PERFORMANCE & CAPACITY REPORT

PREPARED FOR DAYTRADING INC

PROJECT NAME: DAY TRADING SOFTWARE

SYNOPSIS: AN ANALYSIS OF THE OVERALL PERFORMANCE OF THE SYSTEM BASED ON CAPACITY, SPEED, FAULTS, AND COMMAND DENSITY.

DATE: APRIL 7, 2015

PREPARED BY: BRAIGHTON POLACK (V00763687), EVAN WILLEY (V00703788)

CONTENTS

1. INTRODUCTION

The purpose of this document is to show the performance of the system as a whole, and to indicate any bottlenecks or components that could be adjusted to increase performance. The analysis of each component is also discussed with regard to transactions per second, and the resources each component uses on the server.

2. OVERALL PERFORMANCE

The stock trading system is fully scalable in that additional web servers, transaction servers, log servers, and load balancers can be added to support additional system load. Though for the purposes of this report, performance data based on number of log servers and load balancers was not recorded, and only one of each was used.

The primary bottlenecks of the system are the database at low quote loads, and the load balancer at high quote loads. If a particular workload has more than 1000 unique stock symbols, performance of a single load balancer begins to slow the system down. As seen in the difference between the 1250 and 1400 user workload file, performance dropped to less than 4000 transactions per second from over 5000. The database is the main bottleneck of the system when the load balancer is not being overloaded. A single isolated PostgreSQL database can only write on average 5000 transactions per second without turning off synchronous commit, or other unsafe settings.

The mean transactions per second for the 100 user workload file, which was used to test most performance using 5 transaction servers, is shown below. The system configuration tested in this report is using 5 PostgreSQL databases, which are written to in a distributed manner to increase maximum write speed.

Mean TPS =	4789.23
Standard Error =	+/- 56.66

The highest performance for each workload file is shown below, using 3 transaction servers and the most stable options on a single Postgres database. The 100 user workload file generally attains a lower TPS then the 1000 user workload file due to its shorter length. The initial slowdowns from database reads and uncached quotes impact the total TPS more with a shorter length. The 1250 and 1400 user workload files are slower than the 1000 user workload file due to the significantly larger number of unique stock symbols in each.

Table 1 - highest TPS for each Workload

Description	Highest TPS	Quote Count
Execute 2 user workload file.	330.502	22
Execute 10 user workload file.	652.571	123
Execute 45 user workload file.	662.12	248
Execute 100 user workload file.	4806.768	2563
Execute 1000 user workload file.	5317.169	16378
Execute 1250 user workload file.	5471.178	24047
Execute final workload file. (1400 user)	3877.141	46105

3. ANALYSIS OF COMPONENTS

In the sections below, the performance for each component of the system is discussed. The impact on the system of each component, and the decisions made to implement each is shown.

3.1. TRANSACTION SERVERS

The performance of a single transaction server under ideal conditions is between 3000 and 5000 transactions per second. This varies based on the number of quote requests and triggers being processed. Under realistic conditions the average TPS of a single transaction server is a little more than 2000 TPS. The CPU load and memory of a single transaction server is shown in the figure below. The memory usage of each transaction server is very low. The following data was recorded for the 100 user workload file.

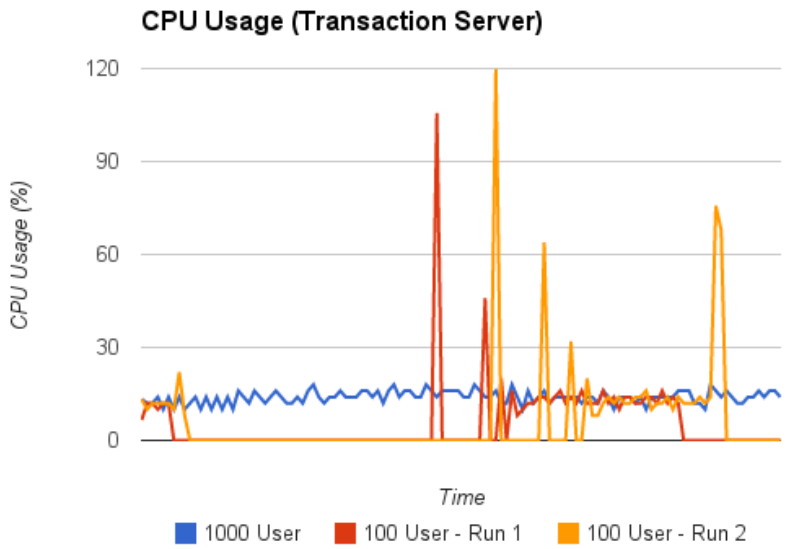

Figure 1 – CPU Usage for Transaction Server

Due to difficulty obtaining nice CPU usage measurements for the transaction server on the 100 user workload file, the steady state CPU usage for the 1000 user workload file is also included. As shown above in blue, it matches the 100 user runs during the short period of time proper data was being obtained. The average memory usage for the transaction server when running the 100 user workload was ~15MB, and for the 1000 user it was ~140MB. The memory usage is primarily affected by the number of users executing commands on it. The CPU usage and overall performance of the transaction servers, is mostly affected by the number of commands it is processing (except with low numbers of users).

Performance based on the number of transaction servers being used is shown in the figure below. The system is configured to distribute to 5 databases, and uses 2 load balancers in these tests. The data compares the impact of 1, 3, and 5 transaction servers on the average TPS using the 100 user workload file.

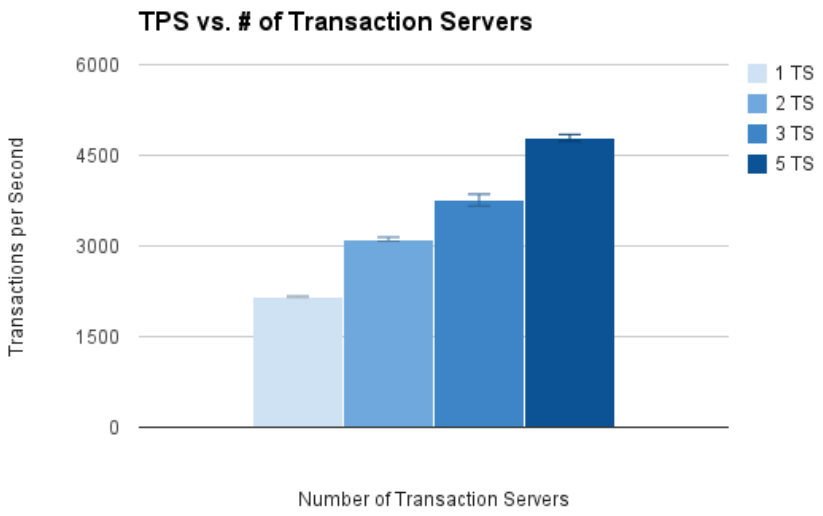

Figure 2 – TPS vs. # of Transaction Servers

3.2. DATABASE

The database component of the system has evolved the most due to performance concerns. Initially one single PostgreSQL database was being used, with the synchronous commit option disabled. Due to the cached user data on the transaction server, the majority of database interaction is writes. Turning synchronous commit off maximized potential transactions per second writing to the database, though this comes at a significant cost. With the setting turned off, Postgres does not wait until the data has been written to the hard drive before responding with success to the transaction server. Rather, SQL commands are queued up in a buffer of a set max size. In the case of the database server crashing, all the commands that have yet to be committed to the disk would be completely lost.

For superior fault tolerance it was decided that synchronous commits should be left on. A comparison of transaction speeds for the 1000 user workload file is shown below with both synchronous commit on and off.

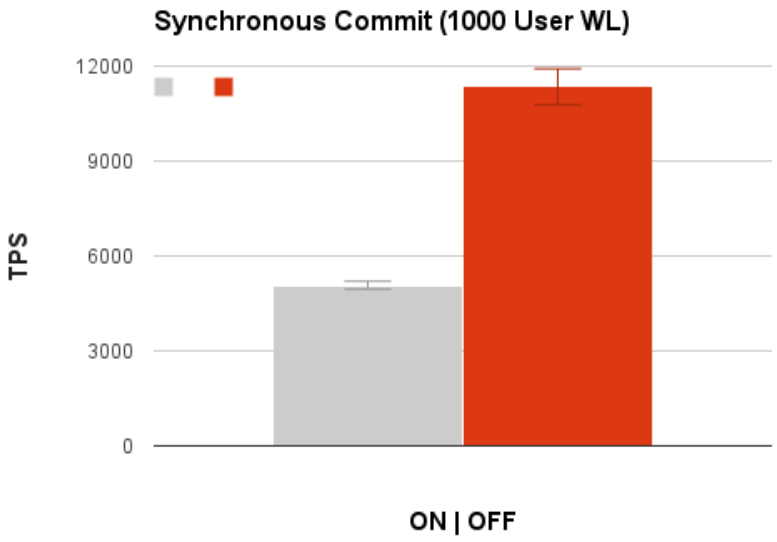

Figure 3 – Synchronous Commit On & Off

In attempting to overcome the bottleneck that a single PostgreSQL database causes on the system, a modified version called Postgres-XL was tested. Postgres-XL supposedly increases maximum write speed by distributing tables among multiple datanodes using a coordinating node. The structure of this setup is shown in the diagram below.

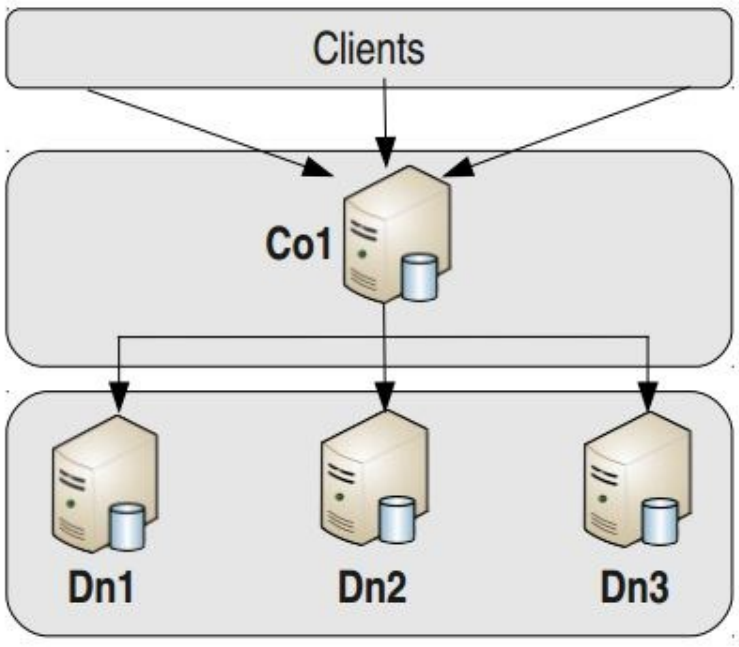

Figure 4 - Postgres-XL implementation

After much testing of this implementation, no performance gains were seen when comparing Postgres-XL to a single Postgres database. In the final implementation, it was decided to allow for a hybrid of this solution. The final system supports the addition of more PostgreSQL databases by distributing tables directly from the transaction servers based on user ID.

The CPU and memory usage of each Postgres database is almost negligible, and an efficient design could involve putting the databases on the same hardware as other, more CPU intensive components.

3.3. LOAD BALANCER

While isolated from the system, the load balancer is able to return roughly 100 uncached quotes per second. This is fast enough up until the 1400 user workload file, which has more than 4300 unique stock IDs. To run the final workload, an additional load balancer was set up so there were two to share among the transaction servers.

The CPU and memory usage of the load balancer under moderate quote load is shown in the figure below. This data was obtained while running the 100 user workload file.

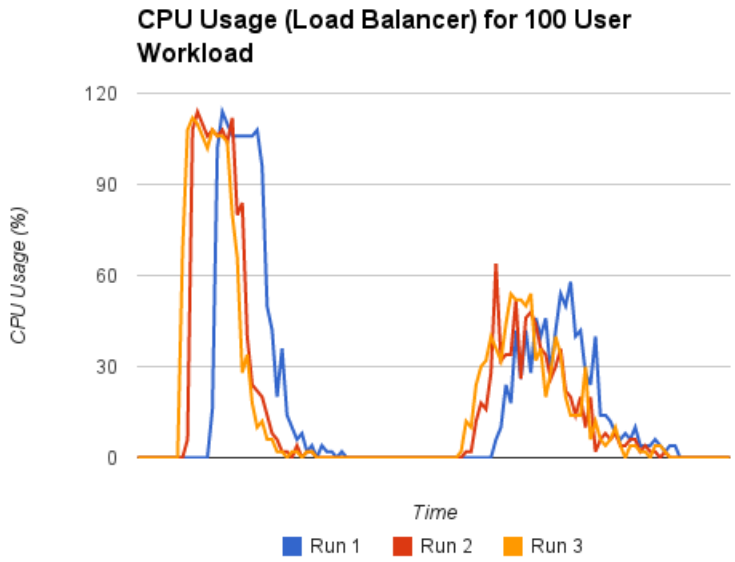

Figure 5 – CPU Usage for Load Balancer

The two pronounced peaks are the times at which the load balancer had cached quotes that were either expired, or hadn't been cached yet. The memory usage for the load balancer during these runs was ~13MB. The CPU usage, memory, and overall performance of the load balancer, are only affected by the number of different stocks in the workload file.

3.4. LOG SERVER

The log server is able to keep up with all logging below transaction speeds of 10000 TPS. This was tested by setting the database up for maximum write speed, and running the 1000 user workload file. If the system runs at anything above 10000 transactions per second, a single log server can not keep up with the incoming logs, and must rely on queueing to write all the logs. This could be an issue if the system were running for an extended period of time, and the memory were to fill up, so the transaction servers were configured to allow multiple log servers if needed. Using multiple log servers allows a transaction server to alternate between available log servers when sending logs. No way to compile the separate log files has been created, so this would be used for purely performance gain.

The CPU and memory usage of the log server under moderate quote load is shown in the figure below. This data was obtained while running the 100 user workload file.

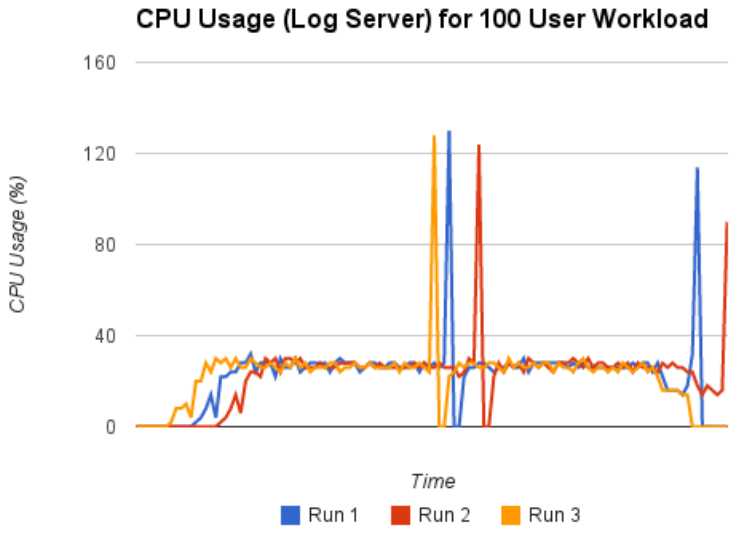

Figure 6 – CPU Usage for the Log Server

The prominent spikes shown at the middle and end of the graph are when the log server saved its memory contents to disk. The memory usage for the log server was ~130MB, which is only dependent on a variable within the program, which determines the maximum allowed logs that can be received before the XML in memory is auto-saved to disk.

4. CAPACITY

The capacity of the system is not limited by number of users connected to the day trading system while number of transactions per second being performed by the user base is similar to the speeds tested in this report. The number of users would mainly affect the memory usage on each transaction server, and due to the scalability of the transaction servers there is no foreseeable limit to this.

The limitations of the system come primarily from the maximum write speed of the database configuration in tandem with the number of transaction servers. This can be seen in the graph in section 3.1. A secondary limitation of the system is the total number of unique stock symbols available to the user base. Using a real life example, the NYSE only has approximately 2000 unique stock symbols, so this number does not need to be excessively large. However, if a larger number of stocks were required, adding multiple load balancers and splitting the stocks among them should increase performance, assuming the quote server can handle the traffic. Allowing for multiple Postgres databases through distributed tables should allow the system accept a higher database write speed, therefore increasing the capacity of the TPS. This was not able to be tested in this report, but if this project were to receive future additions, this would be an area of focus.

5. CONCLUSION

In conclusion, the performance of each individual component of the system allows for some scalability when combined. A lone transaction server performs an average of 2161 TPS for the 100 user workload file. Adding more transaction servers scales according to the graph in section 3.1. The limiting components of the system are the load balancers and the database. Multiple load balancers and databases seem to increase the potential of the system as seen in this report, though more testing would need to be done to see if these components are truly scalable. Future development of this project could involve finding an ideal database solution, which could scale easily to provide higher write speeds.