ARCHITECTURE REPORT

PREPARED FOR DAYTRADING INC

PROJECT NAME:	DAY TRADING SOFTWARE
SYNOPSIS:	AN OUTLINE OF THE PROJECT MILESTONES, COMPLETION OF OBJECTIVES, AND A DESCRIPTION OF THE FINAL ARCHITECTURE THAT WAS IMPLEMENTED.
DATE:	APRIL 7, 2015
PREPARED BY:	BRAIGHTON POLACK (V00763687), EVAN WILLEY (V00703788)

CONTENTS

1. INTRODUCTION

The purpose of this document is to outline the business requirements of the client, and to show how the project evolved during its development. Included is a detailed description of the software and reasons why each component was chosen. The documentation also compares the original plan for the software architecture, and the final architecture that was implemented.

2. BUSINESS REQUIREMENTS

DayTrading Inc. has contracted this company to develop and end-to-end prototype of their complete day trading system. The overall design goals of this prototype are as follows.

- Cost effective scalability
- Transaction performance
- Efficient system capacity vs. development cost
- Minimizing costs associated with stock quotes (5 cent charge per quote)
- Speed and reliability

The basic structure of the day trading system is to support a large number of remote clients through a web browser interface, which interacts with a centralized transaction processing system. Upwards of 1000 clients at a time are to be tested. The system must support the following activities.

- View account balance
- Add to account balance
- Retrieve stock quote
- Buy and sell shares in a stock
- Set automated buy and sell points for a stock
- Review full transaction history
- Cancel or commit transactions in a two-step buy/sell process
- Dump companywide transaction logs for auditing

These requirements have been thoughtfully incorporated into our plan for the project architecture, and will be used to test the design at several milestones.

3. PROJECT MILESTONES AND RESULTS

The project milestones are listed below, along with our highest recorded TPS for each milestone. The highest TPS for the final implementation was noted here rather than some of the higher values corresponding to versions that were deemed not usable.

Date	Description	Highest TPS	Quote Count
Feb 6	Execute 2 user workload file.	330.502	22
Feb 20	Execute 10 user workload file.	652.571	123
Feb 27	Execute 45 user workload file.	662.12	248
Mar 5	Execute 100 user workload file.	2340.44	4583
Mar 12	Execute 1000 user workload file.	5317.169	16378
Apr 6	Execute 1250 user workload file.	5471.178	24047
Apr 7	Execute final workload file. (1400 user)	3877.141	46105

4. ORIGINAL ARCHITECTURE PLAN

The architecture for this project was designed with the following goals in mind.

- Fast transaction processing
- Reliability and fault recoverability
- Efficient use of resources and scalability
- Security
- Clean design and easy to use interface

Below is a diagram that shows the projected structure of the day trading software. Each level is intended to be horizontally scalable, and depending on the client traffic, can be minimized to reduce costs. A larger view of this diagram is attached to this document.

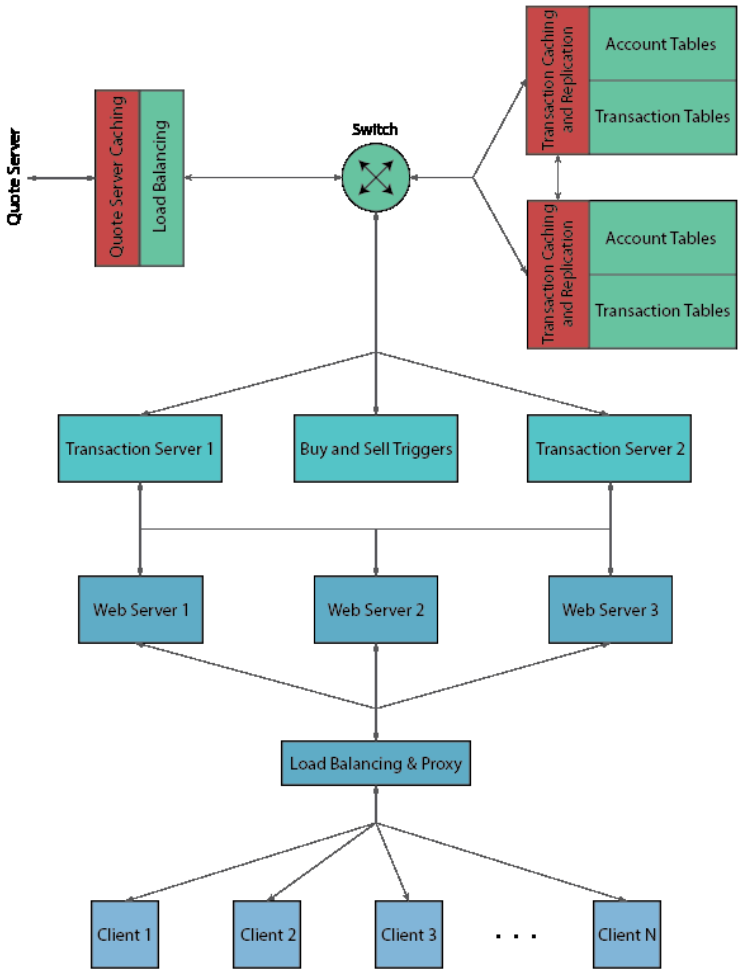

FIGURE 4.1 – DIAGRAM OF PROJECT ARCHITECTURE.

As shown in the diagram above, the project is split into several modules; the development overview of each module will be described in the sections below. The purpose, language, and any libraries used are shown for each module.

4.1. DATABASE SERVERS

The database servers are used to store all user account and transaction information. They will also contain an intermediary buffer between the transaction servers and the database itself.

Structure	A data buffer will serve as an intermediary between the postgreSQL database and the transaction servers. This buffer will be responsible for performing account transactions in memory, then flushing all changes to the database on a regular interval. The buffer will also be used to replicate data between databases if redundant databases are set up. The database itself will contain four tables. The primary table will hold user accounts and balances. Another table will hold owned stocks with reference to user ID. The third table will hold records of each transaction, and the fourth table will hold reserved money for performing automatic buy/sells.
Language	C & SQL
Software Used (if any)	PostgreSQL

4.2. QUOTE LOAD BALANCER

The quote load balancer will be used to interact with the quote server. Its main function is to prevent constant requests to the quote server, in order to lower costs. It will also decrease the amount of time it takes to get a stock quote.

Structure	The load balancer will accept incoming connections from the transaction servers, and reply with the requested stock quote. After a particular quote is requested, the response from the quote server will be stored in a memory cache for later use. These quotes will only be valid for a period of 30 seconds, and if a quote is expired the load balancer is responsible for obtaining a new one.
Language	C
Software Used (if any)	None

4.3. TRANSACTION SERVERS

The transaction servers interact with almost every other component, being the middleman between each transaction. They will be scalable, with the web servers able to distribute requests between them evenly.

Structure	The transaction servers will accept incoming requests from the web servers, and process each depending on the type of command received. The transactions will be put into threads and queues depending on user ID. The servers will communicate with either the quote load balancer or the database servers depending on the type of command received.
Language	C
Software Used (if any)	None

4.4. WEB SERVERS

The web servers are responsible for hosting the user interface, as well as creating requests for the transaction servers.

Structure	The web servers will employ SSL encrypted connections to provide security between the clients and the system. When a command has been sent to one of the web servers, it will be responsible for passing the request along to one of the transaction servers; depending on the current load of any given transaction server. The web servers will be configured using apache, and the user interface itself will use a combination of HTML, JS, and PHP.
Language	HTML, PHP, JS
Software Used (if any)	Apache

5. FINAL ARCHITECTURE

The final implementation of this software was decided due to the following parameters.

- Data consistency and database stability
- Efficient use of resources
- Scalability of Web / Transaction Servers
- Reliable fault tolerance
- Security
- Ease of use for end user

Below is a diagram showing the overall structure of the final implementation. The web and transaction servers are intended to be fully scalable to best meet the client's needs.

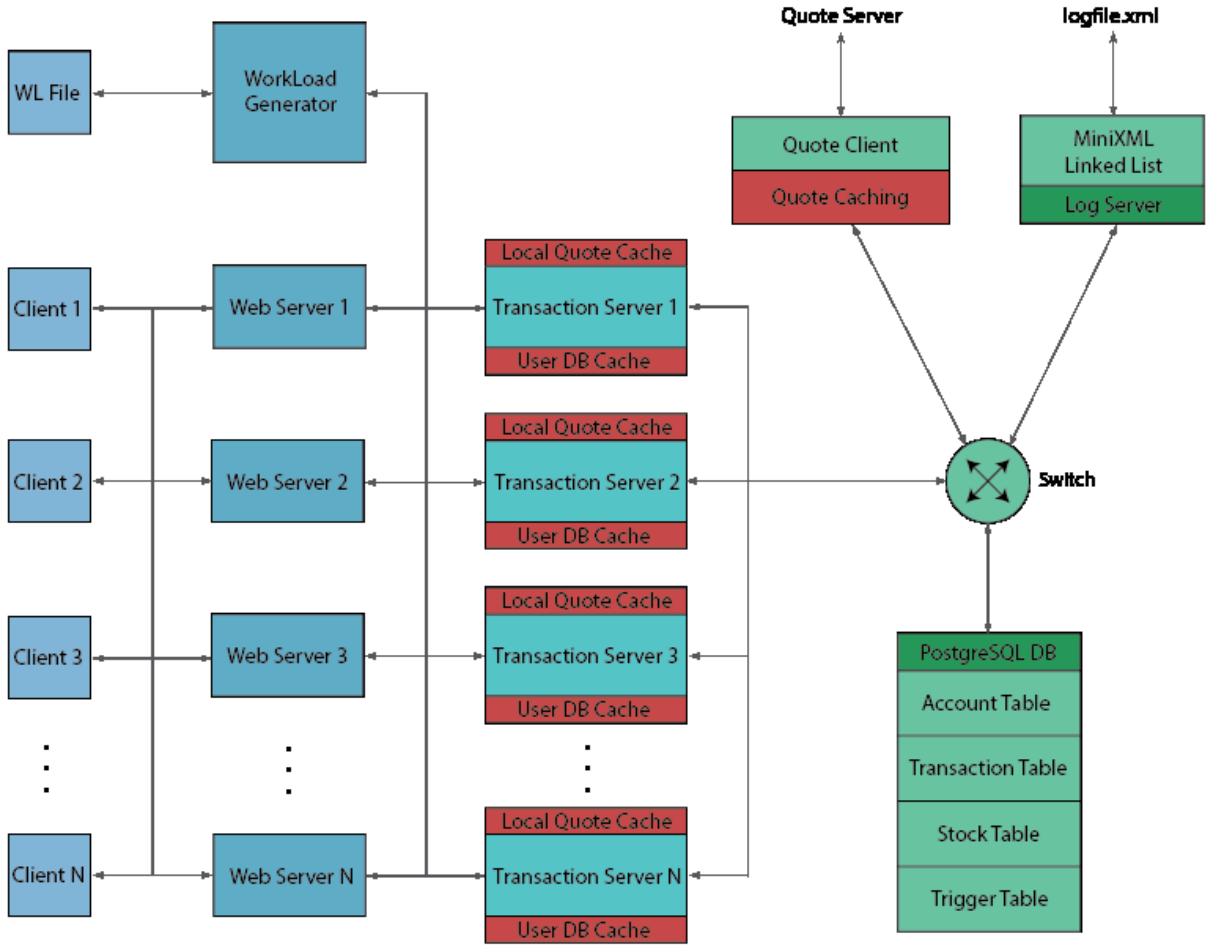

FIGURE 5.1 – DIAGRAM OF PROJECT ARCHITECTURE.

5.1. DATABASE

The database server is used to store all user account, transaction information, owned stocks, and triggers. Our final implementation uses a single PostgreSQL database. This was chosen for its ease of use, fault tolerance, and relatively fast transaction speeds compared to other database solutions. A Postgres-XL database solution was tested, which involves multiple PostgreSQL datanodes spread across a number of servers. This solution was theorized to increase the maximum write speed to the hard disks, but proved to have no significant gain of write speed. In some tests, one single PostgreSQL database performed faster inserts than the Postgres-XL cluster.

As the database in our final solution is a single un-scalable unit, improvement could be made in this area. An alternative database solution, which could be scaled to improve inserts per second, would be ideal. A potential solution would be to distribute data to a set of Postgres databases directly from each transaction server, via the hashed user ID of each inserted row.

Structure	A PostgreSQL database serves the transaction servers directly. The database contains four tables. The primary table holds user accounts and balances. Another table holds owned stocks with reference to user ID. The third table holds records of each transaction, and the fourth table holds reserved money for performing automatic buy/sells. The majority of actions performed on the database are writes. Due to the caching of relevant user information on each transaction server, reads are only required when a new user connects to a transaction server. In order to not lose any data in the event of the database crashing, the setting Synchronous Writes is kept on. This is discussed further in the performance report.
Language	N/A
Software Used (if any)	PostgreSQL

5.2. QUOTE LOAD BALANCER

The quote load balancer is used to interact with the quote server. Its main function is to prevent constant requests to the quote server, in order to lower costs. It also increases the number of quotes that can be received from the quote server to roughly 100 quotes per second. Transaction servers support the use of multiple quote load balancers for scalability. If multiple load balancers are in operation, the transaction servers can iterate through the list of load balancers to distribute the quote load.

Structure	The load balancer accepts incoming connections from a transaction server, and replies with the requested quote. After a particular quote is requested once, the quote will be stored in a cache to be used for another 30 seconds. If a quote is expired the load balancer is responsible for obtaining a new one upon another request. One quote is requested from 10 threads simultaneously, and the fastest response is taken. This is done using non-blocking receive functions which are interrupted by a successful thread.
Language	C
Software Used (if any)	None

5.3. TRANSACTION SERVERS

The transaction servers interact with almost every other component, being the main handler for each transaction. They are scalable, and the web servers are able to distribute requests between them equally as new users log in. Each transaction server in the system has the ability to process between 3000 and 5000 transactions per second; depending on several variables, such as current trigger load.

Structure	The transaction servers accept incoming requests from the web servers, and process each depending on the type of command received. When a transaction server obtains a user request from a new user; it indicates whether or not the user is already present in cached data, or with the total active user count. The web server can then decide what transaction server to select. While processing each transaction, any new data is sent to the database then updated in the local cache of the user data. User data is cached in a trie structure based on user ID. Quotes are also cached locally as they are obtained from the quote load balancer. Each server can be configured to use a set number of threads simultaneously, allowing it to process transactions from multiple users at a time. The transaction servers use one thread to cycle through each of the active triggers and compare new quotes to trigger value.
Language	C
Software Used (if any)	None

5.4. WEB SERVERS

The web servers host the user interface, as well as create requests for the transaction servers based on user input.

Structure	When a user logs into the web interface, a php function iterates between each transaction server to select the ideal one for that user. The same function is called again if a user fails to connect to their selected server. The ideal server is selected based on number of active users on each transaction server, and if the user already has cached data on a server. Commands are created based on user input forms, which are regex filtered to control invalid inputs. The web servers are configured using apache, and the user interface itself uses a combination of HTML, JS, and PHP. Key information is displayed on each page, such as time to expire for buy and sell actions, as well as a recent commands log.
Language	HTML, PHP, JS
Software Used (if any)	Apache

5.5. LOG SERVERS

The log server receives constant log information from each component in the system. This data is stored in memory and periodically written to an xml file using the MiniXML library. Transaction servers support the use of multiple log servers for scalability. If multiple log servers are in operation, the transaction servers can iterate through the list of log servers to distribute the load.

Structure	Every component in the system is configured to have a log queue, and a thread, which sends logs constantly to the log server. Once received, a log is categorized by type based on the project requirements, and inserted into a MiniXML linked list. The structure of this information is shown in the figure below. MiniXML is an extremely small library which is capable of writing and reading xml files from/to a linked list of data very quickly. The data is either written to file when a DUMPLOG command is received, or is written to file during a set interval of time.
Language	C
Software Used (if any)	MiniXML

For example, if you have an XML file like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<data>
    <node>val1</node>
    <node>val2</node>
    <node>val3</node>
    <group>
        <node>val4</node>
        <node>val5</node>
        <node>val6</node>
    </group>
    <node>val7</node>
    <node>val8</node>
</data>
```

the node tree for the file would look like the following in memory:

```
?xml version="1.0" encoding="utf-8"?
  |
data
  |
node - node - node - group - node - node
  |      |      |      |       |      |
val1   val2   val3     |     val7   val8
                       |
                  node - node - node
                    |      |      |
                  val4   val5   val6
```

FIGURE 5.5.1 – MINIXML DATA STRUCTURE COMPARED TO XML FILE

6. CONCLUSION

In conclusion, the selected final architecture for the day trading software sacrifices some performance gains, for a more fault tolerant database. PostgreSQL is configured in the final implementation to respond with success to a connection, only once data has been completely written to the disk. This avoids any potential issue with the database going down in the middle of a single or series of commands being executed. To see a more in depth analysis of this, please refer to the performance and fault tolerance reports. Other significant features of the architecture are a fully scalable and efficient transaction server. The quote client is also very fast, and able to keep up with roughly 100 non-cached quotes per second. Group 2 believes this software to be the best solution due to its efficient components, and fault tolerant architecture. The one area of growth that could be considered is the database design.