACT

A FRAMEWORK FOR A WEB BASED TRANSACTION COORDINATOR SWITCH

By

Juan A. Correa Colón

JSwitch was designed and developed at this thesis research. It is a Web-based transactional coordination systems designed to accept batches of transactions and route them to the appropriate transactional server application that must handle each individual transaction. JSwitch can be used as a framework to implement intra-agency and inter-agency solutions that allows transactions to be exchanged seamlessly. Moreover, JSwitch can be used by a single provider of services as a tool to balance the load among various servers used to manage transactions. These servers might be located at a single site, or distributed geographically, but accessible by means of a corporate intranet. We present an initial implementation of the system, and a performance study that discusses the tradeoff between the different load balancing policies used in the system to distribute the processing of transactional batches. These policies are a) Round Robing Scheduling, b) Random Scheduling, c) Least Loaded Scheduling, and d) Random Towards Least Loaded Scheduling. Our performance study shows that this latter provides the best performance for JSwitch.

RESUMEN

UN ESQUEMA PARA UN SWITCH QUE COORDINA TRANSACCIONES BASADO EN EL INTERNET

Por

Juan A. Correa Colón

Como parte de la investagación de la tesis JSwitch fue diseñado y desarrollado. Es un sistema basado en el Web para la coordinación de transacciones que acepta lotes de estas y los dirreciona hacia el servidor de transaciones appropiado el cual debera manejar cada transacion individualmente. El JSwitch puede ser utilizado como un esquema para implementar soluciones intra-agencia y inter-agencia que permiten el intercambio de transacciones de una manera sutíl. Aun mas, JSwitch puede ser usado por un solo proveedor de servicios como una heramienta para balancear la carga entre varios servidores utilizados para manejar transacciones. Dichos servidores podrian estar en una sola localización o distribuidos geograficamente, pero accesibles a traves de una red corporativa. Presentamos una implementación inicial del sistema y un estudio de ejecutoria en el cual se discuten las ventajas y desventajas entre las diferentes políticas de balanceo de carga utilizadas en el sistema para distribuir el procesamiento de los lotes de transacciones. Las políticas son a) distribución todos contra todos, b) distribución aleatoria, c) distribución menos cargado, d) distribución aleatoria hacia el menos cargado. Nuestro estudio de ejecutoria muestra que esta última provee la mejor ejecutoria del sistema.

To my great family

ACKNOWLEDGMENTS

First and above all God. My family, my girlfriend, my thesis advisor Dr. Manuel Rodríguez and Dr. Bienvenido Vélez and Dr. Pedro Rivera for their support and reviews of my work. I thank my friends Elliot and Rene for their constant key tehcnical and emotional support along with all the recreational time all the way from the start of the master. Finally, I would also like to thank all other people not mentioned here who have also helped me throughout the years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ADB - Application Database
BTCP - Batch Transaction Coordination Protocol
DB - Database
DBS - Database Server
DBMS - Database Management System
EDI - Electronic Data Interchange
ER - Entity-Relationship
GB - Giga-bytes
HTTP - Hyper-Text Transfer Protocol
LAN - Local Area Network
MB - Mega-bytes
Mb/s - Mega-bits per second
min - minutes
ms - milliseconds
RAM - Random Access Memory
RDBMS - Relational Database Management System
SOAP - Simple Object Access Protocol
SQL - Structured Query Language
SSL - Secure Sockets Layer
UDDI - Universal Description, Discovery, and Integration
W3C - World Wide Web Consortium
WS - Web Service(s)
WSDL - Web Services Description Language
XML - eXtended Markup Language

CHAPTER 1

Introduction

1.1 Overview

Nowadays, the advent of information technology's new generations is of great importance in all areas, especially those of eCommerce and eBusiness. The Web is well known to evolve constantly. It keeps improving in areas such as interoperability of technologies and availability of commonly used functions. This trend can be seen by the emergence of various revolutions in many areas, such as databases, as stated by Jim Gray's article: A Call To Arms. Among the database revolutions stated on the article are those entitled: Object Relational Arrives; Databases and Web Serives; Queues, Transactions, Workflows; Smart Objects: Databases Everywhere; and Self Managing and Always Up.

This constant technical evolution normally translates into the birth of mature and robust solutions. However, this innovation also brings new unforeseen and often undesirable problems. Examples of some of these problems include system interoperability, scalability and backwards compatibility. These must be addressed as soon as they emerge or the developer risks deploying an unusable software system.

In this thesis we consider the problem of building transaction coordination systems on top of Web technologies. These systems are designed to collect various types of transactions and forward them to the the proper transactional engine (e.g. a relational database system). For that reason, these type of systems are often called *Transactional Switches*. Typically, the transactions are submitted in batches and the user later wishes to follow up on the status of each individual

transaction. eCommerce and eBusiness applications rely on transaction coordination solutions to manage large amounts of transactions that arrive every day. Often, the transaction coordination solutions are built in a very ad-hoc manner, using proprietary components. As part of this thesis, we present a Web-based framework that can be used to built transaction coordination solutions. Our approach builds heavily on open standards (e.g. Web services) to provide a scalable, easy to use and efficient solution. We call our approach the **JSwitch Web Based Transaction Coordination Switch Framework**, and it is designed to improve the current state of technology in the online-transaction processing area.

1.2 Problem Statement

Currently, the emergence of eCommerce and eBusiness applications presents some new challenges that must be met. Among these challenges are the increase in the variety of audiences reached and transactions performed, the great number of eCommerce and eBusiness technologies involved, and the integration of emerging ones. Some of the target audiences for these applications are the government, military, health, education, industrial and the financial sector. The solutions implemented range from banking, manufacturing, and health billing to retail and media transactions, among many others.

This diversity produces a level of heterogeneity among the systems that demands special attention. This problem, along with the scalability challenges brought on by the large utilization of such systems, presents one of the biggest challenges to the eCommerce and eBusiness technologies in order for them to continue to prosper. Online-transaction processing systems must take into account the emergence of promising new information technologies, along with the challenges mentioned above, in order to offer robust, scalable, and adaptable solutions.

In this thesis, we study the problem of building a transaction coordination systems using Web-technologies, particularly Web Services. Our hypothesis is that such system can be built by customizing a set of components with clear responsibilities and behaviors. The key to our approach is to deploy various components responsible for transaction coordination as Web Services. These services are run on servers located on the corporate LAN to ensure reliability and availability. We study various alternative policies to schedule the assignment of transaction batches between the

servers in the LAN. Such batches will find their way to the proper and most efficient transactional engine available in the enterprise. Our experiments show that a hybrid scheduling approach, which combines randomization with informed site selections provides the best tradeoff for scheduling transactional batches.

1.3 Contributions

The aim of this research is to define a framework that establishes the foundations of unification among eCommerce and eBusiness technologies in terms of transaction batch coordination for the Web environment. In addition to this outcome, the system should provide a robust solution in terms of work distribution and load balancing, along with crash recovery at a basic level.

While RDBMS and TP Monitor systems have been around for some time now, much of their utilization has been in the private sector. These systems have been widely deployed and used with success in centralized, population fixed environments at companies and organizations providing reliable business essential services. In contrast to current transaction processing systems, the JSwitch framework offers a solution specially designed to cope with the scalability presented by a highly dynamic Web environment.

Furthermore, JSwitch offers the integration of promising cutting edge technology into the framework, therefore bringing innovation to the information technology field. The success of this integration is possible thanks to the efforts that the Web community has agreed upon by producing standards for Web applications. This is a practice that one should strive for whereever possible, as this framework does. The JSwitch framework provides an example of how to use current standards and a motivation toward the production of more standards of efficient and effective communication between the different systems.

The JSwitch framework defines a Batch Transaction Coordination Protocol in order to enable the processing of batch transactions by the system. The framework also defines a communications protocol to be followed by the clients of the JSwitch. These protocol definitions strive to enforce the practice of standard creation for Web applications, resulting in guidelines for others to follow and improve.

Through the implementation, testing, and analysis of a JSwitch prototype we are able to

obtain measurements to shed light in the system performance, and its capabilities for handling load and crash recovery. These measurements will be part of the system profile and will also contribute in the evaluation of emerging solutions based on the technologies and standards used here or even new generations of these.

This research provides results about the use of cutting edge technology applied to the transactions involved in systems deployed over Web environments. Therefore, the experimental results of this research could help in the creation of mature and robust eCommerce and eBusiness technologies, that will offer many more benefits and will reach a larger population.

1.4 Thesis Structure

This chapter has addressed the introduction of the thesis research; the rest of the document is organized as follows. Chapter 2 contains a general overview of available work and articles related to this research and the necessary theoretical background to achieve a better understanding of this work. Chapter 4 introduces the JSwitch framework and describes its functionality as a whole and that of each of its parts. Chapter 5 presents the different experiments that were carried out in order to test the proposed system and analyzes their results. Chapter 6 summarizes the findings of this research and the achievements of the JSwitch framework. It also gives various ideas and possibilities for new research projects and for expanding or improving the current system. Finally, the Appendices give detailed information on the internal workings of the JSwitch framework, the implementation, and the data captured from the experiments.

CHAPTER 2

Literature Review

This chapter provides a literature review of the theoretical background along with the related works in the area of transaction coordination and areas related with the conducted research.

2.1 Theoretical Background

This section provides theoretical background about the topics necessary to understand the research conducted and presented at this document.

2.1.1 eCommerce, eBusiness, eGoverment Technology

The information era in which we live today has been possible due to the success that technology has in constantly making information more accesible, with richer variety and larger user base. This has been mostly done via computer networks and Internet applications, particularly the Web. It is important to mention the fact that the Internet is only posible thanks to the research and success of computer networks. Today, the terms of eCommerce, eBusiness and eGoverment stand out in this era because of their great acceptance and many benefits to organizations and general public.

The term eCommerce is defined as the process of buying, selling, transferring, or exchanging products, services, or information via the Internet. eBusiness is known as the term to characterize a process in which a company performs all or most of its business functions electronically. For instance, the automated supply chain process where products go from the manufacturer to the wholesaler, as Walmart does with many of its partners suppliers (e.g.,, Procter and Gamble) to keep inventory. eGoverment was born from the initiatives that some Goverment agencies have

taken of adopting succesful eBusiness practices to help their intra-business operations and inter-organizational business, along with online goverment information and services for the citizens. Some eGoverment initiatives already in place in the US, acording to a 2000 surveys by researchers at Brown University's Tauban Center for Public Policy [15], include the ability to file taxes online, being able to order publications online, filing complaints, submitting vehicle registrations and ordering hunting licenses. Note that the eBusiness term is recognized as the broadest definition of eCommerce, which includes intrabusiness, interorganizational business, and eCommerce [43] along with eGoverment. Therefore from this point on eBusiness will be use to refer to these three terms unless otherwise specified. Also notice that sometimes eCommerce, eBusiness, and eGoverment Technology are used as generic terms for the tools that made possible electronic transactions. These tools range from computer hardware, including servers, to networks and the software that provides the business logic for the application at hand.

2.1.2 Relational Databases

A relational database management system (RDBMS) is normally used to record the transactions performed by eBusiness systems. The RDBMS in these applications represents the persistence layer. RDBMs are responsible for the maintenance of the data in large structured sets [37]. The relational model was introduced by Edgar F. Codd in 1970 [10], and today is the most widely used model in database applications. This model is based in the fundamental assumption that all data are represented as mathematical relations. A database is a collection of one or more relations, where each relation is a table with rows and columns.

The relational view of data as explained by the author in [10] is as follows. The term relation is used here in its accepted mathematical sense. Given sets S1, S1, ..., Sn, (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S1, its second element from S1, and so on. We shall refer to Sj as the jth domain of R. As defined above, R is said to have degree n. Relations of degree 1 are often called unary, degree 2 binary, degree 3 ternary, and degree n n-ary. For the purpose of explanation, Codd used an array for the representation of relations. An array which represents an n-ary relation R has the following properties:

1. Each row represents an n-tuple of R.
2. The ordering of rows is immaterial.
3. All rows are distinct.
4. The ordering of columns is significant - it corresponds to the ordering S1, S1, , Sn of the domains on which R is defined (see, however, remarks below on domain-ordered and domain-unordered relations).
5. The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

RDBMs support transactions in a way that guarantees all of the ACID properties for each transaction. The ACID properties of atomicity, consistency, isolation and durability (explained in a following section) are one of the key features of a RDBMS, responsible for the integrity of the database.

Middleware is defined as the connectivity software layer between the operating system (OS) and the applications on each site of a distributed computing system. Middleware is the enabling technology for Enterprise applicaiton integration. The access mechanisms to RDBMSs used by middleware appllications are JDBC[25], ODBC[32], and other vendor-specific APIs.The Middleware technology systems are often classified as:

- Remote Procedure Call (RPCs) - a client makes calls to procedures running on remote systems and receives a response over the networks.
- Publish/Subscribe - monitors activity at the data sources and pushes (i.e. sends) relevant information to subscribers interested in those data sources.
- Message Oriented Middleware (MOM) - messages sent to the client are collected and stored until they are acted upon, meanwhile the client continues with other processing tasks.
- Object Request Broker (ORB) - makes it possible for applications to send objects and request services to remote severs in a distributed system.
- SQL-oriented Data Access - middleware between applications and relational database servers.

The entity-relationship (ER) model is a high-level description of the actual data in real world applications. Basically, the ER-Diagram describes the data as entities and the relationships among these entities. In practice the principal use of the ER model is in the initial phase of development of database designs. Once an ER-Daigram is conceived, it is directly mapped to a relational schema. The conventions used for this diagram is as follows: boxes represent entities, ovals represent their attributes, double ovals are composite attributes, and diamonds represent the relations among entities. All entities and some of the relations are mapped direcly or almost directly into tables in the database. The RDBMS book of Ramakrishnan and Gehrke [37], identifies the first three steps of the database design process as:

1. Requirements Analysis - Understand what data is to be stored in the database, what applications will be built on top of it, and what operations are more frequent and subject to performance requirements.

2. Conceptual Database Design - Developmet of a high-level description of the data to be stored in the database, along with the constraints required to be hold over this data. This step involves the creation of the ER Model as a semantic data model that should lead directly to the data model supported by a commercial database. This last model is known as the relational model.

3. Logical Database Design - Refers to the selection of a RDBMS system to implement the database design into a database schema in the data model of the selected RDBMS. The result of this action is called the logical schema.

A database cluster is a collection of databases that will be accessible through a single instance of a running database server[36]. It has a predefined database storage area on disk which is termed the database cluster, but on the other hand SQL refers to this as a catalog cluster instead. In file system terms, a database cluster is a single directory under which all data will be stored. This is called the data directory or data area. There are several variations and puropses for DB clusters but the main two are a failover cluster and a load-balancing cluster [7]. The first is a set of two or more independent computers that share resources, in this way if one of the servers fails, another server in the cluster will take over the resources and the processing load. The system

behaves differently in a load-balancing cluster, since the requests for processing are distributed among the servers. The different servers in a load-balancing cluster share processing load but do not share resources such as a disk array or memory. If one of the servers fails, the processing load can simply be redistributed among the surviving nodes in the cluster.

The fail-over cluster shares resources between computers. The actual implementation of this sharing can be divided in two solutions: share nothing at any point in time; and share the resources at some point in time. Because the shared resource is most often a disk array, most clustering solutions are based on either the shared-nothing model or the shared-disk model.

In the shared-nothing model, each server in a cluster controls a separate set of resources, such as a different disk partition in a shared disk array. Only one server can own and access a particular resource or partition at a time. In the event of a failure, another surviving server in the cluster takes over the resources of the failed server and subsequent client requests are routed to this server. In the other option the shared-disk model, multiple servers in a cluster can simultaneously access a shared disk. The synchronization logic required to maintain data integrity makes this model more complex.

2.1.3 Web Services

A Web Service is defined as a software system designed to support interoperable machine to machine interaction over a Web-based network, which can be either the Internet or an internal intranet. This interoperability is reached through a series of protocols and standards . Basically, a web service is a software that makes RPC between machines using the HTTP protocol to encode the communication exchange.

The standards and protocols that enable Web services are XML, SOAP, WSDL and UDDI, a definition of these is next. The eXtensible Markup Langueage (XML) is a general purpose markup language that is of great use since it is capable of describing many different kinds of data. The primary purpose of XML is to facilitate the sharing of data across different systems connected over the Internet. The Simple Object Access Protocol (SOAP) [45] is responsible for the exchange of XML-based messages over computer networks, therefore achieving cross-platform inter-application communication. Another standard used is the Web Services Description Language (WSDL) [8], this

is an XML-based service description on how to communicate using the web service. In other words, WSDL describes the entry points or specific operations allowed for each available service. Finally, the Universal Description, Discovery, and Integration (UDDI) [44] protocol is used for publishing the information about web services, and help discover these on the Internet.

2.1.4 Transaction Processing

A transaction in its more pure form is a transformation of database state, for example a set of queries which access and possibly update the database. Such transaction has the ACID properties of: atomicity (all or nothing), consistency (a correct transformation), isolation (execution is serializable) and durability (effects survive failures) [23]. In the business context a transaction also follows the ACID properties detailed next and is defined as an interaction in the real world, usually between an enterprise and a person, where something is exchanged. This business transaction could involve exchanging money, products, information, services among many others. Usually bookkeeping is required to record what happened. Often this bookkeeping is done by a computer, for better scalability, reliability, and cost[3].

The ACID properties and their respective definitions are:

- **Atomicity** - ability of the DBMS to guarantee that either all of the tasks of a transactions are performed or none of them are executed if a failure occurs.
- **Consistency** - refers to the database being in a legal state when the transaction begins and ends. A transaction cannot break the rules and integrity constraints of the database.
- **Isolation** - ability of the application to make operations in a transaction appear isolated from all other operations. Thus, each transaction is unaware of other concurrently executing transactions.
- **Durability** - guarantee that once the user has been notified of success, the transaction will persist, and will not be undone, no matter if a system failure occurs.

Compensating transactions are closely related to the transaction processing concept. This mechanism exists because of the fact that people make mistakes and if a transaction is commited (which makes the transaction durable) it can no longer be aborted. When compensating transaction, another transaction is run to reverse the effect of the one that was committed. Sometimes

perfect compensation is simply not posible since the transaction performed some irreversible act, like selling the last appartment in a condominium to the wrong people. In this case, the compensating transaction may be to record the error in a database and send an apology e-mail to the original clients.

2.1.5 On-line Transaction Processing

Data interchange mostly in the form of on-line transactios are the core mechanisms in every eBusiness process. On-Line transactions are defined in [3] as the execution of a program that performs a business function by accessing a shared database, usually on behalf of an on-line user. It is important to notice that a transaction always refers to the execution of a program that contains the steps involved in a business operation.

The collection of transaction programs designed to automate a given business activity is known as a *transaction processing application*. These are implemented in specially engineered hardware and software environments. Transaction processing (TP) is also a style of system, this is, there are different ways to configure software components to do the type of work required by a TP application. There are several styles of systems for a transaction processing application but in the context of this research, the most relevant is the Batch Processing System, which is presented next.

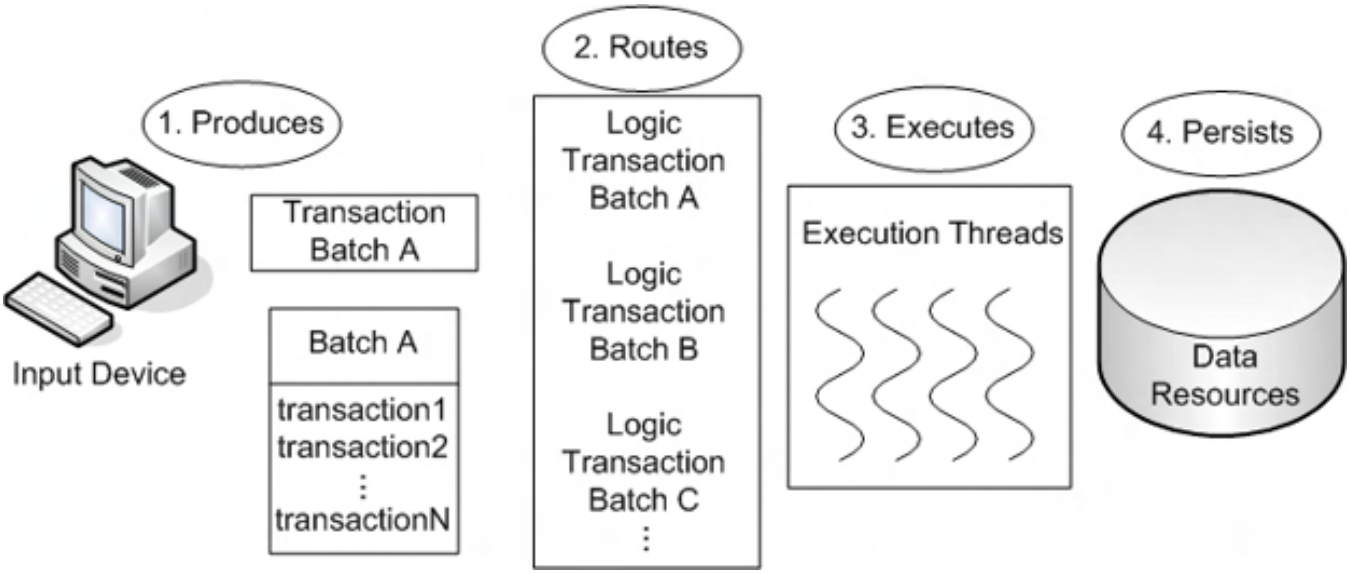

Figure 2.1. Batch Processing System Interactions

A Batch Processing System takes of processing groups of transactions that arrive as a group, known as a batch. This batch is often implemented as file. This batch is also defined as a set of requests that are processed together, in general they are processed some time after the orginal request was submitted. Figure 2.1 displays the interactions in such batch processing system. Initially a batch is produced at some computer containing a set of transactions. After the batch is submitted to the systems the batch is routed among the trasaction processing logics in order to determine how it is going to be processed. Finally the batch is processed as determined by the transaction processing logic and the reults are persisted to a data storage commonly a RDBMS.

Early data processing systems, those of 1960s and 1970s, were designed to be used primarily for batch processing. Current systems, however, allow both batch processing and on-line processing to occur concurrently. Often, however, the batches are executed at non-peak periods since batch processing generally does not have time constraints. Batch Processing systems are uniprogrammed in the sense that each batch is a sequence of transactions that is executed sequentially one transaction at a time. Finally batch processing system performace is measured in terms of throughput which is defined as the amount of work done per unit of time.

Transactions that update databases at multiple systems in a distributed environment require what is known as the two-phase commit protocol[3]. This protocol ensures that the transaction commits at all sites or aborts at all of them. The Transaction Processing Performance Council (TPC)has several benchmarks used as a standard suite to measure the performance of a TP system [42]. These benchmarks compare TP systems based on their maximun transaction rate and price per transaction for a standardized order-entry application workload.

Another critical aspect in TP systems is the system availability, is the fraction of time the system is running and able to do work. Availabililty is determined by how frequently a TP system fails and how quickly it can recover from failures. In summary, TP systems must ensure ACID properties, throughtput, and availability as their key features.

2.1.6 On-line Transaction Coordination

Let us begin with the definition of coordination in its pure form which is the additional information processing performed when multiple, connected actors pursue goals that a single actor

pursuing the same goals would not perform [28]. The elements of coordination are: (1) a set of (two or more) actors, (2) who perform tasks, (3) in order to achieve the goals. For example, a football team has a set of players and coaches (actors) that execute plays (tasks) designed to win the game (goal). The interation between players and coaches is the coordination. Players and coaches by themselves do not win games, but their cooperative efforts and interaction does.

The fact that these components are present does not mean that all the activites in a situation are coordination. The goal relevant tasks are divided in two categories: coordination tasks and production tasks. Coordination tasks are the information processing tasks that are performed because more than one actor is involved. Production tasks are all the other tasks that are performed in order to achieve the goals. For instance in the previous example the actual actions of the players (e.g., passing the ball) can be seen as the production tasks, while all the practices and talks previous and during the game are the coordination tasks.

On-line transaction coordination refers to the activities needed to engage in transaction processing amongst multiple parties. When applying the previous definition of coordination to on-line business transactions we get the following. An entity performs some busniness activity over the Internet. The set of transaction processing applications that handle each task in the transaction are the actors. And the resulting changes to the databases, movement of goods and money become the goals. The coordination is the set of operations needed to make these applications (actors) interact with each other to complete the tasks at hand (the goals). Thus, the coordination tasks are on-line transaction coordination activities between involved parties, and the production tasks are the on-line transaction processing activities at each site. For instance, a student might want to buy a book from an on-line store (e.g. Amazon). But if such book in not in the Amazon inventory, this on-line store has the ability to put a purchase order for the book at another on-line partner book store, lets say Barnes and Noble. In this scenario Web services across on-line stores communicate each party to complete the on-line transaction, and this exchange of data between sites would be the on-line transaction coordination.

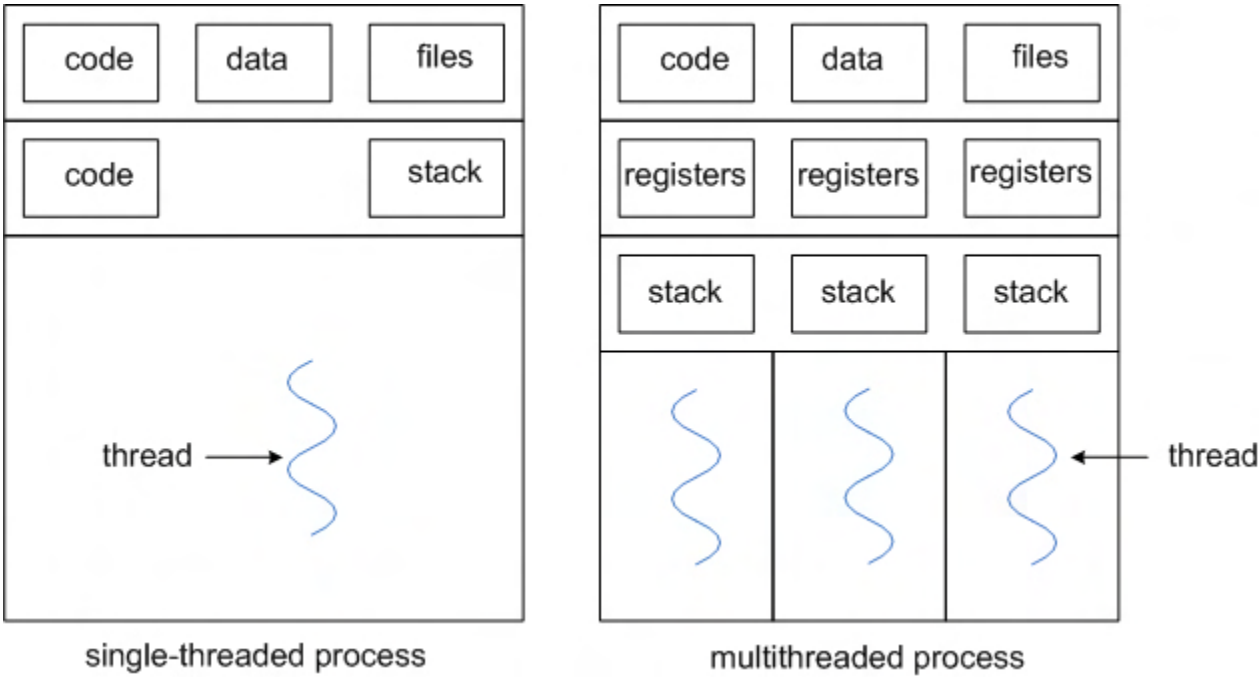

Figure 2.2. Singlethreaded and multi-threaded process

2.1.7 Implementation Techniques

An implementation technique used for improved performance and utilization of system resources is the technique of Multithreaded Programming. In order to define Mutilthreaded Programming lets first introduce the concept of a thread. A thread is a basic unit of CPU utilization; it poseses a thread ID, a program counter, a register set, and a stack [40] [30]. A thread shares with other threads belonging to the same process its code section, data section, and other operating system resources as displayed in Figure 2.2.

The concept of a thread is a software technique used to capitalize on the advances in performance and availability in terms of cost of CPU, Disk and other hardware. Multithreaded programming allows multiple threads of execution on a single processor or multi-processor system. Each thread is allow some time of execution on the CPU. Thus, if some thread become blocked waiting for I/O then another thread might come an use the CPU. The goal is to keep the CPU or CPUs as busy as possible. For example, instead of a single process accesing a database, multiple threads can have access to the data resources simultaneously. The concept of a Thread Pool (see Figure 2.3)is explained here since it is an efficient programing practice use throghout the framework. Thread creation is a very expensive operation,and therefore, it is desirable to pre-allocate a set of

threads and re-use them. This can be done by using a queue to store idle threads, or remove threads when a task must be done, as shown in Figure 2.3. If the thread pool is not used, then a new thread must be created every time one is needed. This often causes decreased performance due to the overhead incurred in the creation and activation of threads. Mutilthreaded programming will be used in the JSwitch framework in order to capitalize on its streghts, which will result in achieving better overall system efficiency and effectivenes. Finaly, multithreaded programming is a key ingredient to achieve application scalability. This is achieved by breaking the clasical sequential process execution by executing multiple threads to attend multiple processes at the same time instead of waiting to execute process sequentially.

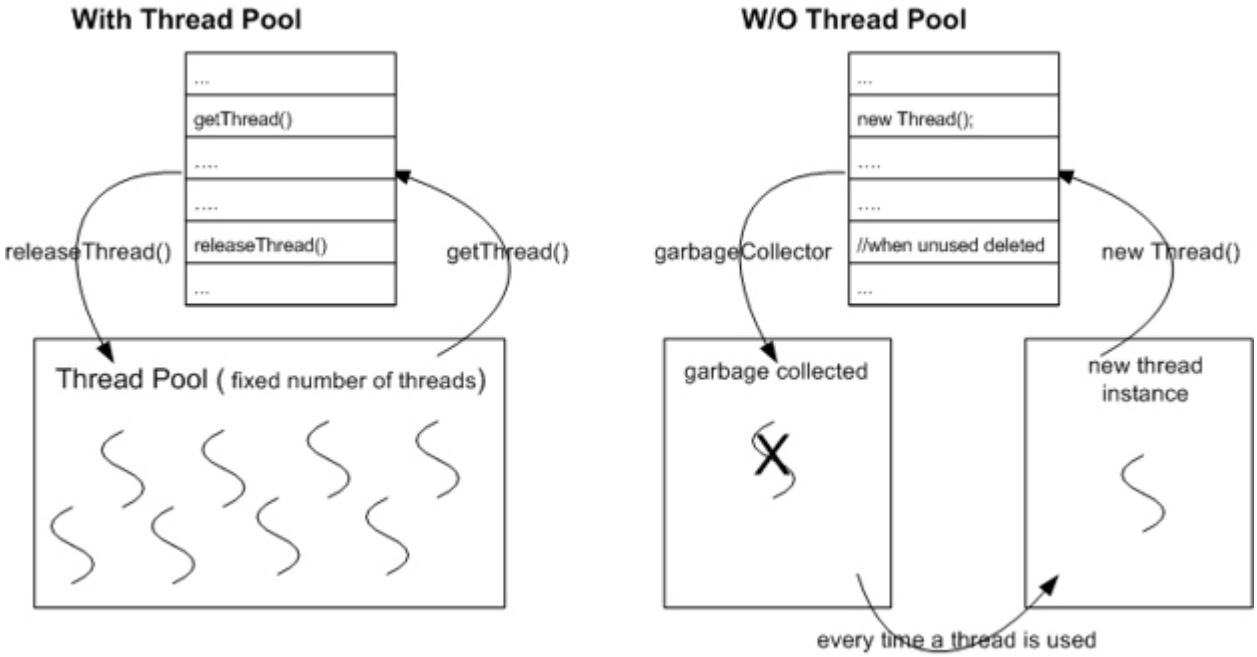

Figure 2.3. Thread pool vs dynamic thread allocation.

The other programming technique that is an essential part for the implementation of the JSwitch framework presented here, is the Producer Consummer Queue. This technique provides the basic properties found in a queue defined as a buffer where various entities such as data, objects, persons, or events are stored waiting to be processed. A queue (a line) with a FIFO (first-in-first-out) behavior is a collection where the first element added to the queue is the first one out, as shown in Figure 2.3. In terms of scheduling, a queue is natural data structure for a system to serve

the incoming requests. Basically, the Producer Consummer Queue (PCQueue) displayed in Figure 2.4 is the base of this programming technique. It uses a queue called the job queue where at one end objects that perform some operation are added to the queue by a single thread. At the other end of the queue, a fixed number of threads, called worker threads, remove these objects one by one and then call the methods that execute some operation implemented by the object. In this way, the task implemented by these objects is run by the system. Once the work is completed, the worker thread release the object, and goes back to the head of the queue to get a new object. The Producer is the single thread that adds new objects to the queue and the Consumer or consummers are the working threads that remove and use the objects. This technique enables the processing of different kinds of operations without the worker threads knowing the specific business logic, they just call the execute method on the object. This technique is used extensively in the JSwitch framework.

The Command Pattern Design is used as the mechanism to implement the object that implement the operations. The complete definition is as follows: the Command Design Pattern encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log reuests, and supports undoable operations [17]. A basic command interface has an execute method which provides the main mechanism of processing a request. In this manner objects implementing the Command Interface with different kind of requests, for example logic of a transaction, are added to the queue for their later execution. This execution is as simple as calling the excute method of the object.

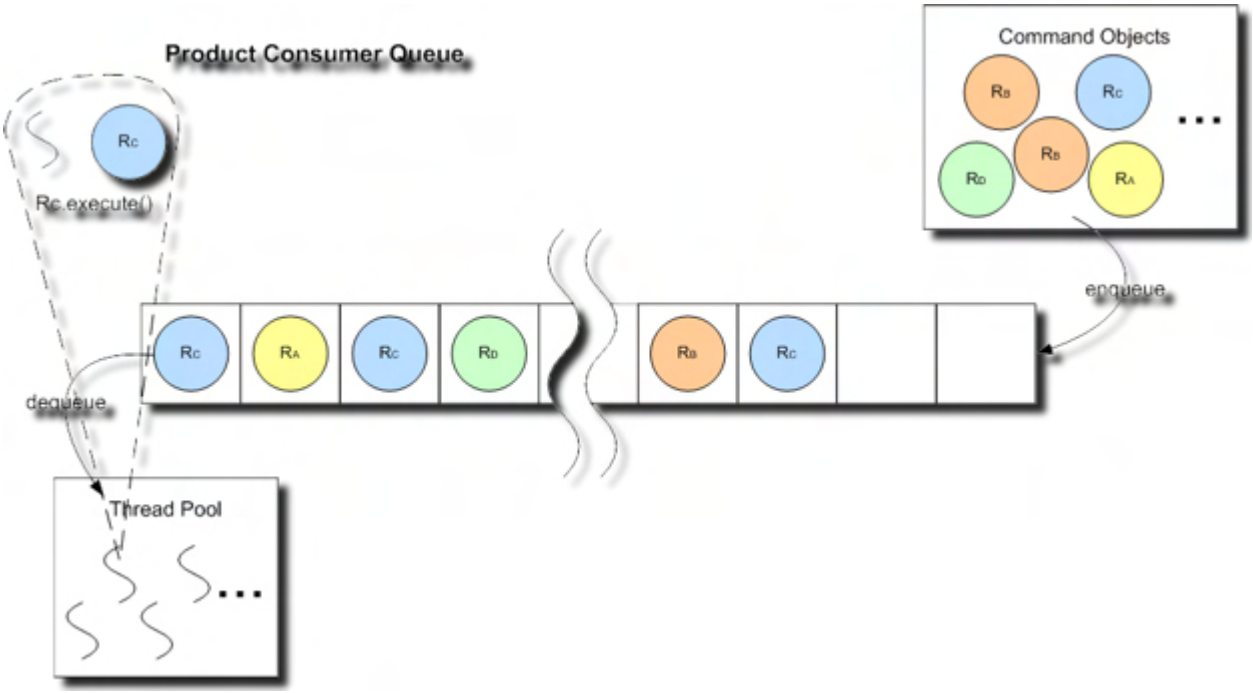

Figure 2.4. Product Consumer Queue

2.2 Related Work

This section presents some important publications and different work related to this research and its topics.

2.2.1 Relational Databases

Gray has made some revealing statements in his publication *The Next Database Revolution* [21] . These statements refer to several database revolutions and among those, the ones that most concern this research are presented next. First, the wide spread use of object-relational systems, and object-oriented API (i.e. Java JDBC) enable far richer data representation. This allows developers to mix SQL with Sun Java or Microsoft C Sharp, and generate complex stored procedures. Hence, rather complex transactions can be fed into the DBMS.

Second, is the statement that databases can be exposed as web services, which empowers DBMSs for network and cross-platform capabilities. This can provide a solution known as TPlite, which represents the return of the client-server model as a viable design option. Nontheless, this TPlite solutions suffers from security issues since it exposes the databases to attacks, and it also brings vendor specific limitations into the picture. The TPlite option offers a two-tier solution, but the three-tier solutions, like TP-Monitors, are still the main option in use. Both of these technologies are succesfull solutions at transaction processing. RDBMS and TP-Monitors could also coordinate transactions but unfortunately they cannot scale as required by the web. This results from the volume and diversity of transaction requests along with the appearance of sudden load spikes. Basically, these are the issues that the JSwitch framework deals with.

The last statement concerning Jim Gray's research is about queues and transactions. Queues are an integral part of every database since they have been proven a good programming technique, and some concepts adopted in this research come from the queue concept like the Product Consumer Queue. Gray mentions that actually there is a lot of experimentation going on in this field and specifically mentions that the ability to publish queues as Web services has been a productive discovery. It closes the discussion of this last statement saying that there is hope for a design pattern to emerge for this matter.

A database utility called dbSwitch [13] introduces the Database Area Network (DAN) ar-

chitecture. This product offers valuable features such as resource optimization, high availability and the opportunity for consolidation and scaling on demand. This application is targeted for data centers and provides capacity management, monitoring and reporting capabilities. It introduces a new approach for database security called zoning, which filters database access to applications at the network level, and thus goes beyond the basic user/password mechanisms used by the DBMS. The authors of [13] conclude their article with the statement that is their belief that "their product dbSwitch in addition to the DAN architecture represents important steps towards a database utility model".

2.2.2 On-line Transaction Coordination and Processing

The Web Services field is a fruitful research area where there are constant improvements and discoveries. Currently, Web services are evolving towards the foundation technology for eBusiness. This is being accomplished by providing the means for on-line trnsaction coordination and processing. For instance an example of this trend can be seen in the well-known Web-based application portals of eBay and Amazon.com. Protocols and standards are being established which are essential to the succes of Web-service as the enabling technology for eBusiness. Several of these protocols and standards are presented later in this section. Furthermore, there is research being conducted currently in the area of Web services load balancing [35]. This is a business-critical aspect that must be addresed by eBusiness since without it, even the availability of dozens or hundreds of middleware components or the addition of new ones to scale in response to new resource demands will not suffice to attend the web flash traffic patterns that lead to system overload. This overload is directly related to the poor performance that such systems display during demanding seasons or events, for example the holiday season or at special events where discounts are available. This poor performance is well known to result in loss of potential customers and revenue. Experiments have revealed that users can tolerate roughly 8 seconds [4] of delay before they either retry their request or leave the site altogether.

The work of [26] is related the composition of web services and the management of web service-based transactions on such web services. This paper mentions the Transaction Internet Protocol (TIP) 3.0, defined in RFC2371, as a transaction protocol enabling distributed transac-

tion coordinators to communicate over the Internet. This protocol performs well on short lived transactions but is lacks the ability to process long lived transaction. The latter ones are called *business transactions* that consist of a large number of component transactions with largely different response times, which therefore block resources controlled by short lived transactions for unacceptably long periods of time. This makes TP systems unable to process new service requests.

Web services have been found very helpful in achieving the potential of eBusiness, but to follow this trend its shortcommings must be addressed. Some of these shortcommings are: a) the opinion of IT profesionals in terms of finally trusting the reliability of this new network technology [12]; b) Web Service security[12] [34], c) exception handling [34], d) Web service load balancing [35] and e) Web services-based transactions in special business transactions [12] [34]. Our work is related with Web Service load balancing since it is a technique used for the coordination of transactions in the framework.

Business transactions differ from traditional transactions in the sense that eBusiness transactions sometimes execute over long periods of time or even across different business boundaries, requiring commitments to the transaction to be negotiated at runtime and isolation levels must be relaxed [27]. Therefore as a conclusion and the statement of the article [27] traditional transactions semantics (ACID) and protocol have proven to be inapropiate for business transactions.

Mark Little, among others in [27] and [12], present the Business Transaction Protocol (BTP) developed by the Organization for the Advancement of Structured Information Standards (OASIS) to address the latter shortcoming, specially for Web-based long-running collaborative business applications. Additional to BTP, Web services coordination (WS-C) with its WS-Transaction (WS-T) protocol sponsored by the W3C organization and Intel's tentative-hold protocol (THP) present near future solutions to the problem of long running transactions. These standards are essential to unleash the full potential of eBusiness in collaborative business applications. The reason for this was revealed by a Computer Sciences Corporation survey to senior information technology executives where they ranked "connecting to customers, suppliers, or partners electronically" as the top global IT management issue [34]. The description of the approaches that these protocols take are presented next.

The Business Transaction Protocol (BTP) developed by OASIS, is designed to support inter-

actions that cross applications and adminstrative boundaries, thus requiring extended transaction support beyond classical ACID. It handles long-running transactions no matter who defines them to permit flexible outcome for the transactional activities, and to support a transaction concept that goes beyond data-centric transactions [5]. This is important since business transaction must ensure process consistency as well as the standard requirement for data consistency for Internet-based applications.

The next protocol, named Web Services Coordination (WS-C), provides abstract notion of coordination that is extended to specific coordination protocols by the addition of third party agents. WS-C differs from BTP in the sense that BTP has been designed specifically for transaction coordination and interactions, therefore the WS-C is more flexible. The only protocol proposed for WS-C is the WS-T which has two transaction models, atomic transactions and business activities. Atomic transactions follow the classical semantics of ACID which asume the locking of resources while transaction execution. The business activities model is the novel one for the long-running transactions which ensures that any update state in a system is made inmediatelly with the objective of reducing the period during which the lock must be held. A shortcoming of the business activities models presented up to this point is that they do not support two-phase commit. If a failure occurs, compensations transaction are performed to restore the data to a consistent state.

Finally, the third protocol we discuss is Intel's tentative-hold protocol (THP) which introduces the tentative-hold phase where business exchange tentative commitments to the terms of a contract or interaction in a nonblocking manner [38]. Clients in a business transaction can tentatively obtain resources for long periods of time without requiring the resource provider to lock such resource for the duration of the period. If a client subsequently commits a transaction and acquires the resource, THP will inform them that the holds they have are no longer valid. The concluding remarks of [12] is that eventually the proponents of these three approaches will reach a consensus where a single specification that harvests the better points of each will likely prevail.

Currently, transaction standards as those mentioned above are working their way through industry channels and into vendor products. The purpose of these standards is to combine Web services to create higher level, cross-organizational business processes. Approaches for creating business processes from commposite Web services are known as Web Services Orchestration and

Choreography. Web service orchestration refers to an executable business process that can interact with both internal and external Web services. The interactions occur at the message level. They include business logic and task execution order, and they can span applications and organizations to define a long-lived, transactional, multi-step process model.

Chreography tracks the message sequences among orchestration processes between multiple parties and sources. Typically the public message exchanges occur between Web services rather than specific business process that a single party executes. Web service Orchestration primarily differs from Choreography in the sense that Orchestration always represents control from one party's perspective. Choreography is more collaborative and allows each involved party to describe its part in the interaction.

Both of these middleware technologies must provide several properties to overcome some of the traditional transaction processing and Web service standards limitations already discussed; that is ACID not being flexible enough to execute long-running transactions accros organizations (business transactions). One of these properties is asynchronous service invocation which improves reliability and scalability. Another property is flexibility in terms of separating the business logic from the Web services used. This is achived with an orchestration engine. Finally, process designers must be able to compose higher level services from existing orchestrated processes.

The article [34] presents that the intiatives of Business Process Execution Language (BPEL4WS), Web Service Interface Choreography Interface (WSCI) and Business Process Management Language (BPML). BPEL4WS on their own supports Orchestration and Choreography with their Executable processes and Abstract processes respectively. WSCI describes only the observable behavior between Web Services. It does not address the definition of executable business processes as BPEL does, it only provides Choreography. BPML is an XML based language for describing business processes, therefore only provides Orchestration. The last two combine to offer a complete Orchestration and Choreography which taken together are an alternative to BPEL. The author personally selected BPEL over these alternatives because it includes tutorials that are well suited for the developers. The author also mentions that BPEL is being embraced by the industry and that in the long run vendor and tools support will clearly influence the software industry's adoption rate.

The author of [34] presents the lack of direct support for security by the Orchestrantion and

Choreography standards as an area for future work. Another to area in which research is needed is end-to-end management infrastructure for applications [34], systems and networks that gives them flexible control over business processes involving Web services, including control of specific steps in the process. A robust management infrastructure must provide funtionality to both monitor the computing environment and adapt to changes in real time.

CHAPTER 3

Web Switch Transaction Coordinator Framework

This chapter describes the *Web Switch Transaction Coordinator* (JSwitch) framework. We present its architecture, describe its major components, and the protocols for transaction coordination. In addition, the Scheduling Policies used to load balance the system are discussed.

3.1 General System Architecture

The JSwitch framework is a Web-based middle-tier solution designed for environments in which transactions arrive in batches, each one containing a collection of related transactions. For example, a batch related to the traffic ticket application implemented based on the JSwitch framework, might contain new tickets issued during the day by police officers of a given unit. The JSwitch architecture shown in Figure 3.1 follows a three-tier model: a) Preparation Tier, b) Persistance Tier, and c) Processing Tier. The *Preparation Tier* consists of components that accept each batch and decompose them into individual transactions. The elements of this tier are the Transaction Coordinator (TCoord) and Transaction Collector (TColl). Users submit batches to the system using a Web-based application. This application first connects to the TCoord, using a Web service call, to authenticate into the system and then obtain the URL of a TColl that can handle the batch. Hence, the responsibility of the TCoord is to look into a pool of available TColls, and find the best one to handle the batch to be uploaded. We developed load balancing metric for the system load and workload in order to have an informed selection criteria for this purpose. Once

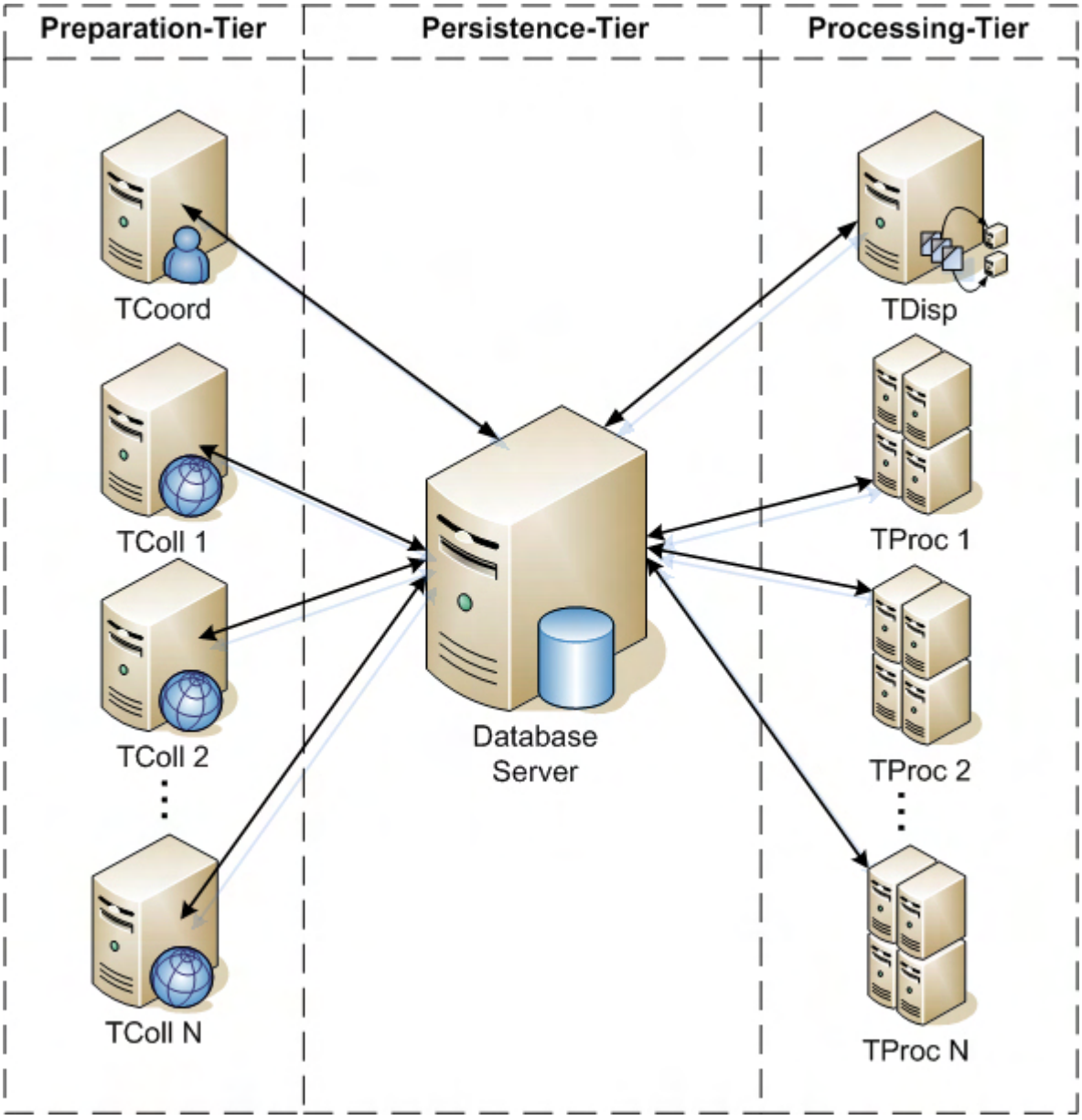

Figure 3.1. System Architecture

the client application obtains a TColl, it issues another Web service call, this time to the target TColl, to upload the file that contains the batch of transactions. The target TColl receives the batch, and begins to decompose, analyze, and validate the structure of the individual transactions.

Each individual transaction is persisted into a relational database from which it will be further processed by the *Persistence Tier*. This tier consists of a relational Database Management System (DBMS) and all other ancilliary services required to keep track of the location of each individual transaction within a JSwitch instance. The reason for choosing a relational technology is because of its maturity and availability. Nonetheless, this layer can be implemented using some other persistance mechanism available to the users. Furthermore, this layer can be implemented using a load-balancing DB cluster which will result in improved performance and scalability if necessary, or a fail-over DB cluster if data replication and system availability are desired.

Once a batch of transactions has been persisted, the next step is the process of routing the batch of transactions to the proper transaction manager. This process is handled by the components located at the *Processing Tier*. This tier consists of two types of components: a Transaction Dispatcher (TDisp) and Transaction Processors (TProc). The TDisp takes care of monitoring the persistent store to find new transaction batches that have been validated and persisted. For each of those batches, the TDisp will then look into a pool of available TProcs to find one to which the batch gets assigned. This TProc is the transaction processing component with the business logic necessary to process the operations in the transactions contained in the batch. The TProc can be a Transaction Processing Monitor (TP Monitor), an On-Line Transaction Processing (OLTP) engine, or some other type of server application used to process transactions.

3.2 JSwitch Clients

A JSwitch client is an entity that submits batches containing related transactions. Transactions are related in terms of the application that generated them. Examples of such transactions include medical bills, purchases for movie tickers, credit card charges, and so on. Very often, client application submit these transaction in batches for the convenience in terms of computational resources. For example, a doctor might wait until later afternoon to submit a batch with all the insurance claims the JSwitch clients has accumulated during the day. Although, the physician

could use a model or broadband connection to submit the transaction in real-time, they often prefer to wait to submit in batches to prevent any Internet-related problem to get in the way of managing their patients data. In other instances, the user might be out of the office accumulating the transaction on a portable device, such as a PDA or laptop computer. Only when the user returns to the office can he/she transfer all transaction into the TP system. Notice that wireless and broadband communication might allow users to submit transactions one by one. But in many instances companies do not follow this approach out of concerns for the security of the data or the cost of the communication infrastructure. Thus, we can expect then batched transactions to continue in the foreseeable future.

The batches holding the transactions must satisfy several constrains. First, batches must meet all the well-formedness and validity constraints given in the language used to encode them. In our case, they must follow the constrains of eXtensible Markup Language (XML) specification [16], since this is the format used to encode them. Second, these batches must contain metadata about the nature of the batch and the client. These metadata will be used to identify the unit that can process the batch. In addition, the metadata will enable the client to track the status of the batch. In our JSwitch framework, these metadata consists of: a client Id, a batch Id, and batch creation and sent times. Figure 3.2 displays an example of what a batch might look like, sent by a client for an entertainment ticket purchase. As we can see from the figure, the batch begins with some metadata used for identification purposes, and then contains a set of transactions. In this case, each transaction represents a purchase of one or more tickets for a given event.

The JSwitch client can be a Web application or a fat client, with the ability of creating well-formed and valid XML batches. In addition it has to be able to connect to Web services since this is the mechanism used for the communication between the clients and a JSwitch framework solution. The Batch Transaction Coordination Protocol specifies the complete interaction between a JSwitch client and a solution based on our framework. This protocol is detailed in section 3.5.

It is recommended that client application performs a thorough validation of the data field before submitting them. This helps reduce the burden on the validation efforts done by the business logic components at the JSwitch solution. Furthermore, the need for transactions compensations could be reduced. From the client perspective, validation has also the benefit of reducing the

```
<?xml version = "1.1" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE TicketPR  PUBLIC "http://www.TicketPR.org/DTD/ticketpr.dtd">
<batch>
 <batch-metadata>
  <id>ticketpr-XXXXXX-YYYYYYYYYY</id>
  <creation-sent-time>1367336780375-1367901537529</creation-sent-time>
 </batch-metadata>
 <transaction>
  <id>1234</id>
  <event-place>The Nutcraker – Coliseo Miguel Agrelot </event-place>
  <date-time>23-11-2006:8:00pm </date-time>
  <classification>ballet</classification>
  <price>50.00</price>
  <quantity>2</quantity>
  <credit-card-id>827043JKHNSD</credit-card-id>
 </transaction>
 <transaction>
  <id>1235</id>
  <event-place>Titanic – Plaza de Las America CineVision </event-place>
  <date-time>02-09-2006:8:00pm </date-time>
  <classification>ballet</classification>
  <price>15.00</price>
  <quantity>5</quantity>
  <credit-card-id>257309APRNLW</credit-card-id>
 </transaction>
 <transaction>
  <id>1236</id>
  <event>Rolling Stones – Coliseo Miguel Agrelot </event>
  <date-time>17-10-2006:9:00pm </date-time>
  <classification>concert</classification>
  <price>115.00</price>
  <quantity>2</quantity>
  <credit-card-id>419504YVEHSL</credit-card-id>
 </transaction>
 .
 .
 .
</batch>
```

Figure 3.2. TicketPR client entertainment XML batch

probability that a batch will be rejected by the JSwitch system. Often, those rejection are the result of syntactic or semantic errors in the transactions submitted with the batch. For example, insurance companies tend to return entire batches with medical claims if they find a few malformed claims. Then the client must re-submit the batches and keep some reconciliation information. Validating client applications early and often will reduce the likelihood of major problems when a batch is submitted.

3.3 JSwitch Server Components

A JSwitch implementation is as a collection of Java [41] server objects whose functionality is exposed via Java Web services [20]. This implementation task was done using the Apache Axis2 Soap Toolkit [1]. The JSwitch Services communicate by exchanging Soap messages based on RPC calls, with support for Soap messages with attachments. These attachments are XML-based, and are marshalled and unmarshalled with the use of the JAXB API. This solution is simpler and more efficient than alternative solutions such as SAX and DOM, respectively. Each server object leverages on the built-in thread capabilities in Java to generate derived objects and methods calls that are run on other threads independent from the one running the server object. Thus, a JSwitch solution can easily leverage on multi-processor or multi-core hardware for optimal performance. These next subsections present details of the functionalities and specially on how these JSwitch Server Components (JSC) were implemented. Additional, specific implementation details like software versions are are available in Appendix A.

3.3.1 Transaction Coordinator (TCoord)

The Transaction Coordinator (TCoord) serves as the entry point to the system. The communication and interaction among the clients and this Java server object is through the use of Web services at all times. The TCoord responsibilities are a) client authentication into the system, b) selection of an adequate TColl to handle the client request for batch processing, c) tracking the load metric on the various TColl available to the system, and d) re-distribution of batch processing tasks in case of a TColl failure. Arguably, the single most important task for the TCoord is that of selecting a TColl to handle a batch. The selected TColl must be a capable one in terms of avail-

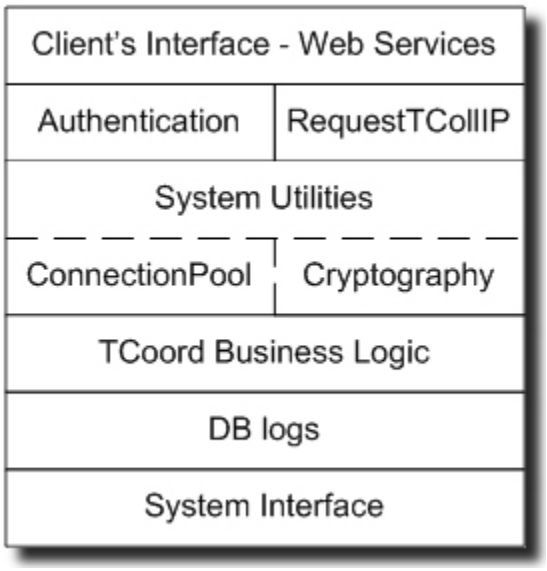

Figure 3.3. TCoord internal organization

ability and the best suited in terms of current load. The TCoord uses a selection policy to perform this task. This policy is configurable, and currently the framework provides four principal modes to choose: a) Round Robing assignment, b) Random assignment, c) Least Loaded assignment, and d) Random-Least Loaded assignment. Some of these modes offer a tuning parameter that can be set to system specific requirements. We shall discuss these policies and their tuning parameter with more detail in the next section. The main goal of these policies is to try to keep the load of the system balanced between the different TColls, therefore providing improved response time to the system users and efficient utilization of system resources.

Figure 3.3 displays a layered view of the internal organization of the TCoord. We can see that it has a Client Interface layer used to manage client connection. The Authentication module validates the user to prevent unauthorized access to the system. The RequestTCollIP module deals with finding the target TColl to handle a given batch. The System Utilities layer contains two sub-systems. The first is the ConnectionPool module, in charge of managing the connection to the DBMS used for persistence. The TCoord uses this module to access the metadata for the TColl in the system. The Cryptography layer provides the mechanism for assisting in user authentication. The business logic for coordination and scheduling are implemented in the TCoord Business Logic layer. Meanwhile, all logging operations are handled by the DB Logs layer. Finally, the System

Interface is the layer used to send messages to the TColl via tuples stored in the DBMS.

3.3.2 Transaction Collector (TColl)

The Transaction Collector (TColl) is a multi-threaded server application whose role is to clean and prepare the transactions. Its functionality is exposed as a Web service, and each call is handled by a thread within the main TColl process. The TColl receives a batch of transactions from the Web, and breaks up this batch into individual transaction elements. These batches are encoded in XML, with the appropriate delimiters to separate each transaction. We used the functionality of Axis2 SOAP with attachments to make the batch an attachment to the SOAP message. We argue that this arrangement produces a more efficient solution than simply sending each transaction as an independent Web service method call. In JSwitch, each transaction is validated, and those that are incorrectly specified are sent into a file designated for this purpose. The rest are stored into the relational database for further processing.

The client which submitted the batch gets a report indicating the transactions that were accepted and those that were declined. For those transactions that were declined, the system includes a brief description of the error. In addition, the report includes a URL for the client to download the file with the declined transactions. Once this process is completed, the TColl writes a special tuple to the relational database, making the batch ready for processing by the remaining components of JSwitch. TColl acts as a "hot plugable" component since it has the ability to register itself with the remaining JSwitch components to make them aware of its availability for batch processing. The TColl also provides a Web service method called Is-Alive, used by the TCoord to periodically check if the TColl is active or not. Each TColl writes information on its system load and current workload to the database that stores statistics. This information is used to update statistics used to track the current load on the machine running the TColl. This information is used by the TCoord at the moment of choosing a TColl to handle a client request.

Figure 3.4 displays a layered view of the internal organization of the TColl. The First two layers in the TColl are used to manage the connection with the client, and to read submitted batches. The System Utilities layer contains the following components: a) Thread Pool - module to handle thread allocation, deallocation and dispatch to work, b) Connection Pool - module used to

handle connections to the DBMS, c) PCQueue - module used to handle the queue that stores work requests with batches to be processed by worker threads, and d) XML Processor - module used to parse and validate the format of the XML-encoded batch. The TColl Business Logic module contains the code that decomposes the batches into the individual transactions, and stores them into the DBMS. The IsAlive WS module is used to reply to TCoords asking if the TColl is still alive. Meanwhile, the DB logs keeps of the all logs generated while processing a batch. Finally, the System Interface layer contains the logic to actually access the DBMS to write the batches to disk.

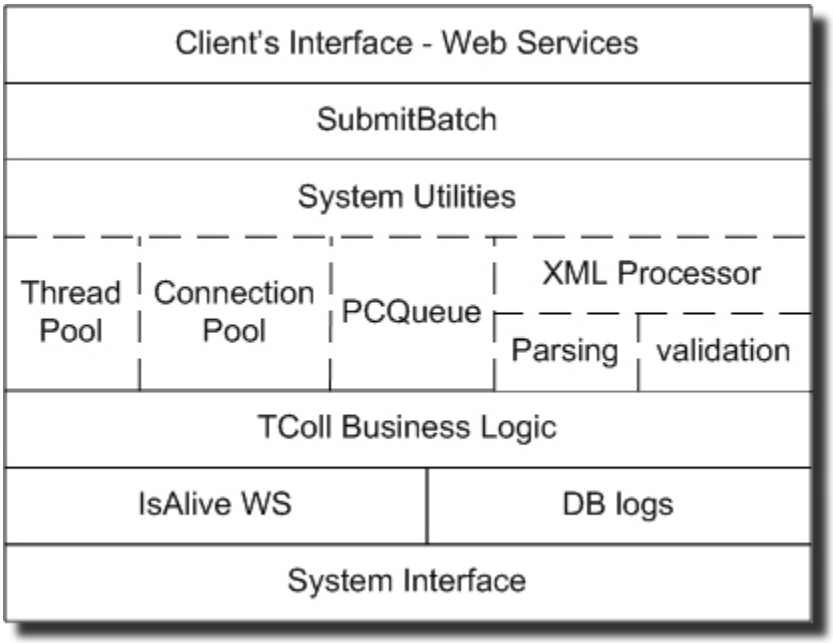

Figure 3.4. TColl internal organization

3.3.3 Transaction Dispatcher

The Transaction Dispatcher (TDisp) takes care of assigning already prepared batches of transactions to capable and proper TProc elements. The TDisp is also a multi-threaded server application. The TDisp has one thread which continuously monitors the database, looking for batches that are ready for processing. When one such batch is found, this thread invokes a worker thread to handle the batch (handle in this case refers to dispatch the batch to the proper TProc). This is done by placing the batch on a producer consumer queue (PCQueue), from which worker threads can pick batches that are ready for processing (with dispatch status in the TDisp context).

This implementation follows the PCQueue application based on the Command Pattern. The worker thread then queries the TProc catalog in order to apply a scheduling policy to determine a capable TProc of handling the batch. It also gets the information about the load metric on each candidate TProc. After a TProc has been determined by the scheduling policy the worker thread inserts a tuple with the batch metadata at the JSwitch DB Server at the workDispatcher table where the TProcs search for their assigned batches. This is referred as the dispatch action.

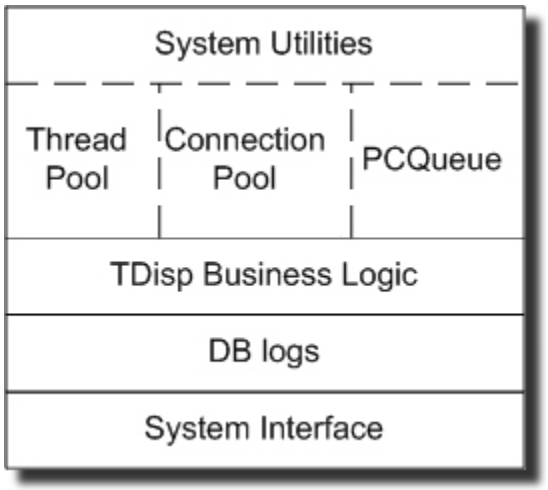

Figure 3.5. TDisp internal organization

Figure 3.5 displays a layered view of the internal organization of the TDisp component responsible for accomplishing the aforementioned functionalities. We can see that it has a Client Interface layer used to manage client connection. The System Utilities layer contains to sub-systems. The first is the Thread Pool which is the module that handles thread allocation, deallocation and dispatch to work. The ConnectionPool module is in charge of managing the connection to the DBMS used for persistence. The TDisp uses this module to access the metadata for the TProc in the system. The business logic of this component uses a specified scheduling policy at the SelectTProcIP function to determine the target TProc that a batch will be dispatched for its processing. Meanwhile, all logging operations are handled by the DB logs layer. Finally, the System Interface is the layer used to communicate with the TProc via tuples stored in the DBMS.

3.3.4 Transaction Processor

A Transaction Processor (TProc) is a server application that contains the business logic required to perform the different transactions that a batch requires. This component can be implemented using an off-the shelf solution such as an OLTP system, a TP Monitor, or a custom-made solution that integrates systems such as workflow engines. Additionally, the JSwitch framework could use a taskflow engine in the Processing Tier to accomplish the business logic required by the user requests, in combination to the already mention transaction processing applications. Basically, this off-the shelf solutions autonomously process the transactions that had gone through the Preparation and Persistence tiers of the system.

In the JSwitch framework, the TProc component consists of two sub-systems. One of them is the Processor Adapter, that gets the batch for processing. The Processor Adapter converts the batch into the format in which the Transactional Engine expects its transactions to be input. The Transactional Engine is the actual application that processes the transactions and generates a result for each one of them. In addition to containing the business logic to execute the transactions, this component, similar to the TColls, is **hot plugable** since it has the ability to register itself when online and maintains system metrics about its own system and load. This information is used by the TDisp when choosing which TProc will handle, and process the batch transactions prepared for processing.

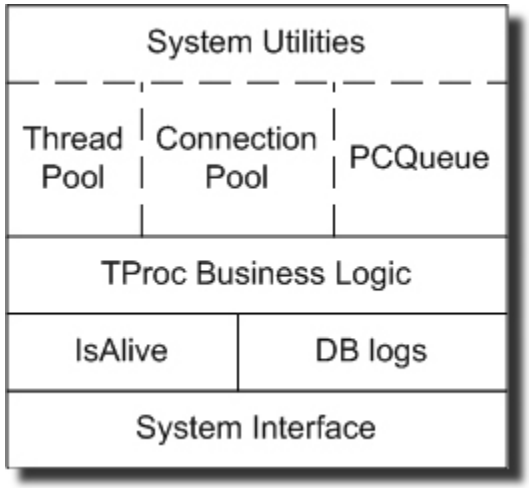

Figure 3.6. TProc internal organization

Figure 3.6 displays a layered view of the internal organization of the TColl component. The First layer is the System Utilities which contains the following components: a) Thread Pool - module to handle thread allocation, deallocation and dispatch to work, b) Connection Pool - module used to handle connections to the DBMS, and c) PCQueue - module used to handle the queue that stores work requests with batches to be processed by worker threads. The TProc Business Logic module contains the code that signals the OLTP unit to perform the transaction processing. The IsAlive WS module is used to reply to TDisp asking if the TProc is still alive. Meanwhile, the DB logs keeps of the all logs generated while signaling a batch for processing. Finally, the System Interface layer contains the logic to actually access the DBMS to write the signaling events to disk.

3.3.5 Database Server

The JSwitch DB Server is a relational database whose role is to store the transactional batches submitted by the users. Each of these batches eventually reaches a particular TProc, which will also store information about the transaction. However, the user shall not confuse the role of the DB server, which is more of a catalog of transactions that have been received and/or processed. For example, if the TProc is an OLTP system implemented with a relational database (e.g. Oracle), then the TProc will store information specific to the business logic needed to process the transaction. But the JSwitch DB Server simply stores the batch and its metadata; the DB server does not store any information related to the processing of the transaction. In our current implementation of the persistance layer, we use the PostgreSQL 8.1 DBMS at the DB Server.

3.3.6 JSwitch System Application Database

The JSwitch Application Database (JSwitch ADB) ER-Diagram provides a full description about the data used at the JSwitch framework in terms of entities and relationships. The JSwitch ADB was designed and developed with the purpose of storing all the data needed for the JSwitch framework. Basically the JSwitch ADB contains metadata about the system components, system clients and the batches that these submit along with metadata of the transactions included at the batches.

The Entity-Relationship (E-R) Diagram used for describing the JSwitch ADB is displayed in Figure 3.7. This diagram presents all the entities of the system along with the relationships

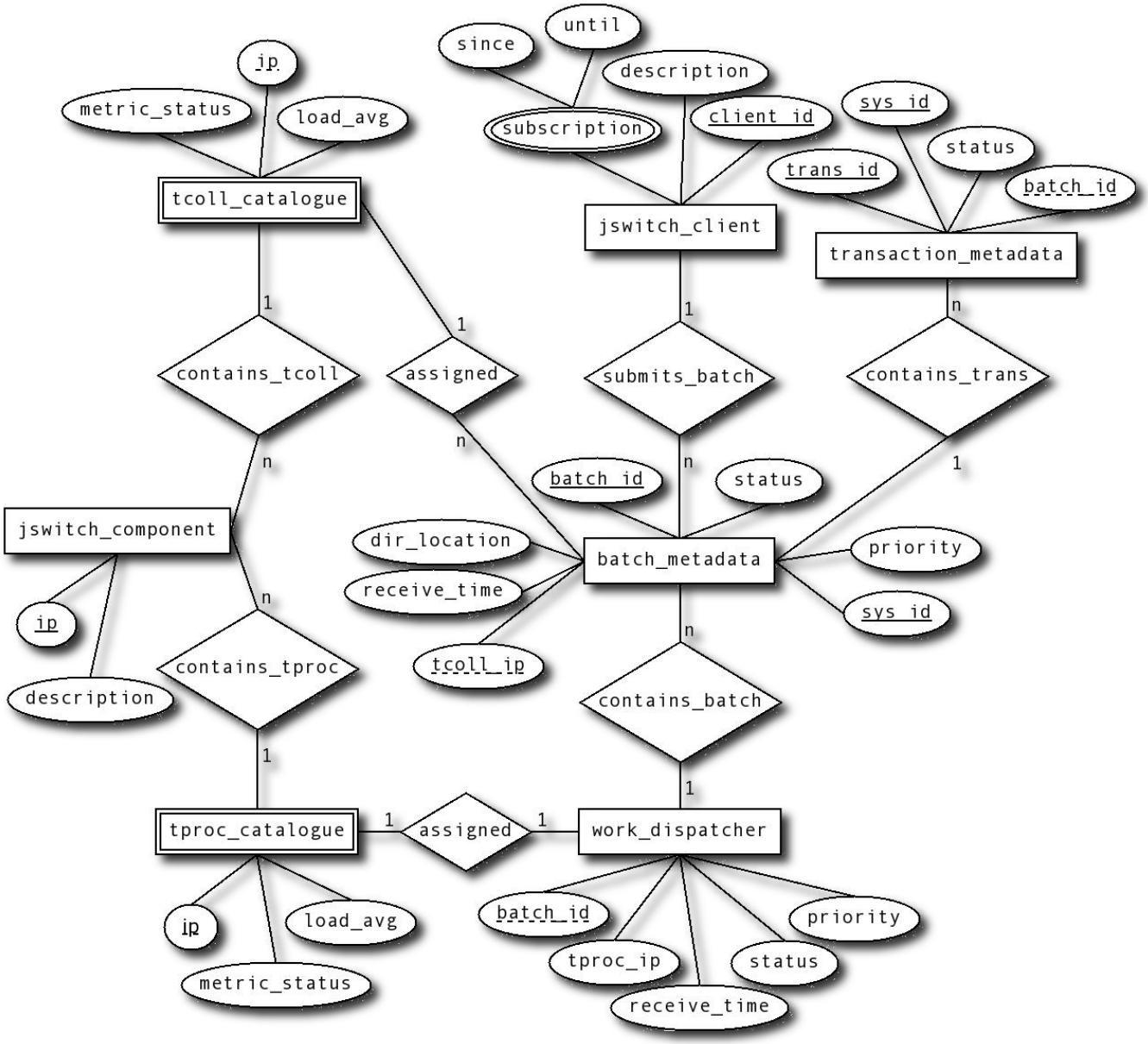

Figure 3.7. JSwitch application database ER-Diagram

between them. Basically, the boxes represent entities, double boxes represent weak entities, ovals represent their attributes, double ovals are composite attributes, and diamonds represent the relationships between entities. Underlined attributes represent primary keys and the underlined dashed attributes represent the foreign keys. Numbers and letters at either side of relationships represent their mappings: 1:1 (one to one) or 1:n (one to many) for this ER-Diagram.

For the JSwitch ADB, the following entities were identified, their respective attributes are detailed at the ER-Diagram:

- jswitch_client - this entity refers the clients that are registered at the system for transaction batch submission. These are detailed at a later section but some of their more relevant attributes are a client_id assigned by the JSwitch and the subscription period.
- batch_metadata - represents transaction batch instances. Basically the attributes contain the necessary information to identify, locate and know the current status of the batch at any given moment. A priority field is provided but for now it is only used for signaling the order as batches arrive along with the receiving time. Finally the system also assigns an id to each batch for tracking purposes.
- transaction_metadata - represents transaction instances; basically the attributes identify the transaction with their own business logic id along with one assigned by the system for tracking purposes. Also, each transaction knows its parent batch, and a status attribute is provided to know the current status of the batch at any given moment.
- jswitch_component - represents the components at the JSwitch framework This entity provides a general description and identification for any given component within the JSwitch implementation.
- tcoll_catalog - this entity represents the metadata about any given TColl, it contains the metric status, and system and batch load metrics (i.e. number of batches assigned to a given component).
- tproc_catalog - this entity represents the metadata about any given TProc, it contains the metric status, and system and batch load metrics (i.e. number of batches assigned to a given component).

- work_dispatcher - basically this entity contains the information about the transaction batches that have been already prepared and persisted, and have been assigned to a particular TProc for its later processing.

The following relationships between the entities were also identified and are displayed in the Figure 3.7:

- submits_batch - refers to the event of a client submitting a transaction batch.
- contains_transaction - refers to the nature in which transactions arrive to the system, inside the batches.
- assigned - the relationship of assigned between the batch_metadata entity and the tcoll_catalog. Refers to the transaction batches that have been asssigned to a given TColl for its proper preparation.
- contains_tcoll - represent the fact that the table tcoll_catalog is a weak entity associated with the jswitch_component.
- contains_batch - refers to the nature in which transaction batches are dispatched by the systems; when a batch has been found prepared is dispatched to a TProc
- assigned - the relationship assigned between the work_dispatcher entity and the tproc_catalog refers to the transaction batches that have been dispatched to a given TProc for its proper processing
- contains_tproc -represents the fact that a tproc_catalog is a weak entity associated with the jswitch_component.

As detailed in the ER-Diagram theory, all entities and some of the relationships translate into or have equivalent tables in the database. The result of this translation is the databases Relational Schema which is presented completely in Appendix A.

3.4 JSwitch Resource Scheduling

The JSwitch architecture devotes its efforts towards an efficient coordination of transactions in the Web environment. The system components enforce this efficiency by employing several policies that seek to balance loads among system components. The idea is to make the assignment

of transactional batches as efficient as possible. There are two main components of the JSwitch framework on which these policies are used: the TCoord and TDisp. In the other hand each TColl and TProc component tracks several usage statistics about their resources, needed to estimate the current system and work load (i.e. number of assigned batches) at each host in the JSwitch deployment site.

3.4.1 Round Robin Scheduling

The Round Robin Scheduling Policy (RRS) is the simplest of the mechanisms used for batch assignment in JSwitch. It can be used by either the TCoord or the TDisp. As batches of transactions arrive, the TCoord assigns them to the pool of TColls in round-robin fashion. Figure 3.8 clearly displays the RRS behavior when assigning batches to a pool of TColls. Likewise, as batches become ready for processing, the TDisp assigns them in round-robin fashion to the pool of TProcs. This is a starvation-free type of scheduling algorithm that does not enforce any kind of priority and produces an uniform work distribution among system components. The main advantage that this policy offers to the JSwitch architecture is uniform work distribution, which translates into equal full utilization of the components whithin the TColl and TProc collections. We should expect the best outcome of this policy when workloads are homogeneous and priority is not required among works.

3.4.2 Random Scheduling

This Random Scheduling (RS) policy assigns batches at random to the target components in JSwitch. It is used by both the TCoord and the TDisp. The RS policy is also starvation free and does not enforce any kind of priority. Randomization in general is used to eliminate unknown problems and is known to produce a normal distribution among the components in the system [29]. The features that this policy brings to the JSwitch architecture are full utilization of the components within the TColl and TProc collections along with low probability of hot spots in the presence of heterogenous batches of transactions. Figure 3.9 displays a TCoord assigning batches to a pool of TColls, the RS policy behavior when assigning these batches is appreciated here. Also this same behavior can be seen when it is applied by a TDip to dispatch batches among TProcs.

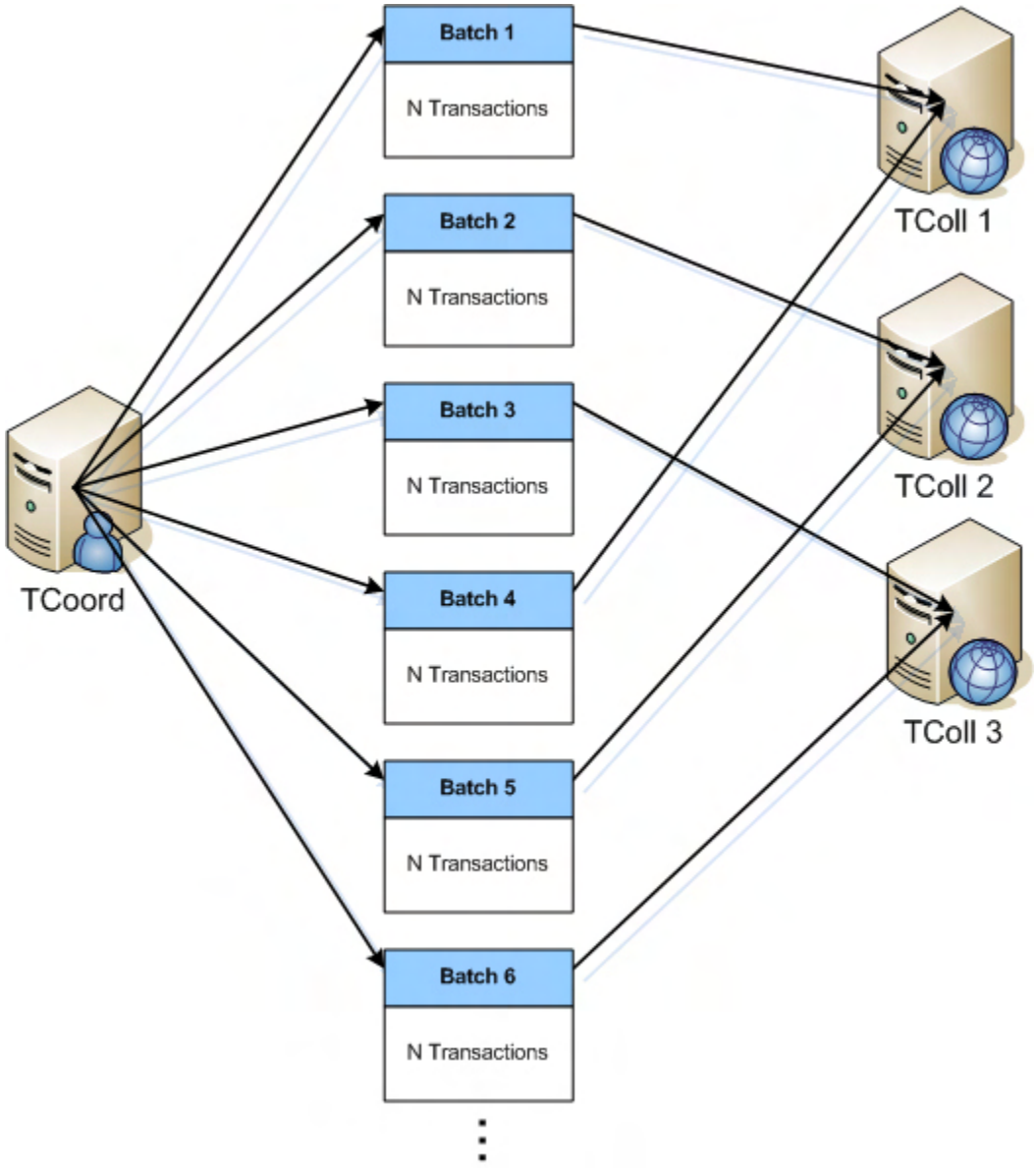

Figure 3.8. RRS behavior when assigning batches to a pool of TColls

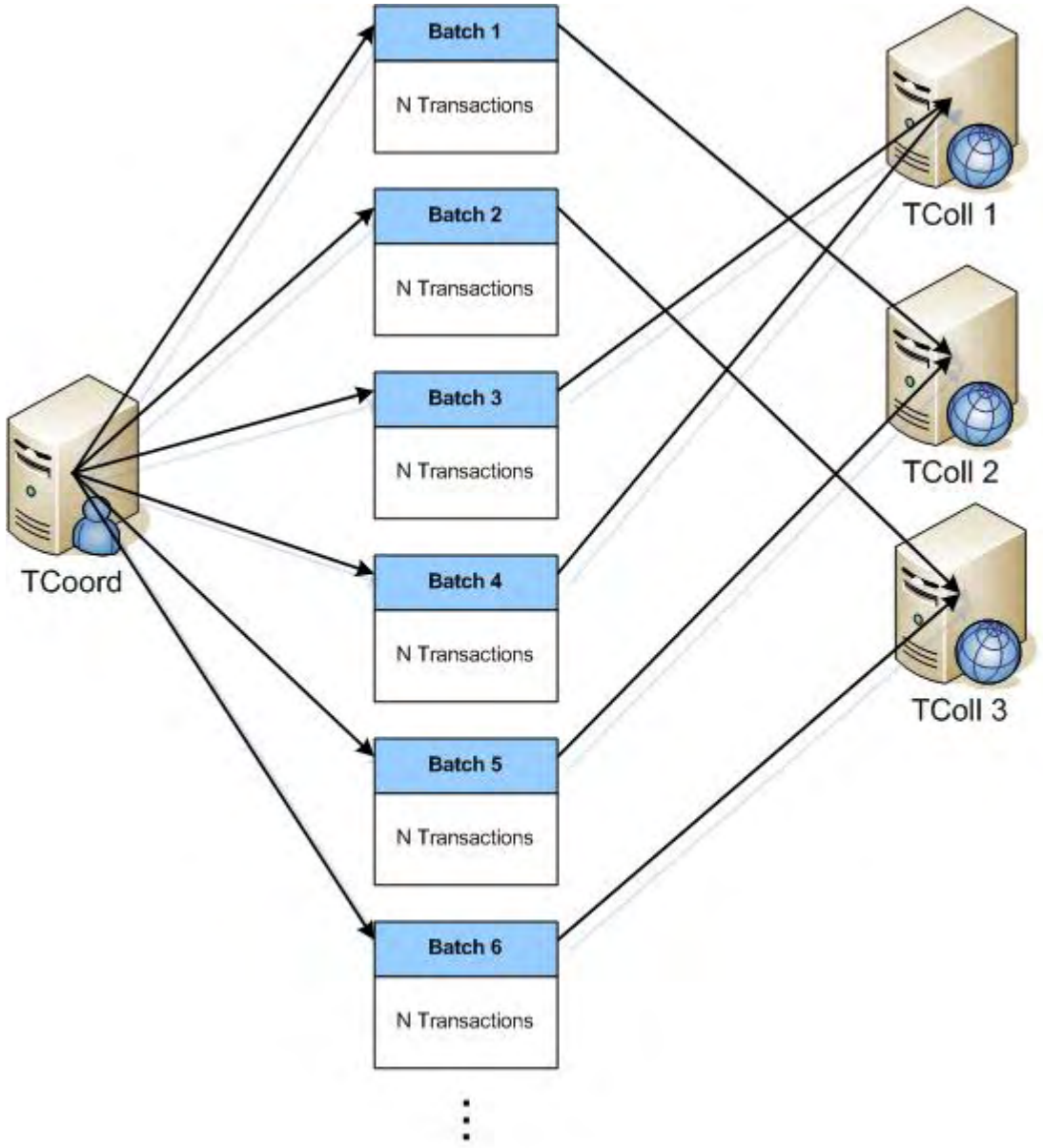

Figure 3.9. RS behavior when assigning batches to a pool of TColls

3.4.3 Least Loaded Scheduling

The Least Loaded Scheduling (LLS) Policy is a type of Proportionally Fair Algorithm that is driven by a priority function. This priority function describes the data rate that is potentially achievable at the present moment at a particular system component. We studied two options for the priority function for the JSwitch architecture. The first one was the average machine load during the past 30 seconds at the host running a particular component (either TColl or TProc). The second option was the work load in the machine in terms of number of batches that have been assigned to it.

Chapter 4, of experimental results has a corresponding section for the analysis of scheduling policies, detailing the findings when applying different priority functions to a Jswitch solution. Figure 3.10 displays a TCoord assigning batches to a pool of TColls using the LLS policy. At first all the batches go to the least loaded component, which is at the top of the list. However, as new batches are assigned to this component, it become more loaded. So, the next few batches are assigned to the next component in the list which is now the least loaded component. When this component becomes more loaded, the third component in the list begins to get batches. Clearly, the list of components must be kept sorted by the amount of load. Components with light load will move to the top of the list, while those components that become saturated move to the end of the list. This approach can be easily implemented using a priority queue. The use of this policy should produce the best component selection for the batch at hand. We argue that this policy, when fine tuned, can bring high efficiency to a JSwitch deployment solution.

3.4.4 Random Towards Least Loaded Scheduling

Finally, the Random Towards Least Loaded Scheduling (RTLLS) policy presents a hybrid of the last two scheduling policies (random and least loaded). It seeks improved efficiency when performing transaction coordination at the JSwitch architecture. Basically, we merge the described past two algorithms with the objective of building their strengths into a new scheduling policy. The idea is to require a less precise load metric from the system. Hence, we do not incur in the extra overhead of keeping rather precise information on the load of each JSwitch component. In our approach we chose a subset of the components that are top X percent of the list of least

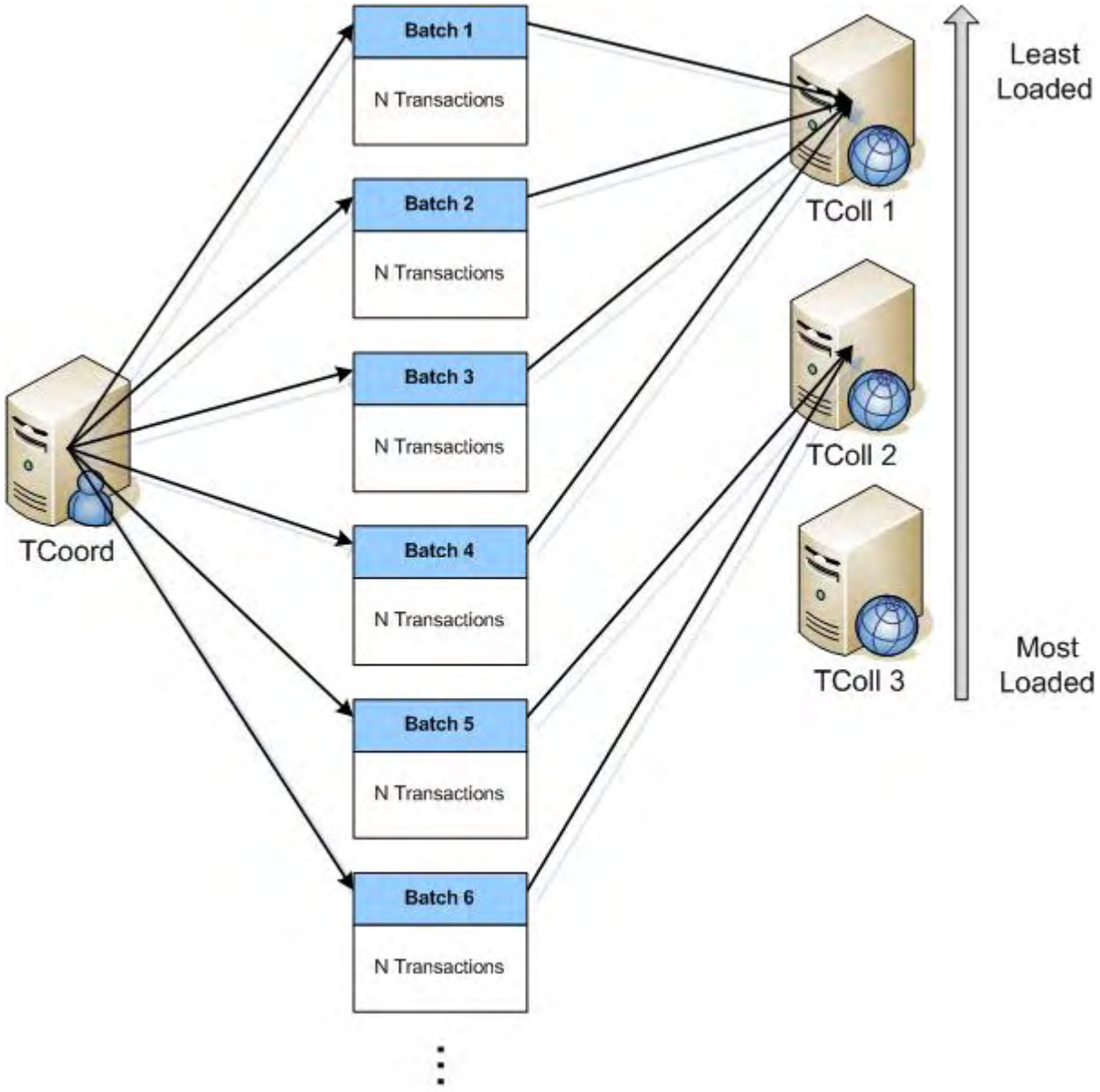

Figure 3.10. LLS behavior when assigning batches to a pool of TColls

loaded components. Here X represent a percentage value such as 40% or 50%. From the top X least loaded elements, we chose one at random and assign the batch to it. This approach prevents making the top component from becoming a hot spot too quickly, but at the same time assigns batches to lightly loaded elements.

Figure 3.11 displays the RTLLS policy used by a TDisp JSC component to assign batches to a pool of TProcs. In this case the scheduling policy is used among the top half (50%) of the components ordered from the least loaded to the most loaded. This scheduling policy will use the best priority function found among the options developed for a JSwitch solution.

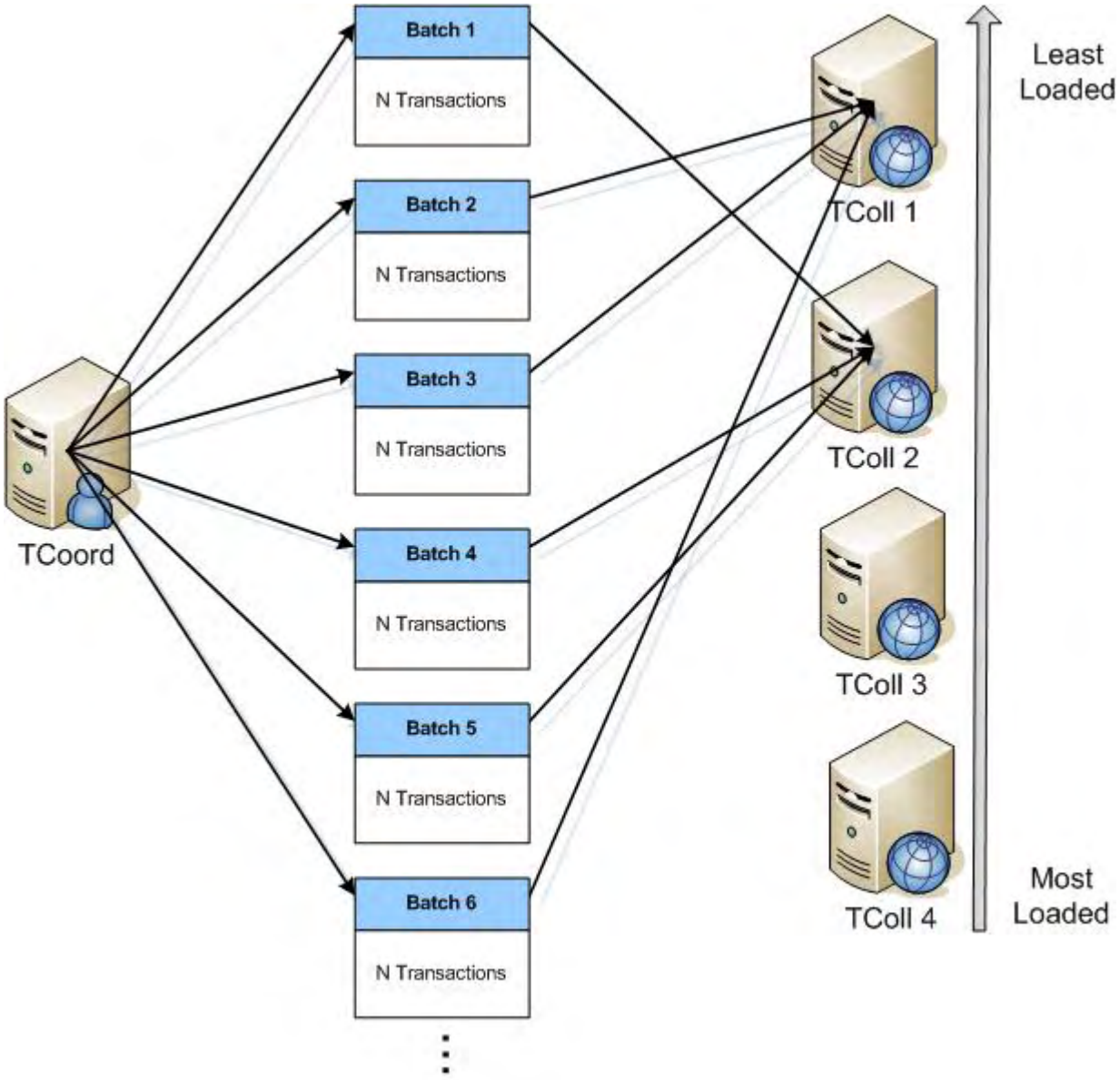

Figure 3.11. RTLLS-Half behavior when assigning batches to a pool of TColls

3.5 Batch Transaction Coordination Protocol

The Batch Transaction Coordination Protocol was envisioned as the mechanism to describe the interactions among the components at the JSwitch architecture which were already detailed in the previous sections. This protocol basically defines the exchanges of data and process calls across system components and even architecture tiers. Figure 3.12 displays these interactions in an exchange diagram and a step by step explanation of what is happening at each step is given next.

- Step 1 - When a user decides to submit a batch of transaction to the JSwitch system, it calls the clientAuthentication() Web service method at the TCoord JSC from its web client with the already agreed client key.
- Step 2 - The ClientAuthentication Web service at the TCoord JSC connects to the DB Server to look up the received client key at the designed store table, if it is valid verifies if it has not expired or is not in any other in-valid state.
- Step 3 - The DB Server acknowledges key validity.
- Step 4 - The ClientAuthentication Web service at the TCoord JSC acknowledges the client about its authentication.
- Step 5 - Once these initial steps have been completed, the client calls the requestTCollURL() Web service method at the TCoord JSC.
- Step 6 - The TCoord verifies that the client key is valid for the session that was already created at the authentication step. If the key is valid, the TCoord connects to the DB Server to look up TColl JSC statistics from the TColl catalog (metadata).
- Step 7 - The DB Server returns the TColl catalog information to the TCoord (metadata).
- Step 8 - Based on the information acquired, provided by the TColl catalog, a scheduling policy at the TCoord determines the URL of a capable TColl and sends it to the client in the RequestTCollURL Web service return value.
- Step 9 - At this stage the client calls the SubmitBatch() Web service method at the TColl URL already acquired which contains the XML batch file.

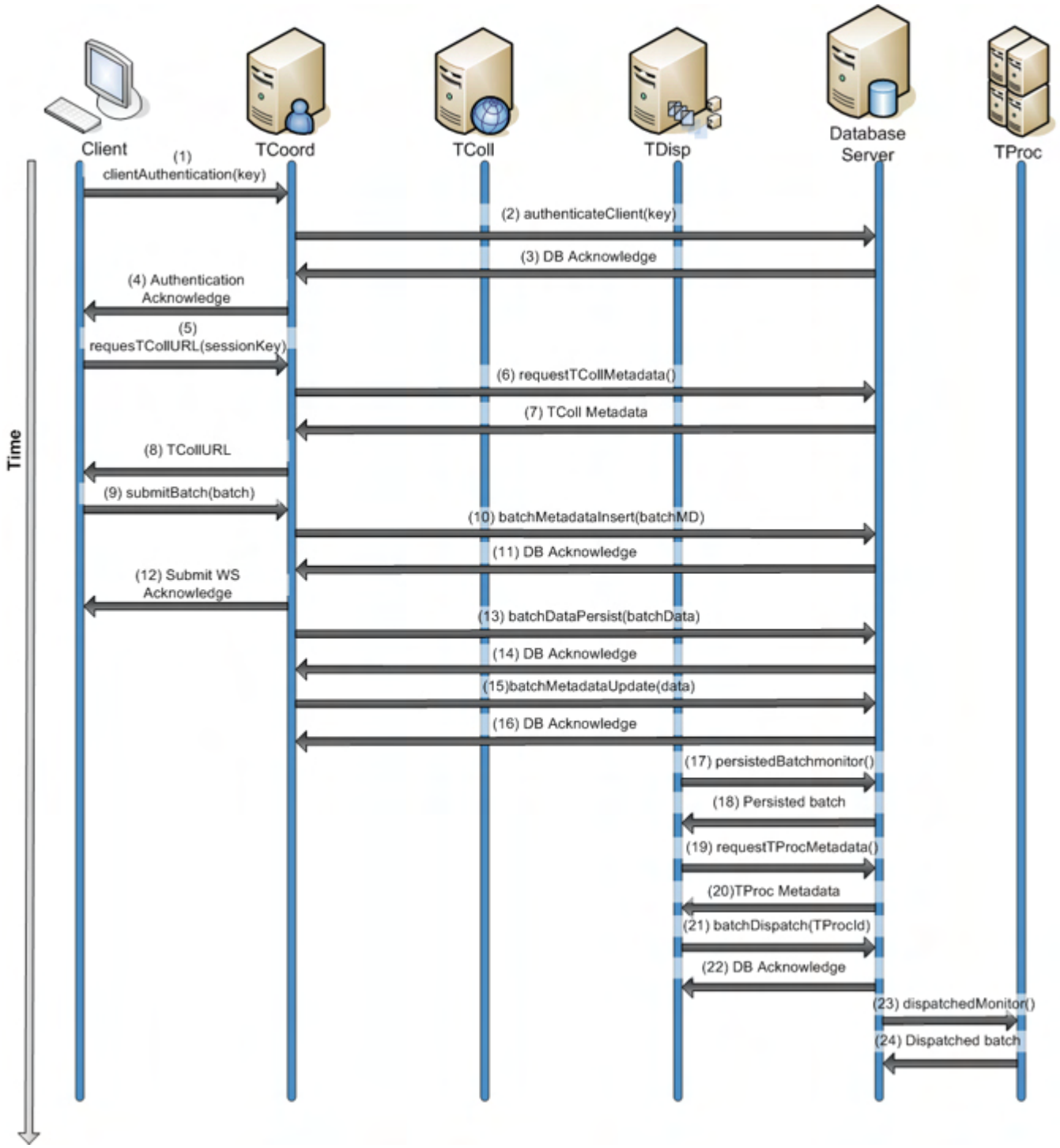

Figure 3.12. Batch Transaction Coordination Procol Exchange Diagram

- Step 10 - The SubmitBatch Web service first uploads the batch to the designed repository at the TColl then inserts a tuple at the batch metadata table with its corresponding batch information.
- Step 11 - The DB server acknowledges the TColl about the metadata inserted.
- Step 12 - An acknowledge about the Web service result is sent to the client which ends the client interaction with the system.
- Step 13 - Some time T after the client receives the SubmitBatch() Web service result acknowledgment the TColl starts the batch validation and persistence phases. Basically the TColl reads the uploaded file from its designed repository, validates it, persists it into the designed storage facility.
- Step 14 - The DB server acknowledges the TColl about the batch data inserted.
- Step 15 - Upon acknowledge recieval the TColl updates the status of the batch at the batch metadata table as persisted.
- Step 16 - The DB Server acknowledges the TColl about the update committed.
- Step 17 - The TDisp that is monitoring the batch metadata table detects update notification indicated that the batch was persisted to the DB.
- Step 18 - The DB Server acknowledges the TDisp about the persisted update finding.
- Step 19 - The TDisp connects to the DB Server to look up the TProc JSC statistics from the TProc catalog.
- Step 20 - The DB Server returns the TProc catalog information to the TDisp.
- Step 21 - Based on the information acquired from the TProc catalog, the scheduler at the TDisp determines a capable TProc to dispatch the batch work. The TDisp connects to the DB Server to dispatch the newly discovered batch work.
- Step 22 - The DB Server acknowledges the TDisp about the committed batch work dispatchment.

- Step 23 - At any time T the TProc that monitors for batch dispatched detects the newly assigned job and informs the off-the shelf transaction processing component designed to attend the batch of its existence.
- Step 24 - The DB Server acknowledges the TProc about the dispatch update finding.
- Step 25 - At this point the transaction coordination functions of the JSwitch are completed.

3.5.1 Component Crash-Recovery

The Batch Transaction Coordination Protocol developed as a key feature of the framework also implements crash-recovery at a basic level. The scope of crash-recovery in the framework is in case of Transaction Collector or Transaction Dispatcher failure. The framework provides a constant crash detection of these components. In the events of a TColl failure the system will perform the steps displayed in the exchange diagram of Figure 3.13. These crash-recovery steps are explained next:

- Step 1 - There is a crash detection function that monitors TColls for failures at the TCoord. When this function is executed, it connects to the database to retrieve the metadata from the TColl catalog.
- Step 2 - The DB Server returns the TColl catalog information to the TCoord.
- Step 3 - The TCoord calls the Is-Alive() Web service method on each of the TColls registered in the catalog.
- Step 4 - The TColls reply with an alive message, or not reply anything at all if they are not available.
- Step 5 - If the TColls reply with an alive message, components are online and nothing is done. In the event that one component does not reply within the designed time, the component is set to off-line at the TColl catalog at the DB Server.
- Step 6 - The DB Server acknowledges the TCoord about the update committed at the TColl catalog.

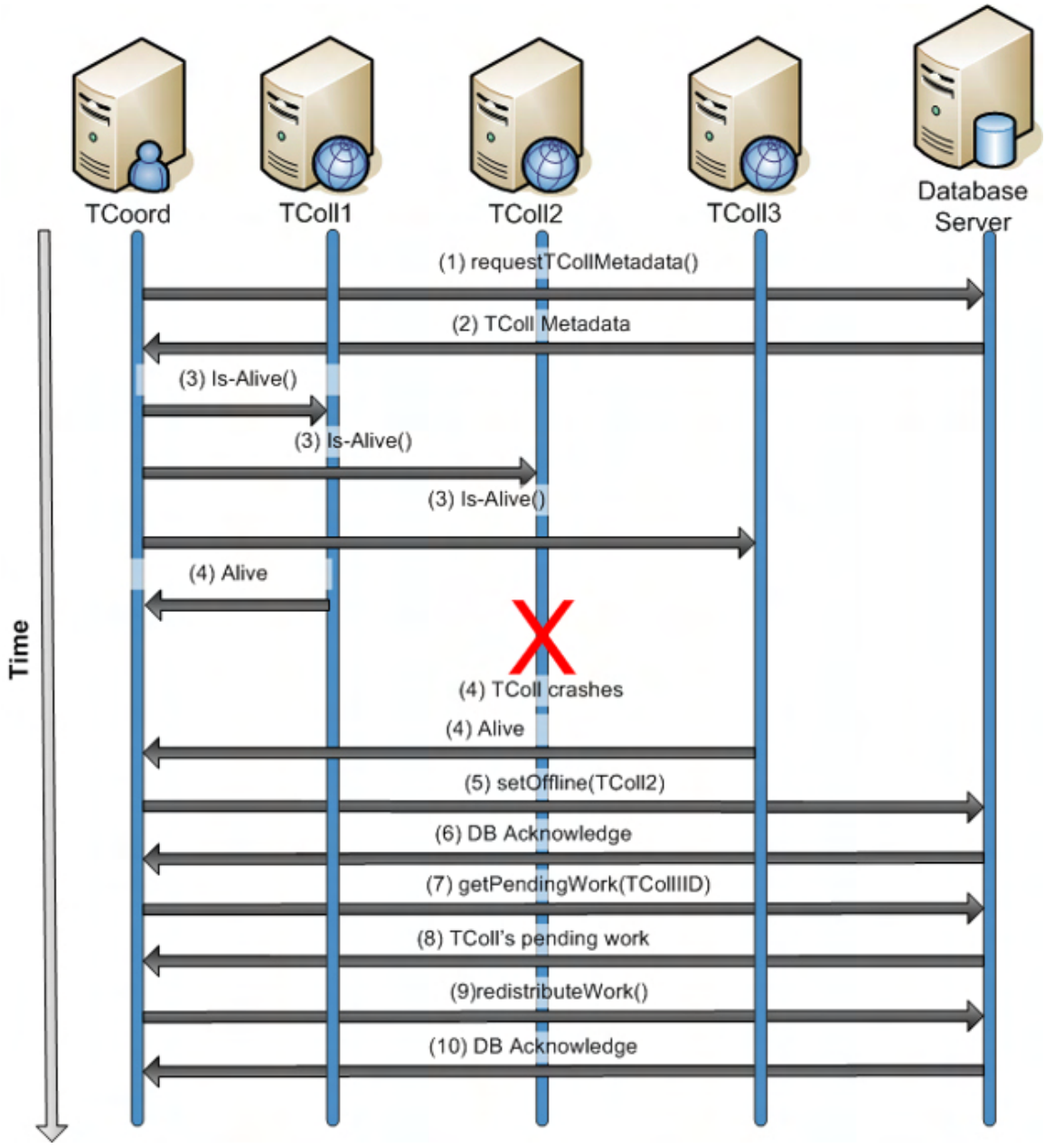

Figure 3.13. Crash Recovery at the events of a TColl failure Exchange Diagram

- Step 7 - At this point the TCoord knows the metadata about the failed TColl component or components and connects to the database to collect the pending work, this is the un-processed batches of each component.
- Step 8 - The DB Server returns the information about the work assigned and pending to the TColl that crash.
- Step 9 - Once this information is acquired, the TCoord re-distributes the work pending from the failed TColl components using the scheduling policy assigned for crash-recovery events.
- Step 10 - Finally the DB Server acknowledges the TCoord about the operations committed and crash recovery is completed.

Similarly, in the events of a TProc failure the system will perform the steps displayed at the exchange diagram in Figure 3.14. These crash-recovery steps are explained next:

- Step 1 - There is a crash detection function that monitors TProcs for failures at the TDisp. When this function is executed, it connects to the database to retrieve the metadata from the TProc catalog.
- Step 2 - The DB Server returns the TProc catalog information to the TDisp.
- Step 3 - The TDisp calls the Is-Alive() Web service method at each of the TProcs registered at the catalog.
- Step 4 - The TProcs reply with an alive message, or not reply anything at all if they are not available.
- Step 5 - If the TProcs reply with an alive message, components are online and nothing is done. In the event that one component does not reply within the designed time, the component is set to off-line at the TProc catalog at the DB Server.
- Step 6 - The DB Server acknowledges the TDisp about the updated committed at the TProc catalog.
- Step 7 - At this point the TDisp knows the metadata about the failed TProc component or components and connects to the database to collect the pending work, this is the un-processed batches from each component.

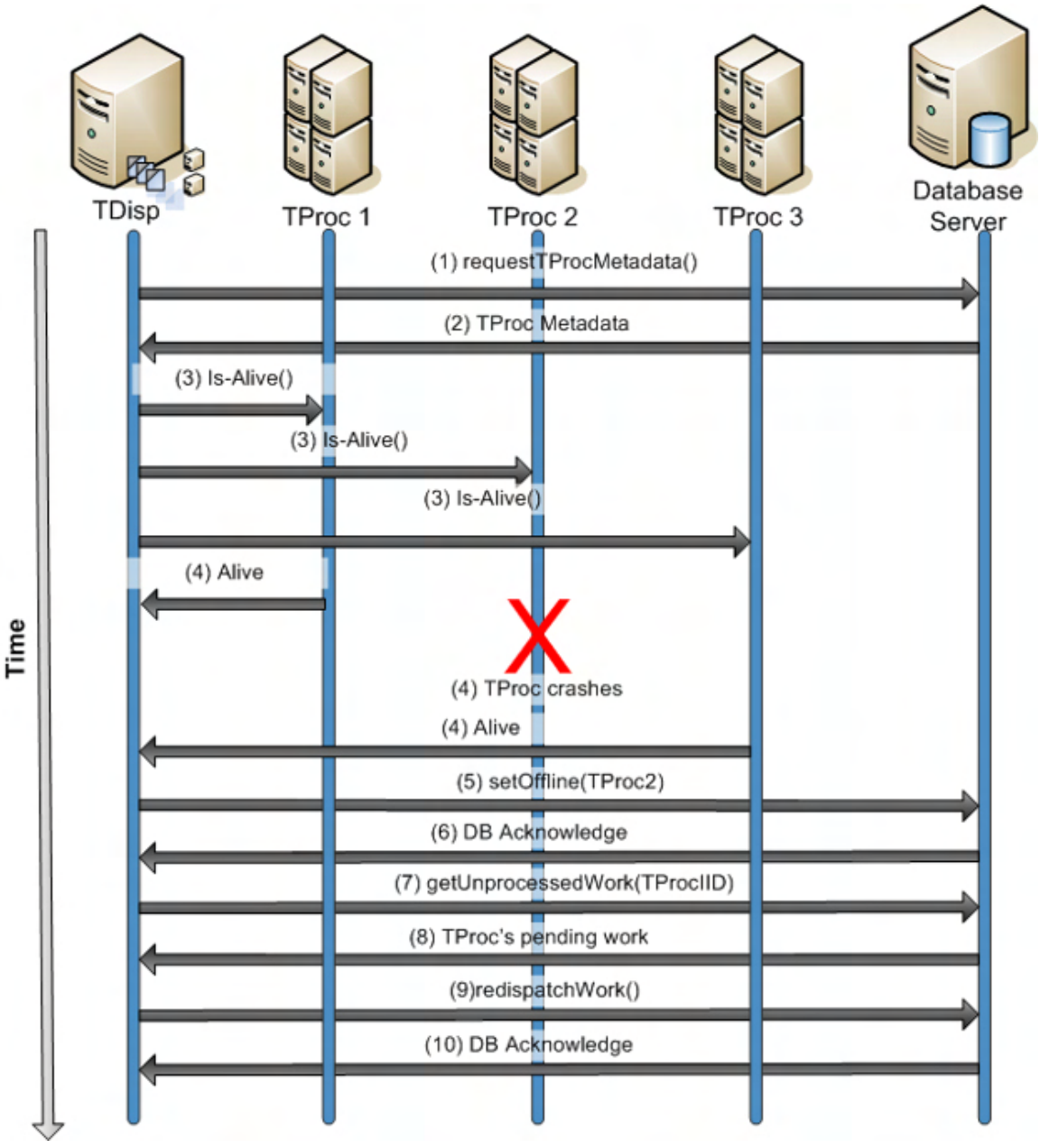

Figure 3.14. Crash Recovery at the events of a TProc failure Exchange Diagram

- Step 8 - The DB Server returns the information about the work assigned and pending to the TProc that crash.
- Step 9 - Once this information is acquired, the TDisp re-dispatches the work pending of the failed TProc components using the scheduling policy assigned for crash-recovery events.
- Step 10 - Finally the DB Server acknowledges the TDisp about the operations committed and crash recovery is completed.

3.6 Performance and Load Analysis

A Traffic Ticket Application based on the JSwitch was developed with the purpose of simulating a real use case of the framework. The implementation experience was only for testing purposes. In the next chapter, we present a series of tests carried out to analyze the performance of the system. This analysis is mainly based in terms of throughput, which refers to the number of transactions performed per second since it is the main quantitative measure of a batch processing system. Additionally, system overall load capacity will be put to the test in order to determine how many users can the system effectively attend without becoming unstable or unusable. The next chapter is dedicated to these topics.

3.7 Security Analysis

The security analysis of the JSwitch framework will only be concerned with the client's proper authentication, authorization, and data integrity. The many areas involving security [11] represents a topic of research on its own that can by conducted later but that it is not under the scope of this research. Currently, there are no Web service security standards widely in use [12] and Web service security is currently a research area with high demand [19] [31]. Based on these facts there is no trusted option for securing Web services, therefore the security is a weakness of the framework since its entry points are Web service based. Alternatively the JSwitch implementation can communicate with it users through the use of the SSL protocol [18]. Finally, there are security mechanisms mentioned [24] and [39] to enforce data privacy that could be also apply to this framework.

CHAPTER 4

Experimental Analysis

A complete JSwitch solution was implemented for the purpose of testing the JSwitch framework. This JSwitch implementation is designed to handle an application for managing traffic tickets and payments, and we provide details in this chapter. We have begun validating the performance of the transaction coordination interaction within the framework known as the Transaction Coordination Protocol, with the ticket traffic solution based on the JSwitch architecture. A set of experiments have been designed with the purpose of acquiring insight into the nature of the system. Two data sets were used to acquire further insights, one with batches with an homogeneous number of transactions and another with an heterogeneous amount per batch. The main goal was to find the system's throughput while *stress-testing* it. We define throughput as the number of transactions coordinated by the system per unit of time. Furthermore, we wanted to determine which scheduling policy produces the best system response for transaction coordination and under what circumstances. This chapter explains the experiments schema, methodology, presents their results, and analyzes them.

4.1 Experimental Application and Data Schema

The Traffic Ticket solution based on the JSwitch framework is responsible for receiving and processing traffic ticket transactions that arrive in batches. This solution was conceived with the exclusive purpose of testing the JSwitch framework. Basically, the Traffic Ticket solution consists of an implementation of the JSwitch Server Component (JSC) as specified by the framework. Additionally, a custom made solution for the transaction processing layer at the TProc JSC that

contains the complete transaction processing logic for the ticket and ticket payment processing was implemented. Also the required application database for the Traffic Ticket solution was designed and developed. In order to be able to test the Traffic Ticket solution data to address this kind of testing was produced, explained at a later section. Finally, Police and Department of Motor Vehicles (e.g. DTOP) clients were developed to complete the Ticket Traffic testing schema. Detailed explanation about the implementation and functionality of these are next along with a description about the process of data generation to test the Ticket Traffic solution.

The JSwitch implementation for the Traffic Ticket solution consists of the following JSC: a) fixed size of six components for the TColl and TProc JSC collections, b) one TCoord, and c) one TDisp that knows all the components of its corresponding collection. This organization can be seen in Figure 4.1. Additionally, a DB Server was used for persistence purposes. The PostgreSQL 8.1 relational database management system (RDBMS) was used as the data source.

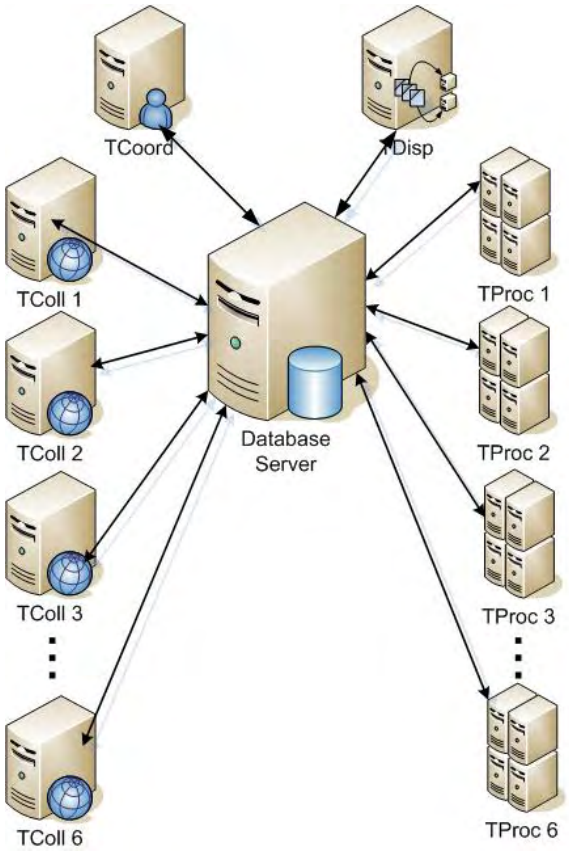

Figure 4.1. Traffic Ticket JSwitch solution architecture

The custom made OLTP solution, named the Traffic Ticket OLTP (TT-OLTP), for the transaction processing layer at the TProc JSC is responsible for traffic ticket and ticket payment transaction processing. This solution has the business logic necessary to accomplish the transaction processing that a batch of one of these types requires. Basically, once the Processor Adapter in charge of converting the batch to the format required by the TT-OLTP solution, the first signals the second for job initiation. After signal acquisition, the TT-OLTP solution starts executing the business logic for each of the transactions for the batch at hand.

The data sets used for the experiments have as its primary unit, traffic ticket or traffic ticket payments. There are two kinds of batches, those that contain only traffic ticket, and those that only have traffic ticket payments. The batches are referred as ticket batch for the ones containing tickets and pay batches for the ones containing ticket payments. In order to test the nature of the JSwitch framework and the performance behavior of its Scheduling Policies the data sets are divided in two categories: homogeneous batches and heterogeneous batches. The transaction totals are 250 transactions for homogeneous batches, and between 150 and 350 transactions for heterogeneous batches.

The data in the traffic transaction batches followed a schema based on made up information about traffic tickets given to motor vehicle drivers and the respective ticket payments. The drivers have probabilities of receiving a ticket from 20 to 70 percent, depending on its driving classification which is selected randomly from the driver population. This percentage determines the amount of tickets each driver has within the the data sets used.

The ER-Diagram for the Traffic Ticket Solution Application Database (TTS ADB) schema is detailed in Figure 4.2. This diagram provides a full description about the data used at the TTS in terms of entities and relationships. The TTS ADB was designed and developed with the purpose of storing all the data about traffic tickets and ticket payments and the results of the business logic of processing these transactions. Basically, the TTS ADB maintains driver, vehicle, police and precinct records of those involved in a traffic ticket issue event. The TTS ADB also stored information about the proper ticket payment adjudication after traffic ticket payment event.

For the JSwitch ADB, the following entities were identified, and their respective attributes are detailed in the ER-Diagram:

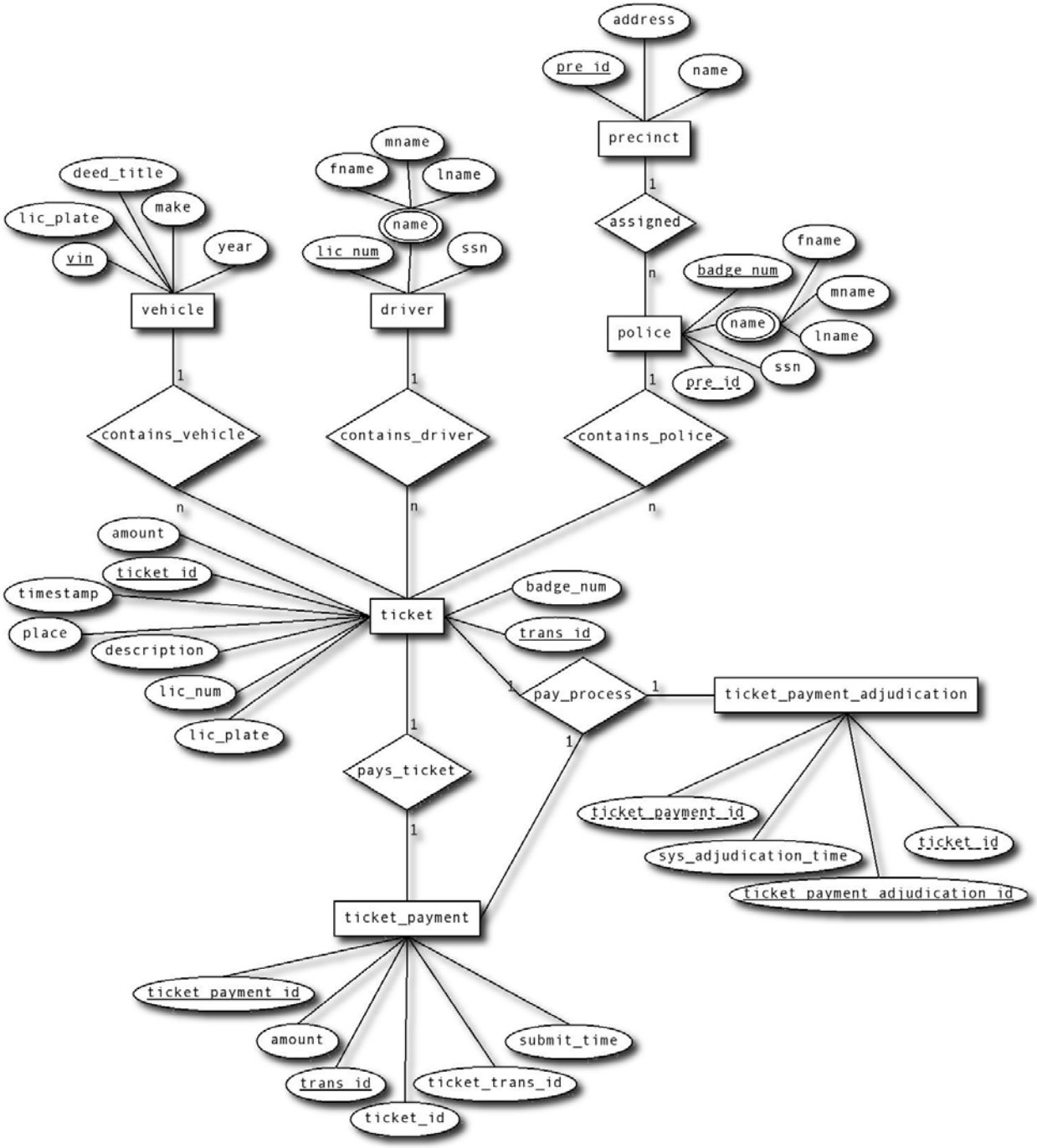

Figure 4.2. Traffic Ticket JSwitch solution application database ER-Diagram

- **precinct** - contains information about a police precinct to which police officers belong to.
- **police** - contains the information about the police officer that has issued traffic tickets.
- **driver** - this entity refers to the person that receives the traffic ticket.
- **vehicle** - refers to the vehicle on which a driver received a traffic ticket.
- **ticket** - the traffic ticket entity contains all the information about a traffic ticket, the police that issues the ticket, the driver that recives the ticket, and the vehicle driven at that moment.
- **ticket_payment** - a traffic ticket payment contains reference information about the ticket that is being payed and the ticket payment date.
- **ticket_payment_adjudication** - refers to the information that results from the processing of a ticket payment including the date on which it was processed.

The following relationships between the entities were also identified and displayed in the Figure 4.2:

- **assigned** - this represents that police officers are assigned to a precinct.
- **contains_police** - it means that a ticket instance contains a police, since this latter is the author of the ticket, also a ticket is issued by only one police officer but many tickets can be issued by a police officer.
- **contains_driver** - this relationship tells us that a ticket contains a driver in the sense that it is a receiver of the ticket. Notice that a ticket belongs to one driver but a driver might receive several tickets.
- **contains_vehicle** - here a vehicle is related to a ticket instance for a specific driver.
- **pays_ticket** - this relationship describes what a ticket payment does in relation to a ticket, there is only one ticket payment for a specific ticket.
- **pay_process** - describes the interaction of events between the ticket issue, the ticket payment submission and the adjudication of a payment to the corresponding ticket.

Basically, this ER-Diagram summarizes the logical events for the Traffic Ticket transaction processing solution as follow. For each ticket, there is an associated police officer, driver and vehicle. Also, for each ticket there is a ticket payment which results in a ticket payment adjudication. The

entities and relationships translate into corresponding tables in the database. The result of this translation is the database Relational Schema which is presented completely in Appendix A.

The functionality of the Police Client is to serve as the means for electronic traffic tickets issued, traffic ticket batch creation and batch submission to the Traffic Ticket Solution based on the JSwitch framework. The DTOP Client offers similar functionalities as the Police client, but instead of dealing with the traffic tickets deals with the payments of the tickets. In addition to the clients already mentioned, a web application that offers driver profile revision without authentication was also implemented. The purpose of this application is to provide a driver profile revision interface for the DTOP administration as well as for any driver that desires to see his or her traffic ticket profile from wherever he wants as long as an internet connection is available. All these clients are Web based applications that use the Apache Jakarta-Struts as its implementation technology. These clients are able to produce well-formed XML documents that contained the transaction batches.

The Traffic Ticket Police client is designed for ticket batch submission. This Web based client application offers the required forms for traffic ticket issuing. On these forms police officers enter their issued traffic tickets during their shifts. Once the first traffic ticket is entered with the client application, an XML document is initiated with the current time and batch sequence and client JSwitch identification key. Additional traffic tickets are added incrementally to this document until one of three conditions are met: 1) the size threshold specified for the XML document is reached, 2) the time threshold specified for batch submittal is due, or 3) the user simply decides to submit the batch. At batch submission time, the Web based application calls the authentication() web service method at the TCoord to start the Batch Transaction Processing Protocol already detailed.

The DTOP Client is responsible for the traffic ticket payments instead of the traffic tickets registry. Thus, it has the same logical batch creation and submission process. Additionally, the DTOP client can be used to view any driver traffic ticket profile.

These two traffic ticket clients were developed in order to have an interface from which to create and send complete traffic ticket batches or ticket payment batches. But in addition, to stress-test the system we developed a complete test client infrastructure . Basically, the test client is a Java application that contains the methods needed to call the Web service methods

of authentication() and requestTCollIP() on the TCoord system entry point. This enables the test client to submit previously generated batches full with transactions to the system and collect measurements from the resulting system behavior. Furthermore, a management application was developed to further automate the tests. This management application is a Java application that is responsible for managing the JSC components; it can start, reset or shutdown any JSC component.

4.2 Experimental Scenarios and Methodology

The previous experimental setup, based on the JSwitch architecture, along with the workload used to conduct the experiments and its stress-test client are used on a series of testing scenarios. The coordination of transaction batches was determined as the most important task of the framework and the one expected to be performed more frequently by a JSwitch based solution. Therefore, the experimental analysis of the framework is focused around the performance of the Transaction Coordination Protocol involved in transaction batch coordination. The performance metric to be measured is *throughput*, in terms of transactions batches coordinated per unit of time. The experimental scenarios are classified based on the number of concurrent clients, on the scheduling policy in use for batch assignment, and on wether the batches have an homogeneous or heterogeneous number of transactions.

In all the scenarios, the JSwitch DB Server and the application database (ADB) resided on the same computer. The same is true for the Traffic Ticket JSwitch OLTP (TT-OLTP)solution ADB. The computer used for these purposes contained dual Intel Xeon processors at 3.0GHz with 64-bit extensions, hyper-threading support, and 1MB of Level 2 cache each, 2048MB of RAM, and 3 SCSI drives in a RAID 5 array running the 64-bit edition of SuSE Linux 9.2. The clients were all independent computers running SuSE Linux 10.0. All communications between these computers were transmitted through 100 Mbps Ethernet connections and the computers were in the same LAN. As mentioned in Section 3.2, all the components in our implementation of the JSwitch framework were developed using the Java [31] programming language. For the Web services implementation, Apache Axis2 [2] was used. The Web server and servlet container used was Apache Tomcat [3]. Both the JSwitch ADB and the TT-OLTP ADB were implemented using PostgreSQL [22] databases. For a more detailed description of the computer equipment and the applications used for the JSwitch

implementation see Appendix A.

The Experimental Methodology is detailed here, showing the steps and logic followed to perform the test of the prototype JSwitch solution. There where two kinds of jobs submitted to the system: ticket batches and ticket payment batches. In the homogeneous data set, each batch contained 250 transactions, worth 290KB and 100KB of data, respectively. In the heterogeneous data set the batches contained from 150 transactions up to 350 transactions, worth between 175KB and 400KB and between 60 KB to 140KB respectively. The tests were conducted using 1, 4, 8 and 16 clients that uniformly submit batches for transaction totals of 16, 32 and 64 thousands that corresponds to workloads of 12.19MB, 24.38MB and 48.75MB, respectively, over a range of 1 to 3 minute periods. This time periods depend on the number of batches and number of clients.

The first scenario is where 1 client submit batches. In this scenario all the batches reside at the same client, which makes the batch submit period the longest. On the other hand the scenario where 16 clients submit their batches posses the shortest batch submit period. Each scenario was tested using each of the Scheduling Policies. The scheduling policies involving some type of load statistic were developed and re-developed to obtain a good outcome.

In order to acquire these measures JDBC insert statements were executed at each of the testing components. Starting with the clients, the JSC components and finally the custom made OLTP solution. The JDBC insert statements log at the DB Server at a separate application database for the exclusive purpose of obtaining these measures. Once the system has finished processing a data set makes a database dump which is a backup of the ADB current state and the experiment result generation. These results are in the form of Excel spreadsheets which significantly aids in the readability and later report creation and graphic generation of the experiment results.

The experiment methodology is described next step by step:

1. The first step is the application database (ADB) clear and reload with the respective schemas already described for the JSwtich ADB and TT-OLTP ADB.
2. The TCoord and TDisp JSwitch Server Components (JSC) are set with the Scheduling Police corresponding to the test scenario.
3. All the JSC components are started.

4. Once the JSwitch system components are started, the client experiment scripts are then started, which triggers the transaction batch submission to the JSwitch system. This starts by calling the authenticate() Web service method at the TCoord JSC.
5. The tests end after the client or clients have send all their transaction batches to the JSwitch system and this latter finishes their proper processing.
6. At this point the automated testing application detects the system has finished processing all the batches and starts the process of ADB backup and experiment results generation. The experiment results are generated in Excel spreadsheets from where tables and plots are generated.
7. After these six steps an experiment iteration for a scenario is completed. There are a total of three repetitions per scenario.
8. The experimental results at each repetition per scenario are averaged to obtain the results for a possible real case scenario.
9. The percentage of time at each JSwitch architecture tier and JSC are also computed.
10. Finally, the average throughput (in transactions and batches per minute) is computed for each of the testing scenarios.

4.3 Performance and Load Analysis

This section gives an in detail analysis about the series of experiments that were carried out to analyze the performance of the system. The system is analyzed in terms of throughput and load capacity by scheduling policy. Throughput is defined here as the division between the number of transactions processed and the processing time for a given policy. Throughput evaluation was performed for each of the 4 scheduling policies one with two variations, later described and the two data sets as explained above with the experimental setup already detailed. The results of the experiments explained above are discussed in the next sections. These next sections contain the average results of the three repetitions performed.

Table 4.1. RRS operation times for the Batch Coordination Protocol (Homogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	5.452	2935	9.055	3534	17.050	3754
4	5.466	2927	9.424	3396	16.724	3827
8	4.912	3257	9.112	3512	17.388	3681
16	4.833	3310	9.255	3458	17.482	3661

Table 4.2. RRS operation times for the Batch Coordination Protocol (Heterogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	5.364	2983	8.769	3649	17.602	3636
4	4.972	3218	9.687	3303	16.687	3835
8	5.231	3059	9.389	3408	17.675	3621
16	4.854	3296	9.034	3542	17.263	3707

4.3.1 Coordination Protocol with Scheduling Policy 1 Analysis

Scheduling Policy 1 is Round Robin Scheduling (RRS). Basically, this policy's primary objective is to achieve uniform work distribution among system components since it is based on the Round-Robin algorithm. Tables 4.1 and 4.2 show the resulting operation times for transaction batch coordination and processing for the Ticket Traffic JSwitch solution for the the homogeneous and heterogeneous data sets and divided by number of clients respectively.

The throughput measured in transactions per minute for the homogeneous batches increases as the total number of transactions increases. Additionally, the throughput tends to increase as the number of clients increases. This is also true for the heterogenous batches. This scheduling policy threats each JSC as being equally capable of executing their work. Which is true, when only homogeneous batches are used. In the case of heterogeneous batches each JSC is not equally capable of executing their work since the workloads are not the same in terms of quantity of transactions per batch. Additionally, if the batches are assigned in a Round-Robin fashion there is a probability of some components ending up with many batches with large workloads and a few components with the smaller workloads. Therefore, resulting in an undesired work distribution since the system

Table 4.3. RS operation times for the Batch Coordination Protocol (Homogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	4.880	3279	9.496	3370	17.059	3752
4	4.954	3230	9.126	3506	17.603	3636
8	5.154	3104	9.141	3501	17.357	3687
16	4.925	3249	9.331	3430	17.719	3612

Table 4.4. RS operation times for the Batch Coordination Protocol (Heterogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	4.957	3228	8.817	3629	17.006	3763
4	5.304	3017	9.406	3402	16.160	3960
8	5.556	2880	9.147	3499	15.955	4011
16	5.297	3020	10.052	3184	16.613	3853

resources are not used efficiently. This fact must be taken in consideration when implementing a JSwitch solution since this scheduling policy simply provides equal utilization of system resources.

4.3.2 Coordination Protocol with Scheduling Policy 2 Analysis

Scheduling Policy 2 is Random Scheduling (RS). This scheduling policy offers complete utilization of system resources along with the probability to avoid hot spots in the presence of heterogeneous workloads here batches for example since it distributes them randomly. Tables 4.3 and 4.4 shows the resulting operation times for transaction batch coordination and processing at the JSwitch solution for the the homogeneous and heterogeneous data sets and divided by number of clients respectively. The probability of hot spots is a desired attribute of this scheduling policy. Tables 4.3 clearly shows that the throughput for the homogeneous batches increases as the total number of transactions increases. This is also true for the heterogenous batches.

It is a good match for a JSwitch solution when full utilization of system resources along with unorder distribution is desired.

Table 4.5. LLS using 5 min refresh rate operation times for the Batch Coordination Protocol (Homogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	19.069	839	41.465	772	86.794	737
4	20.372	785	41.268	775	86.675	738
8	20.666	774	42.381	755	84.328	759

4.3.3 Coordination Protocol with Scheduling Policy 3 Analysis

Scheduling Policy 2 is Least Loaded (LLS). This scheduling policy as described earlier is driven by a priority function that describes the data rate potentially achievable at the present moment at a particular system component. Basically, LLS selects the best available capable component for the task at hand. Simply, this scheduling policy will perform as well as its selective criteria determines its. In this case the selective criteria is the priority function. The selective criteria is in terms of describing the component load, this is workload and system load.

Initially, the LLS policy uses as its selective criteria the load average at the last 5 minutes as given by the Linux kernel. The load average is a UNIX computing term defined by [33] as the measure of the number of active processes at any time, a measure of CPU utilization. Also [9] defines load average as the sum of the run queue length and the number of jobs currently running on the CPUs. This metric was refreshed every 5 minutes and then every 30 seconds seeking improved results since it does not produced the desired and expected results for the LLS policy. Even with a refresh time of 30 seconds this policy does not perform as expected. Further analysis signaled work load as an alternative selective criteria and as the results show workl load resulted to be a far better criteria. Tables 4.5 shows the results obtianed for 1, 4, and 8 clients using the 5 minutes refresh rate for the LLS scheduling policy.

The analysis of the selective criteria for the LLS policy concluded that the work load combined with the load average is the best alternative to produce efficient system resource utilization. This is, workload is the selective criteria primary used along with a threshlod signaling system overload determined by a load average higher than 3.0. Once this selective criteria was developed it was used for the complete testing of the system.

Table 4.6. LLS operation times for the Batch Coordination Protocol (Homogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	5.071	3155	8.678	3687	15.962	4009
4	4.602	3476	8.750	3657	15.980	4005
8	4.744	3372	8.469	3779	16.148	3963
16	4.790	3340	8.207	3899	16.030	3993

Table 4.7. LLS operation times for the Batch Coordination Protocol (Heterogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	5.025	3184	8.577	3731	16.361	3912
4	4.714	3394	9.162	3493	15.985	4004
8	4.704	3401	8.500	3765	15.604	4102
16	5.240	3054	8.218	3894	15.987	4003

Tables 4.6 and 4.7 show the resulting operations times for the transaction batch coordination and processing at the Traffic Ticket JSwitch solution for the the homogeneous and heterogeneous data sets and classified by number of clients. This scheduling policy as shown by the results provides good performance under homogeneous or heterogeneous data sets as expected.

4.3.4 Coordination Protocol with Scheduling Policy 4 Analysis

Scheduling Policy 4 is Random Towards Least Loaded Scheduling (RTLLS). This scheduling policy offers the benefits of the last two scheduling policies. The probability to avoid hot spots since it distributes them randomly among those already selected by the least loaded selective criteria. Two variations of these scheduling policy were implemented and tested. These are random selection among half of the components (RTLLS-Half) and among the third of the components (RTLLS-Third). In this case since there are six TColls and six TProcs random among three and two components respectively. This variation will help understand how the system benefits the most, with the random logic or the use of a metric. Tables 4.8 and 4.9 show the resulting operations times for the transaction batch coordination and processing at the Traffic Ticket JSwitch solution for the the homogeneous and heterogeneous data sets and divided by number of clients for the RTLLS-

Table 4.8. RTLLS-Half operation times for the Batch Coordination Protocol (Homogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	4.669	3427	8.670	3691	15.888	4028
4	4.834	3310	8.194	3905	15.968	4008
8	4.576	3497	8.120	3941	15.992	4002
16	4.655	3437	8.139	3932	15.500	4129

Table 4.9. RTLLS-Half operation times for the Batch Coordination Protocol (Heterogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	4.814	3324	8.338	3838	15.893	4027
4	4.573	3499	8.064	3968	15.773	4057
8	4.885	3276	8.396	3811	16.087	3978
16	4.949	3233	7.985	4008	15.529	4121

Half variant. Tables 4.10 and 4.11 contains the same information but for the RTLLS-Third variant. This scheduling policy is expected to outperform the others since it was designed by taking the advantages of ths last two policies. Throughout these four figures the RTLLS-Half scheduling policy outperforms its counterpart. This leads to the observation that the use of more randomization in this hybrid improves its performance.

Table 4.10. RTLLS-Third operation times for the Batch Coordination Protocol (Homogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	4.715	3393	8.415	3803	15.875	4031
4	4.699	3405	8.212	3897	16.199	3951
8	4.546	3520	8.272	3868	16.075	3981
16	4.602	3477	8.316	3848	15.971	4007

Table 4.11. RTLLS-Third operation times for the Batch Coordination Protocol (Heterogeneous Data Set)

Clients	Number of Transactions					
	16,000		32,000		64,000	
	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)	Response Time (min)	Throughput (trans/min)
1	4.714	3394	8.349	3833	16.334	3918
4	4.645	3444	8.179	3912	15.827	4044
8	4.537	3527	8.306	3853	15.660	4087
16	4.641	3448	8.865	3610	16.071	3982

4.3.5 Scheduling Policy Throughput Comparison

This section analyzes the throughput results for the different scheduling policies. The figures on this section displays the already presented experiment results in order to explain the observations findings based of these results. The figures are classified by the experiment scenarios. Basically the averages of the three experiment repetitions are displayed here classified by each of the four scheduling policies and its variations.

Figures 4.3 and 4.4 display the throughput for the first experiment scenario that contained only one client. This scenario represents complete system resources availability for the client. In this scenario the RTLLS-Third(RLLT) scheduling policy presents the best policy throughput for the system, specially as the transaction load increases. This policy is closely followed by RTLLS-Half variant. This figure shows that the random-towards-least-loaded type of policies provides the best tradeoff for throughput.

Similarly figures 4.5 and 4.6 display the throughput but in this occassion for the second experiment scenario that contained four clients. This scenario is the first of three experiment scenarios where system resources are distributed and even shared by clients. At this scenario, as the figures show, the scheduling performance was very similar between both of the data sets. This time the RTLLS-Half(RLLH) scheduling policy presented better throughput at the system, specially as the transaction load increases. This policy changed places with its Third variant when compared with the first scenario. Again its clear that some policies are in fact more efficient that others and the observation that the metric based scheduling policies perform better that its non-metric based counterparts was maintained.

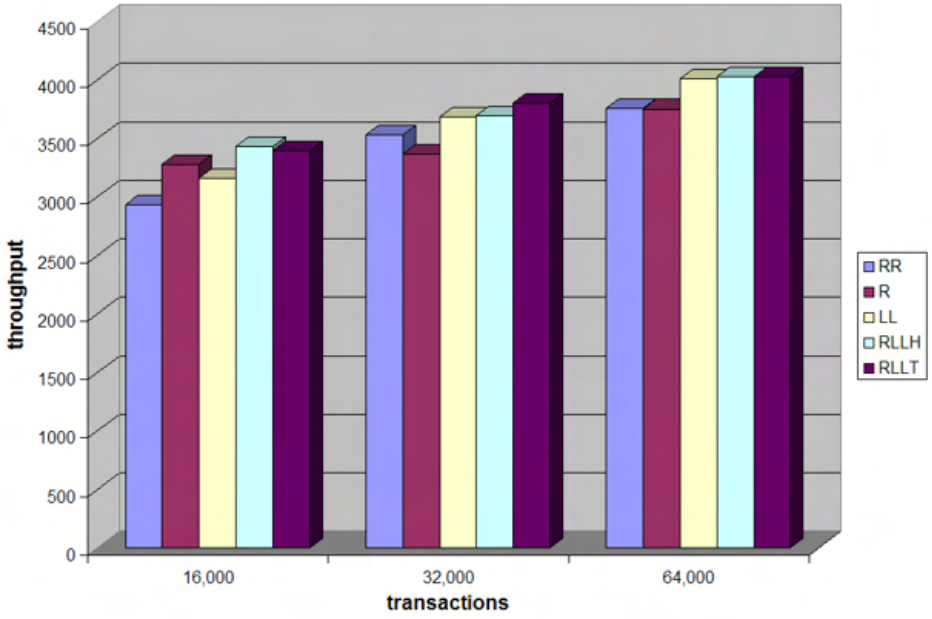

Figure 4.3. Throughput comparison among implemented scheduling policies with 1 client for the homogeneous data set

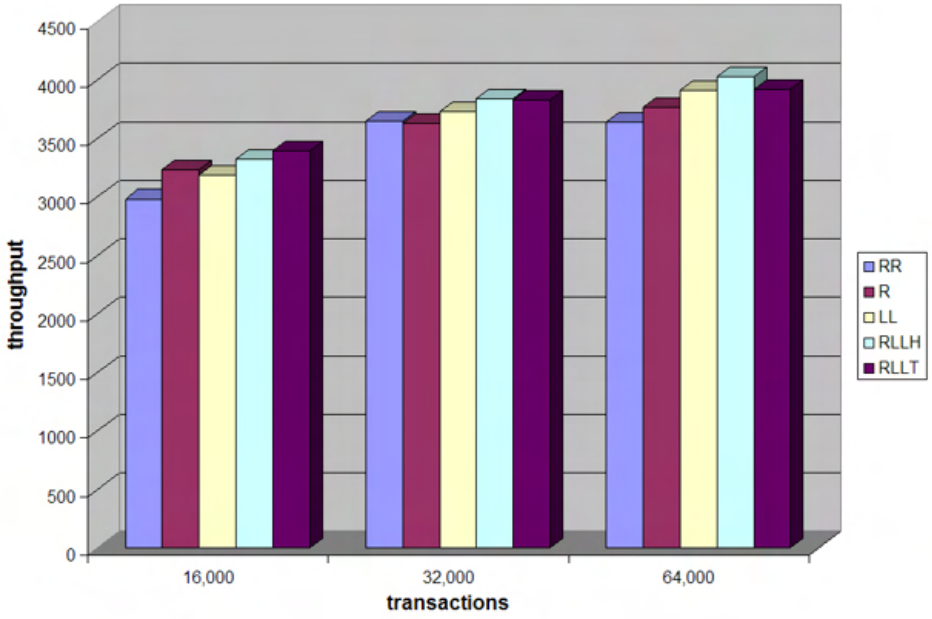

Figure 4.4. Throughput comparison among implemented scheduling policies with 1 client for the heterogeneous data set

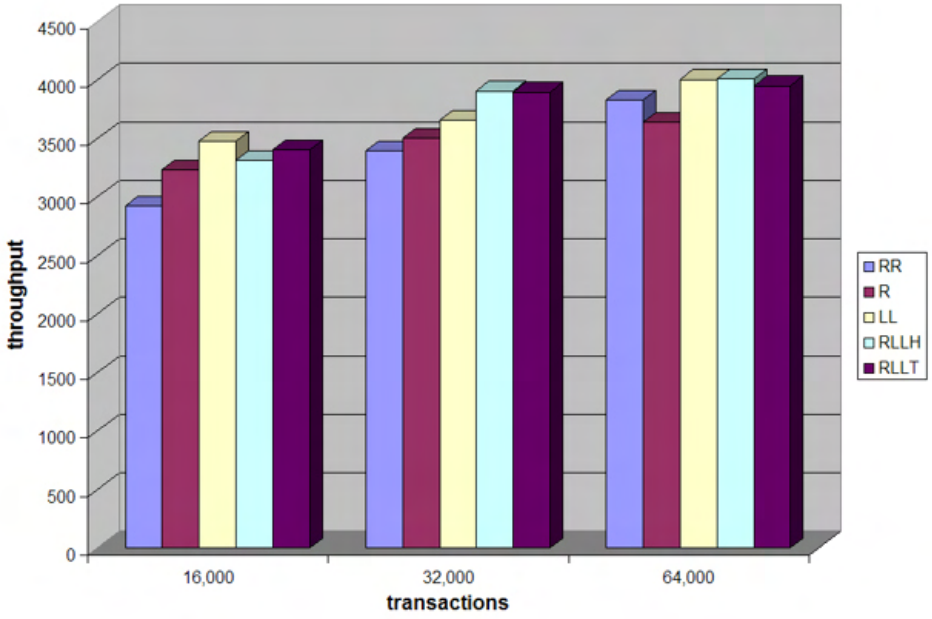

Figure 4.5. Throughput comparison among implemented scheduling policies with 4 client for the homogeneous data set

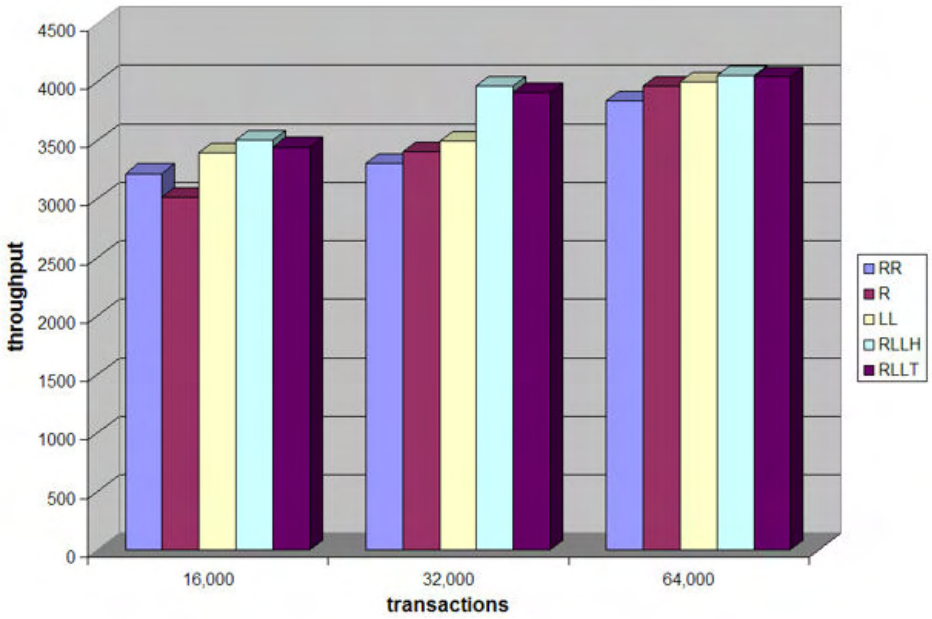

Figure 4.6. Throughput comparison among implemented scheduling policies with 4 client for the heterogeneous data set

Figures 4.7 and 4.8 were obtained at the third experiment scenario which was performed using eight clients. This represents evidence that for a second time the RTLLS-Half(RLLH) scheduling policy presents better throughput at the system, specially as the transaction load increases. The results were similar to scenario number 2. Also, scheduling policie's performance where clearly different. The observation that the metric based scheduling policies perform better than its non-metric based counterparts was also maintained here.

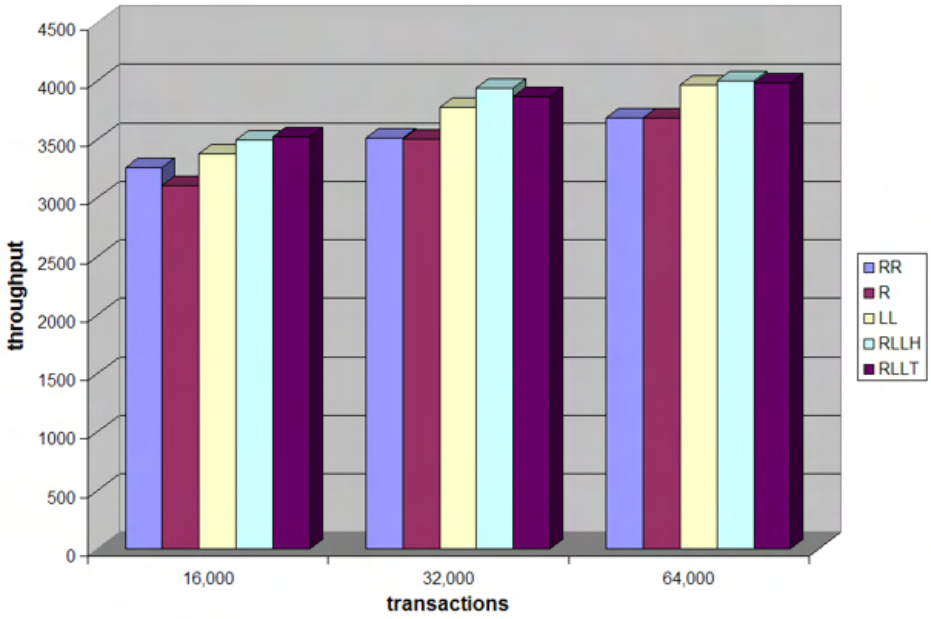

Figure 4.7. Throughput comparison among implemented scheduling policies with 8 client for the homogeneous data set

Finally, the last scenario shown in figures 4.9 and 4.10 show the throughput results for a system used by 16 clients. This scenario tests the system with the largest number of clients which represents the biggest challenge of the scenarios in term of concurrency and system resources availability under stress circumstance. Remember that all the clients send their batches during the same period. The throughput comparison between policies evidences that the RTLLS-Half(RLLH) scheduling policy presents better throughput at the system, specially as the transaction load increases. Here in terms of number of clients and number of transactions also. This time the RRS also made a statement over its no-metric based scheduling policy counterpart of RS outperforming the latter. These observation are maintained for both data sets.

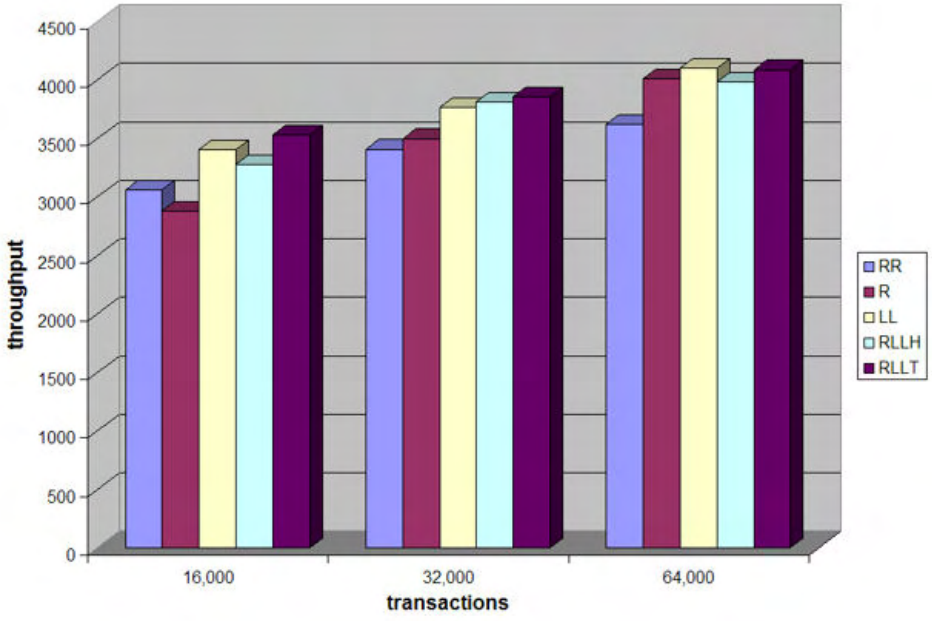

Figure 4.8. Throughput comparison among implemented scheduling policies with 8 client for the heterogeneous data set

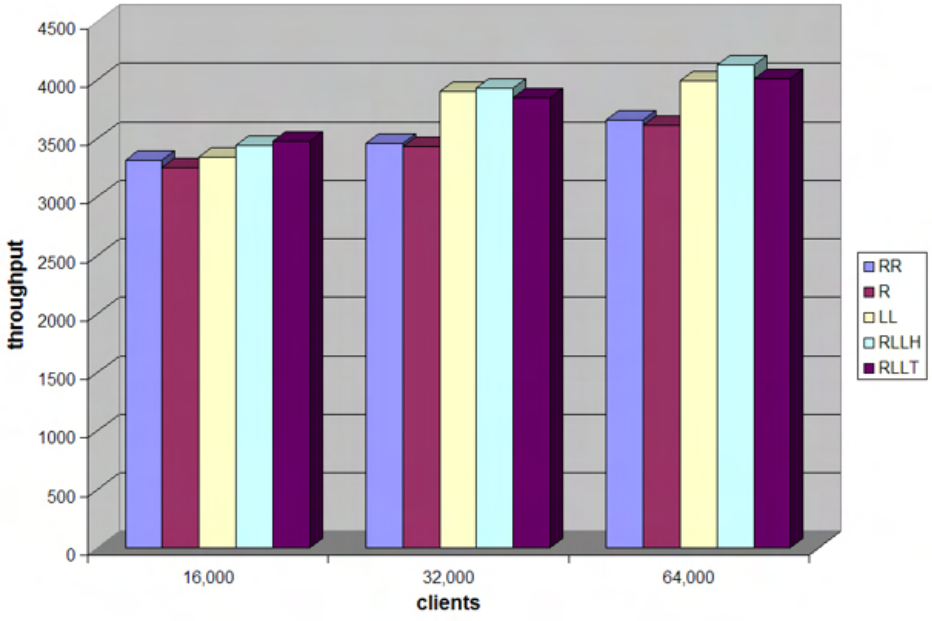

Figure 4.9. Throughput comparison among implemented scheduling policies with 16 client for the homogeneous data set

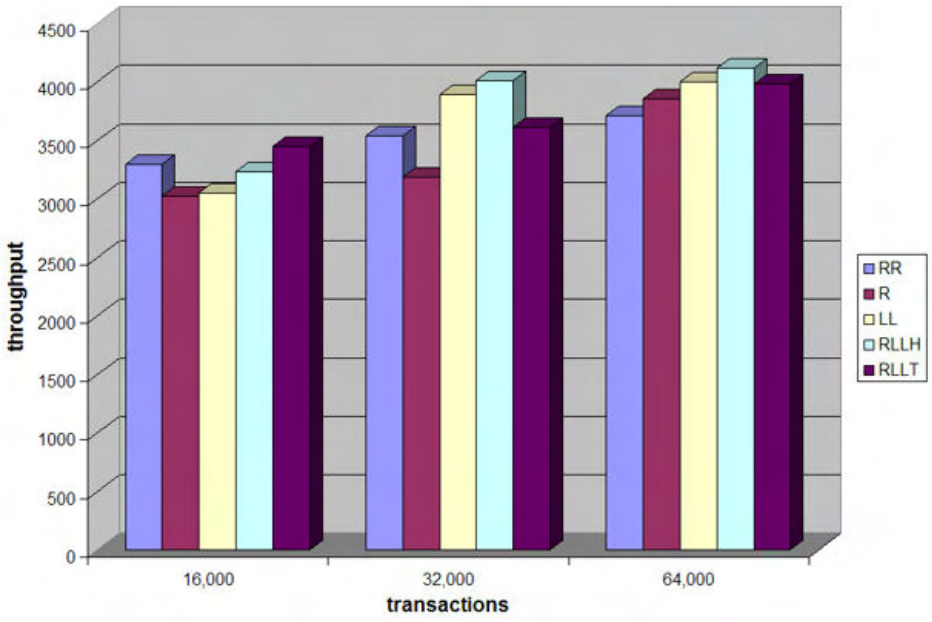

Figure 4.10. Throughput comparison among implemented scheduling policies with 16 client for the heterogeneous data set

These figures have displayed that some policies are in fact more efficient that others. Basically, the metric-oriented policies of LLS, RTLLS-Half and RTLLS-Third have an advantage over the non-metric oriented ones of RRS and RS. The selective criteria designed for the JSwitch solution that these metric-oriented policies use have been proven better that the results produced by the non-metric oriented. These findings are maintained over the homogeneous and heterogeneous data sets. Additionally, among the metric-oriented policies, the RTLLS hybrid has greater efficiency than pure LLS. This result is due to the fact that LLS even though its selects the best possible component at the moment it can still overload components and RTLLS helps reducing this effect. The difference among the RTLLS-Half and RTLLS-Third policies is that in fact the randomization used among the least loaded components improves the the quality of the selection.

4.3.6 Load Capacity Analysis

This analysis refers to the systems' ability to deal with bigger number of transactions since as the number of transactions increases the rate at which the system is used increases considerably. This is directly related to the number of clients the system attends concurrently, but can also be affected by the quantity of transactions at the batches. This is as the level of concurrency increases

deadlock possibility increases exponentially as stated by Jim Gray at The Transaction Concept Virtues and Limitations [23]. Therefore, increases in the factors directly related to the number of transactions at the system reduces the following dimensions: the ability of the system to coordinate and process transactions, the response time and consequently throughput.

Figures 4.11 and 4.12 displays the throughput of the tests for the scheduling policies for systems workload of 16,000 transactions for 1, 4, 8, and 16 clients. The results show that as the number of clients increase, the system throughput is slightly increased or at least maintained. The system throughput for the scheduling policies when attending 16,000 transactions is similar. This behavior observed at both data sets.

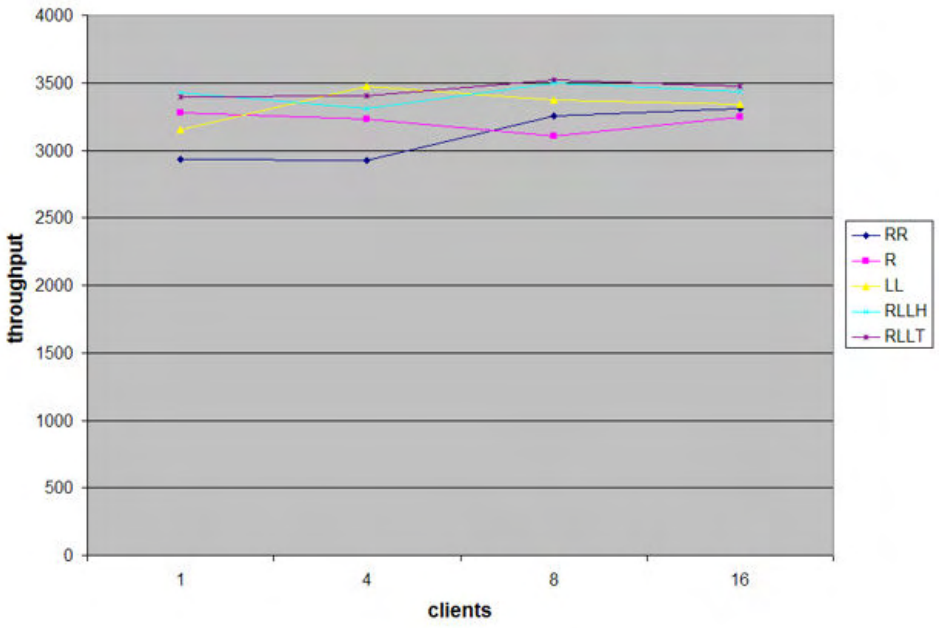

Figure 4.11. Throughput for load capacity of 16,000 transactions for the homogeneous data set

The scheduling policies for a system with a workload of 32,000 transactions behave differently in terms performance. The metric based scheduling policies present better performance than the one displayed by its non-metric based counterparts, specially as the system workload increases. This can be seen in figures 4.13 and 4.14. This difference in performance is seen for the two data sets.

Finally, the scheduling policy performance for the the largest load used to test the JSwitch

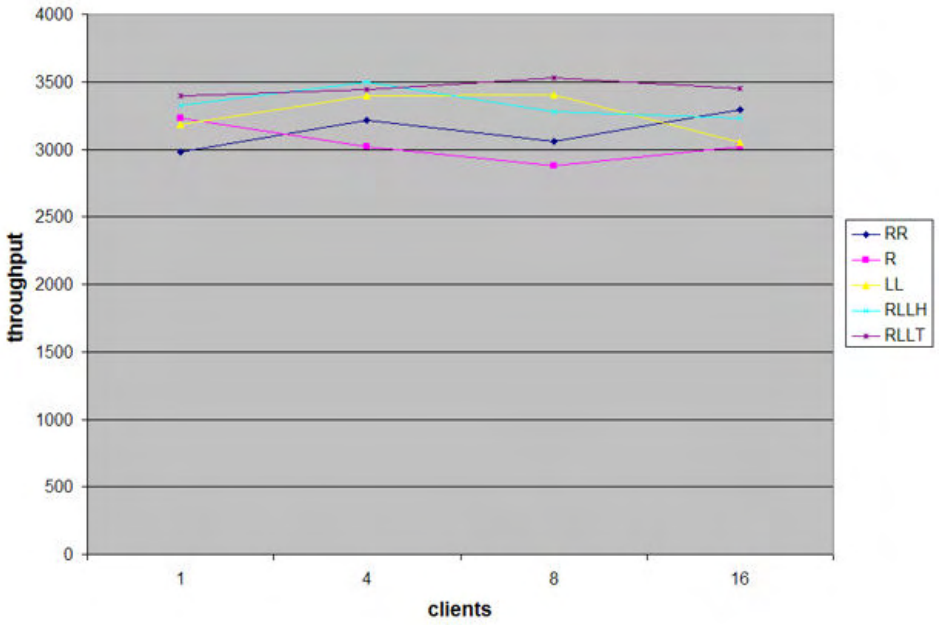

Figure 4.12. Throughput for load capacity of 16,000 transactions for the heterogeneous data set

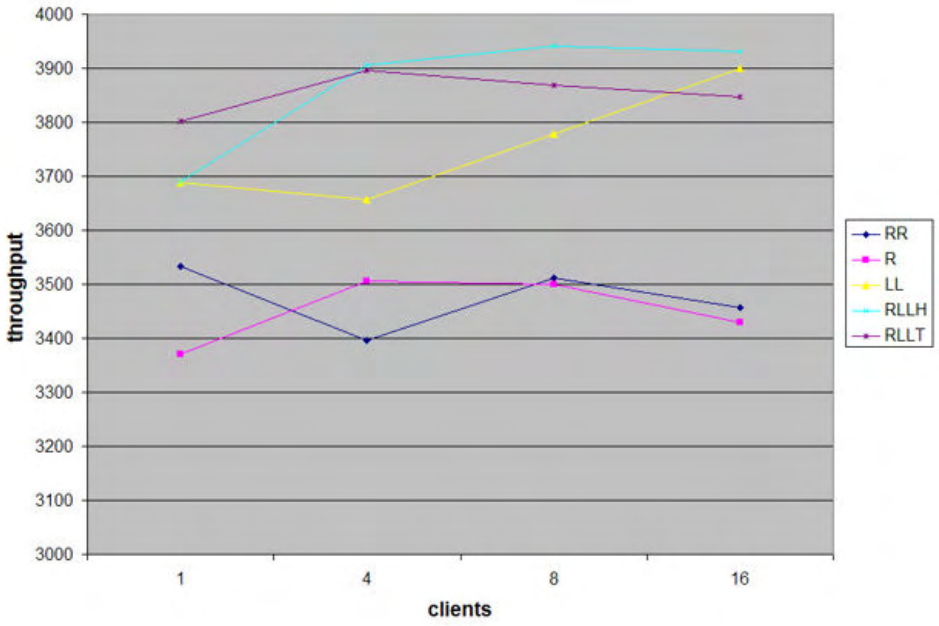

Figure 4.13. Throughput for load capacity of 32,000 transactions for the homogeneous data set

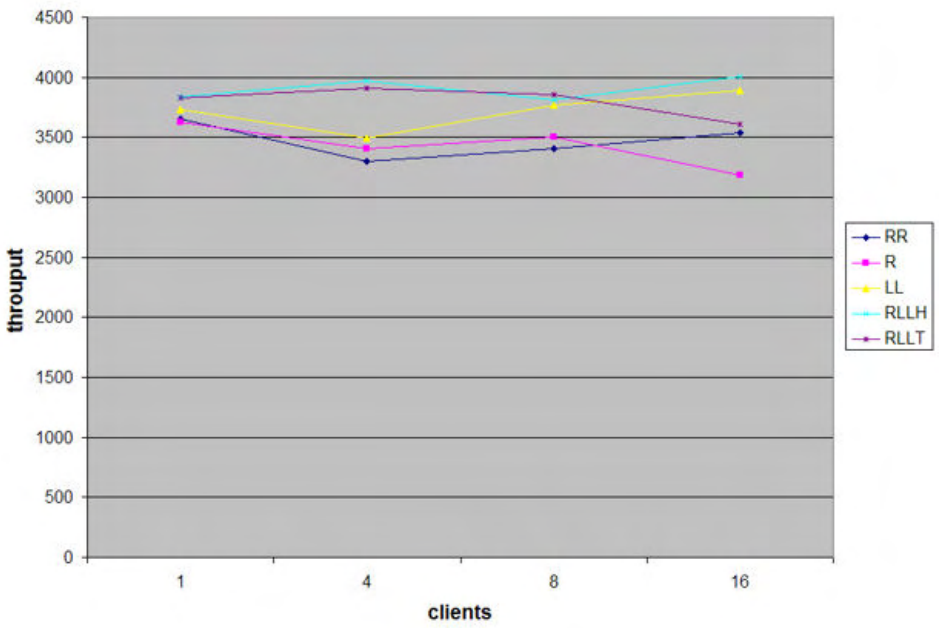

Figure 4.14. Throughput for load capacity of 32,000 transactions for the heterogeneous data set

solution which corresponds to 64,000 transactions is displayed in Figures 4.15 and 4.16. Again the non-metric based scheduling policies were outperformed by their conterparts. This is seen for both of the data sets.

The results displayed at these figures show that as the number of transactions attended by the system increase the system throughput also increases. These is also observed as the number of clients seen by the system increases. These observations make us think about what is the system saturation point at which the throughput reaches a steady state or even worst lowers. This point is mentioned at the next section as a possible future work to further study the nature of the system.

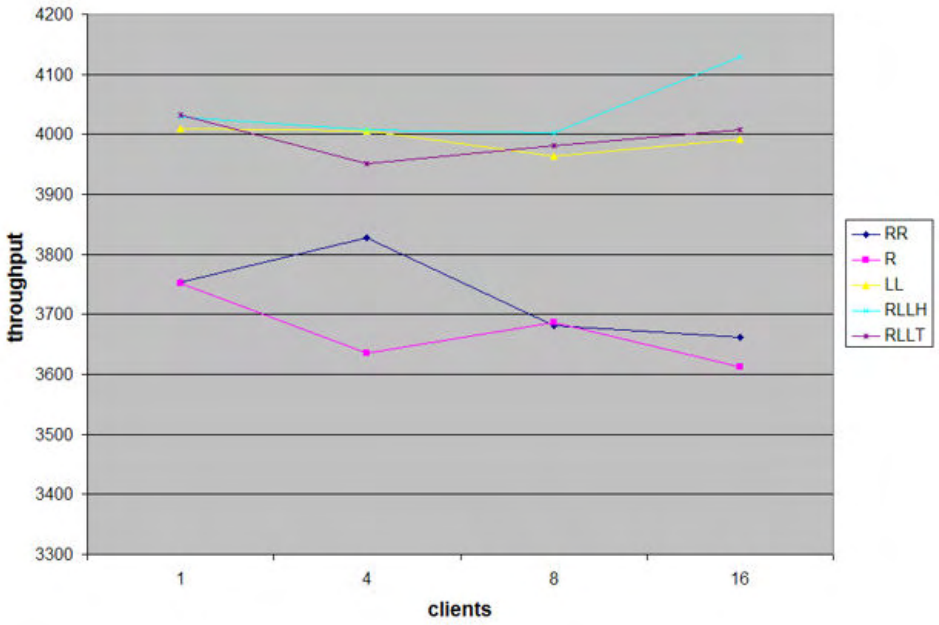

Figure 4.15. Throughput for load capacity of 64,000 transactions for the homogeneous data set

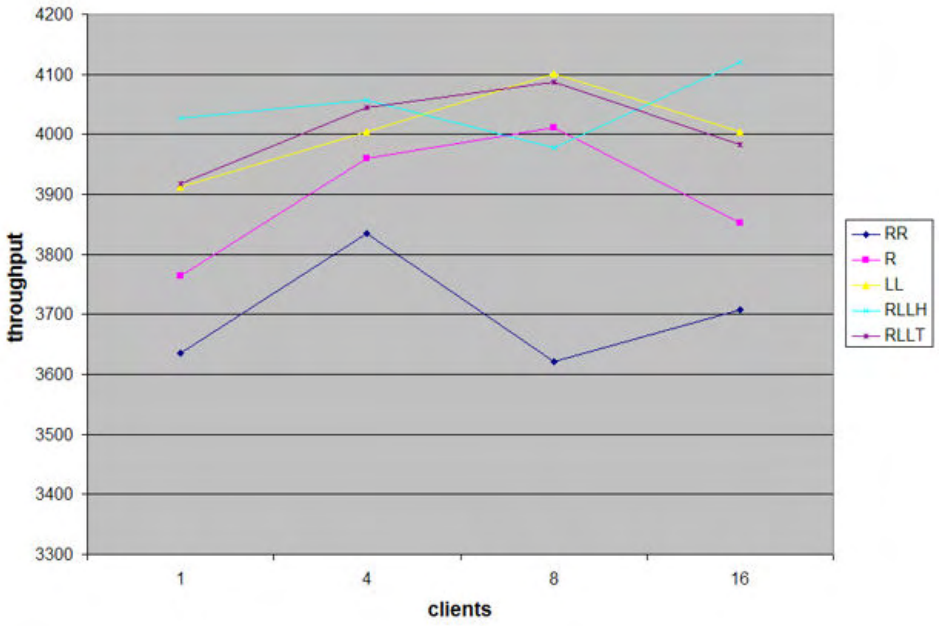

Figure 4.16. Throughput for load capacity of 64,000 transactions for the heterogeneous data set

CHAPTER 5

Conclusions and Future Work

This thesis has presented a Framework for a Web-Based Transaction Coordinator Switch called JSwitch. The JSwitch framework is a Web-based transactional coordination system designed to accept batches of related transactions in Web-based environment and forward them to the appropriate transactional server application. Current on-line transaction processing system can be employed here in order to handle each individual transaction. Additionally, this framework can cope with the scalability load problems that Web environments present by monitoring and recording statistics of its system resources.

The JSwitch framework presents an alternative for transaction batch coordination specifically designed to cope with the Web environment challenges of heterogeneity and scalability. These challenges are the current eCommerce barriers that must be overcomed in order to unleash the full potential of this business technology. The external interface of a JSwitch framework solution is completely based on Web services. Therefore, it naturally achieves the necessary communication cross-platform capabilities to address heterogeneity issues. Furthermore, the JSwitch framework is presented as a means to either scale current batch transaction processing systems or develop new ones based on the framework. A JSwitch based solution will posses specially designed load balancing and monitoring mechanisms to deal with the Web traffic, and deal with unexpected spikes in system load.

5.1 Summary of Contributions

We have discussed how the JSwitch framework can be used to implement solutions that permit transactions to be exchanged seamlessly in intra-agency or inter-agency environments. Basically, Web services standards and the use of eXtensible Markup Language (XML) technology were used as the means for this ease of interaction among transaction parties. We have also discussed how JSwitch can be used by a single provider of services as a tool to balance the load among various servers used to manage transactions. Service providers could integrate the JSwitch framework to their systems and obtain the desired system load balancing immediately. These servers might be located in a single site, or distributed geographically and accessible by means of a corporate intranet.

We have presented the initial implementation of the system, along with a performance study in terms of throughput and load capacity. Throughput in terms of the performance that offers the different scheduling policies when used to coordinate transaction batches at the Batch Transaction Coordination Protocol. This was developed as the communication mechanism among the JSwitch components. We also presented a performance study which discussed the tradeoff between the different load balancing policies used in the system to assign the processing of transactional batches. These policies are a) Round Robing Scheduling, b) Random Scheduling, c) Least Loaded Scheduling, and d) Random Towards Least Loaded Scheduling. Our performance study shows that the latter (Random Towards Least Loaded Scheduling) provides the best performance for JSwitch.

The JSwitch server components offer the hot-plugable ability, making the framework flexible in terms of administration and agile when systems resources are needed the most. This feature in addition to the effective load balancing statistical monitoring mechanism are part the JSwitch scalability features. Furthermore, component (TColl and TProc) crash-recovery and work re-distribution and re-dispatchment are part of the framework.

The JSwitch framework provides many features that make it a unique solution for the coordination of batches of transactions over the Web. The current JSwitch reference implementation is the first effort towards this Web based batch transaction coordinator load balancing framework. There is actual room for improvements that future work should address in order to become a feature

rich and robust coordination utility.

5.2 Future Work

This section signals future work concerning the JSwitch framework:

- Implementation of a JSwitch Report Utility to further improve its current features and offer an administrative aid when deploying and when fully operating a JSwitch solution. In addition we will be interested in developing a report utilities function for transaction processing systems. Having a JSwitch report utility will become essential when mass batch processing is executed since it will be needed for organization purposes. These reports could include metadata and statistics about the JSwitch solution.
- A JSwitch Management Utility to completely control the JSwitch components and monitor its operations. Such a management tool could benefit adopters since it will provide the facilities to administer the system without having to necessarily know its internal structure.
- There is currently a lot of room for improvement in terms of the security of the JSwitch. The security measures could start by developing a security layer for the database and follow with robust authentication mechanism into the system by clients.

REFERENCES

[1] Apache Software Foundation, Apache Axis2, www.ws.apache.org/axis2 December 2006.

[2] Apache Software Foundation, Apache Tomcat, www.tomcat.apache.org December 2006.

[3] Bernstein, P., Newcomer, E., Principles of Transaction Processing For The Systems Professional, 1st. Edition, Morgan Kaufmann, 1997.

[4] Bhatti, N., Bouch, A., Kuchinsky, A., A Integrating user-perceived quality into Web server design Proceedings of the 9th International World Wide Web Conference (WWW9), May 2000.

[5] Business Transactio Protocol OASIS, December 2006; www.oasis-open.org/committees/business-transactions

[6] Chen, P.P., *The Entity-Relationship Model - Toward a Unified View of Data*, ACM Transactions on Database Systems (TODS), vol. 1, no. 1, pp. 9-36, March 1976.

[7] Ching, A. Building a Highly Available Database Cluster, Microsoft Press, 2000.

[8] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., *Web Services Description Language (WSDL) Version 1.1*, W3C Note, March 2001; www.w3.org/TR/wsdl.html.

[9] Cockcroft, A., Sun Performance and Tuning SunSoft Press, 1995.

[10] Codd, E.F., *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, vol. 13, no. 6, pp. 377-387, June 1970.

[11] Common Criteria Implementation Board, *Common Criteria for Information Security Evaluation*, Version 2.1, 1999; ww.csrc.nist.gov/cc.

[12] Dalal, S., Little, M., Potts, M., Webber, J. Coordinating Business Transactions, IEEE Computer Society, pages 30, 39, February 2003.

[13] Dar, S., Hecht, G., Schochat, E., dbSwitch - Towards a Database Utility, SIGMOD ACM, 2004.

[14] Dayal, U., Hsu, M., Ladin, R. Business Process Coordination: State of the Art, Trends, and Open Issues, Proceedings VLDB, 2001.

[15] West, D. Global E-Government, 2005, September 2005.

[16] eXtensible Markup language (XML) specification, www.w3.org/XML/ December 2006.

[17] Freeman, E. Head First Design Patterns, OReilly, 2001.

[18] Freier, A., Karlton, P., and Kocher, P., *The SSL Protocol Version 3.0*, Internet-Draft, November 1996; www.wp.netscape.com/eng/ssl3/draft302.txt.

[19] Geer, D., *Taking Steps to Secure Web Services*, Computer, vol. 36, no. 10, pp. 14-16, October 2003.

[20] Graham, S., Davis, D., Simeonov, S., Building Web Services with Java, 2nd. Edition, Sams Publishing, 2005.

[21] Gray, J., *The Next Database Revolution*, ACM SIGMOD, pp. 1-4, June 2004.

[22] Gray, J., Reuter, A., Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993.

[23] Gray, J. The Transaction Concept: Virtues and Limitations, Proceedings VLDB, pages 1,23, 1981.

[24] He, J., and, Wang, M., *Cryptography and Relational Database Management Systems*, 2001 International Symposium on Database Engineering & Applications, pp. 273-284, July 2001.

[25] JDBC Technology, www.java.sun.com/products/jdbc/ December 2006.

[26] Limthanmaphon, B., Zhang, Y., Web Service Composition Transaction Management, Proceedings of the fifteenth conference on Autralasian database, Vol. 27 CRPIT 04 pp. 171-179, January 2004.

[27] Little, M., *Transactions and Web Services*, Communications of the ACM, Vol. 46, No. 10pp. 49-54, October 2003.

[28] Malone, T. What is Coordination Theory?, Natiaonal Sicence Foundation MIT, pages 1, 29, 1988.

[29] Motgomery, D., Design and Analysis of Experiements Wiley 2001.

[30] Oaks, S., Wong, H., Java Threads, 3rd. Edition, O'Reilly, 2004.

[31] Oasis Web Services Security Technical Committee, *Web Services Security (WS-Sec) Protocol*, January 2004; www.oasis-open.org/committees/wss.

[32] ODCB Basics, www.datadirect.com/developer/odbc/basics/index.ssp December 2006.

[33] Peek, J., O'Reilly, T., Loukides, M., UNIX Power Tools O'Reilly & Assoc. Inc., 1997.

[34] Peltz, C., Web Services Orchestration and Choreography, IEEE Computer Society, pages 46, 52, October 2003.

[35] Porter, G. Effective Web Service Load Balancing Through Statistical Monitoring, Communications of the ACM, volume 49, pages 49, 54, March 2006.

[36] PostgreSQL, www.postgresql.com December 2006.

[37] Ramakrishnan, R., and Gehrke, J., Database Management Systems, 3rd. Edition, International Edition, McGraw-Hill, ch. 2, pp. 25-27, 2003.

[38] Roberts, J., Srinivasan, K. The Tentative Hold Protocol, W3C, 2001.

[39] Sesay, S., Yang, Z., Chen, J., and Xu, D., *A Secure Database Encryption Scheme*, Second IEEE Consumer Communications and Networking Conference 2005 (CCNC 2005), pp. 49-53, January 2005.

[40] Silberschatz, A., Cagne, G., Baer, P., Operating Systems Concepts, 7th. Edition, Wiley, 2005.

[41] Sun Microsystems, Inc., Java Technology, www.java.sun.com December 2006.

[42] Transaction Processing Council, www.tpc.org/tpcc/ December 2006.

[43] Turban, E., Dorothy, L., Ephraim, M., Wetherbe, J., Information Technology for Management, 5th. Edition, Wiley, 2006.

[44] UDDI Spec Technical Committee, *UDDI Version 3.0.2*, October 2004; www.uddi.org/pubs/uddi_v3.htm.

[45] XML Protocol Working Group, *Simple Object Access Protocol (SOAP) Version 1.2*, W3C Recommendation, June 2003; www.w3.org/TR/soap/.

APPENDICES

APPENDIX A

Technical and Implementation Details

For the completion of the experiments, the following computers were used:

DB Server:

Model	Dell PowerEdge 2850
CPU	2xIntel Xeon Processor at 3.0GHz with EM64T and HT / 1MB L2
FSB	800MHz
RAM	2x1024MB = 2048MB DDR2 400MHz Dual-Channel
HD	3x146GB Ultra 320 SCSI 10000rpm in RAID 5 (292GB effective)
OS	SUSE Linux 9.2 for x86_64

JSC:

Model	Dell PowerEdge 2850
CPU	Intel Xeon Processor at 3.0GHz with EM64T and HT / 1MB L2
FSB	800MHz
RAM	2x512MB = 1024MB DDR2 400MHz Dual-Channel
HD	146GB Ultra 320 SCSI 10000rpm
OS	SUSE Linux 9.2 for x86_64

Client 1:

Model	Dell Inspiron 8500
CPU	Intel Pentium 4 M Processor at 2.20GHz / 512KB L2
FSB	566MHz
RAM	512MB + 1024MB = 1536MB DDR 333MHz
HD	80GB Ultra ATA/100 IDE 5400rpm
OS	SuSE Linux 10.1

Client 2 - 16:

Model	Dell Precision Workstation 360
CPU	Intel Pentium 4 Processor at 3.2GHz / 512KB L2
FSB	800MHz
RAM	4x256MB = 1024MB DDR 333MHz
HD	80GB Ultra ATA/100 IDE 7200rpm
OS	SuSE Linux 10.1

- The network at which all the computer were connected consisted basically of 100 Mb/s Ethernet connections. Also the computers were in the same LAN.
- The programming language used was the Object-Oriented Sun Java version 1.5.6 at all system components.
- The Web services implementation used was Apache Axis2 [1] version 1.0.
- The Web server and servlet container on which all of the code ran was Apache Tomcat [2] version 5.5.9.
- The ADBs were implemented using the PostgreSQL [36] version 8.1.

The complete relational schema used along with the initial data for the ADB databases are represented by the following SQL scripts:

```
-- PostgreSQL Script

DROP TABLE ws_component;

DROP TABLE work_dispatcher;

DROP TABLE transstati;

DROP TABLE batchstati;

DROP TABLE webswitch_client;

DROP TABLE tcoll_catalogue;

DROP TABLE tproc_catalogue;

CREATE TABLE tproc_catalogue(
        description varchar(35) default null,
        ip varchar(15) not null,
        metric_status varchar(10) not null,
        load_avg numeric not null,
        batch_count integer not null,
        PRIMARY KEY (ip)
);

CREATE TABLE tcoll_catalogue(
        description varchar(35) default null,
        ip varchar(15) not null,
        metric_status varchar(10) not null,
        load_avg numeric not null,
        batch_count integer not null,
        PRIMARY KEY (ip)
);

CREATE TABLE webswitch_client(
        client_id varchar(15) not null,
        description varchar(30) not null,
        since bigint not null,
        until bigint not null,
        PRIMARY KEY(client_id)
);

INSERT INTO webswitch_client VALUES('client1', 'client1', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client2', 'client2', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client3', 'client3', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client4', 'client4', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client5', 'client5', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client6', 'client6', 100101010, 12000000);
```

```
INSERT INTO webswitch_client VALUES('client7', 'client7', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client8', 'client8', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client9', 'client9', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client10', 'client10', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client11', 'client11', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client12', 'client12', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client13', 'client13', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client14', 'client14', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client15', 'client15', 100101010, 12000000);
INSERT INTO webswitch_client VALUES('client16', 'client16', 100101010, 12000000);

CREATE TABLE batchstati(
        name varchar(30) not null,
        status varchar(15) not null,
        batch_id varchar(20) not null,
        dir_name varchar(100) not null,
        tcoll_ip varchar(15) not null,
        receiving_time bigint not null,
        priority bigint not null,
        client_id varchar(15) not null,
        tracking_num BIGSERIAL UNIQUE,
        FOREIGN KEY(tcoll_ip) REFERENCES tcoll_catalogue(ip),
        FOREIGN KEY(client_id) REFERENCES webswitch_client(client_id),
        PRIMARY KEY(batch_id)
);

CREATE TABLE transstati(
        status varchar(10) not null,
        batch_id varchar(20) not null,
        trans_id varchar(10) not null,
        FOREIGN KEY(batch_id) REFERENCES batchstati(batch_id),
        PRIMARY KEY(trans_id)
);

CREATE TABLE work_dispatcher(
        batch_id varchar(20) not null,
        tproc_ip varchar(15) not null,
        status varchar(15) not null,
        receiving_time bigint not null,
        priority BIGSERIAL,
        PRIMARY KEY(batch_id),
        FOREIGN KEY(batch_id) REFERENCES batchstati(batch_id),
        FOREIGN KEY(tproc_ip) REFERENCES tproc_catalogue(ip)
);

CREATE TABLE ws_component(
        ip varchar(15) not null,
        description varchar(60) default null,
```

```
        PRIMARY KEY(ip)
);


INSERT INTO ws_component VALUES('136.145.116.92', 'Transaction Coordinator');
INSERT INTO ws_component VALUES('136.145.116.77', 'Transaction Collector');
INSERT INTO ws_component VALUES('136.145.116.84', 'Transaction Collector');
INSERT INTO ws_component VALUES('136.145.116.90', 'Transaction Collector');
INSERT INTO ws_component VALUES('136.145.116.91', 'Transaction Collector');
INSERT INTO ws_component VALUES('136.145.116.97', 'Transaction Collector');
INSERT INTO ws_component VALUES('136.145.116.121', 'Transaction Collector');
INSERT INTO ws_component VALUES('136.145.116.28', 'Transaction Dispatcher');
INSERT INTO ws_component VALUES('136.145.116.29', 'Transaction Processor');
INSERT INTO ws_component VALUES('136.145.116.30', 'Transaction Processor');
INSERT INTO ws_component VALUES('136.145.116.31', 'Transaction Processor');
INSERT INTO ws_component VALUES('136.145.116.32', 'Transaction Processor');
INSERT INTO ws_component VALUES('136.145.116.69', 'Transaction Processor');
INSERT INTO ws_component VALUES('136.145.116.111', 'Transaction Processor');
```

The Traffic Ticket Solution ADB relational schema is represented by the following SQL script:

```
-- PostgreSQL Script

DROP TABLE ticket_payment_adjudication;

DROP TABLE ticket_payment;

DROP TABLE ticket;

DROP TABLE police;

DROP TABLE precinct;

DROP TABLE driver;

DROP TABLE vehicle;

CREATE TABLE vehicle(
        lic_plate varchar(6) not null,
        vin varchar(10) not null,
        deed_title varchar(10) not null,
        make varchar(10) not null,
        year integer default null,
        sys_id BIGSERIAL UNIQUE,
        PRIMARY KEY(vin, sys_id)
```

```
);

CREATE TABLE driver(
        fname varchar(10) not null,
        mname varchar(10) default null,
        lname varchar(10) not null,
        ssn integer not null,
        lic_num integer not null,
        sys_id BIGSERIAL UNIQUE,
        PRIMARY KEY(lic_num, sys_id)
);

CREATE TABLE precinct(
        name varchar(20) not null,
        address varchar(50) not null,
        pre_id integer not null,
        PRIMARY KEY(pre_id)
);

CREATE TABLE police(
        fname varchar(10) not null,
        mname varchar(10) not null,
        lname varchar(10) not null,
        ssn integer not null,
        badge_num integer not null,
        pre_id integer not null,
        sys_id BIGSERIAL UNIQUE,
        FOREIGN KEY(pre_id) REFERENCES precinct(pre_id),
        PRIMARY KEY(badge_num)
);

CREATE TABLE ticket(
        ticket_id integer not null,
        timestamp bigint not null,
        place varchar(20) not null,
        description varchar(50) default null,
        lic_num integer default null,
        lic_plate varchar(6) default null,
        amount integer not null,
        badge_num integer not null,
        trans_id varchar(10) not null,
        FOREIGN KEY(badge_num) REFERENCES police(badge_num),
        FOREIGN KEY(trans_id) REFERENCES transstati(trans_id),
        PRIMARY KEY(ticket_id)
);

CREATE TABLE ticket_payment(
        ticket_payment_id BIGSERIAL,
```

```
        amount integer not null,
        submit_time bigint not null,
        ttrans_id varchar(10) not null,
        ticket_id integer not null,
        trans_id varchar(10) not null,
        FOREIGN KEY(ticket_id) REFERENCES ticket(ticket_id),
        FOREIGN KEY(trans_id) REFERENCES transstati(trans_id),
        PRIMARY KEY(ticket_payment_id)
);

CREATE TABLE ticket_payment_adjudication(
        tpa_id BIGSERIAL,
        t_id integer not null,
        t_payment_id integer not null,
        system_adjudication_time bigint not null,
        FOREIGN KEY(t_id) REFERENCES ticket(ticket_id),
        FOREIGN KEY(t_payment_id) REFERENCES ticket_payment(ticket_payment_id),
        PRIMARY KEY(tpa_id)
);
```

The relational schema used to log the experimental results is represented by the following SQL script:

```
-- PostgreSQL Script

DROP TABLE batch_protocol_measures;

CREATE TABLE batch_protocol_measures(
        ip varchar(15) not null,
        batch_id varchar(20) not null,
        time bigint not null,
        cilent_id varchar(20) default null,
        status varchar(15) not null,
        description varchar(60) default null,
        FOREIGN KEY(ip) REFERENCES ws_component(ip),
        PRIMARY KEY(batch_id, time, ip)
);
```