5-2018

Cognitive performance application.

Shade EL-Hadik
University of Louisville

COGNITIVE PERFORMANCE APPLICATION

By

Shade El-Hadik

B.S., American University in Cairo, 2004

M.S., University of Louisville, 2005

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

in Computer Science and Engineering

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

May 2018

COGNITIVE PERFORMANCE APPLICATION

By

Shade El-Hadik

B.S., American University in Cairo, 2004

M.S., University of Louisville, 2005

A Dissertation

Approved on FebÒÕÁÒÙ 23, 2018

By the Following Dissertation Committee:

Dissertation Director

Dr. Ahmed Desoky CECS

Dr. Adel Elmaghraby CECS

Dr. Dar-Jen Chang CECS

Dr. Juw Park CECS

Dr. Michael Losavio

ABSTRACT

COGNITIVE PERFORMANCE APPLICATION

Shade El-Hadik

January 12, 2018

This work shows that combining the techniques of neural networking and predictive analytics with the fundamental concepts of computing performance optimization is genuine in many ways. It has the potentials to: (1) reduce infrastructure upgrade costs (2) reduce human interactions, by enabling the system to learn, analyze, and make decisions on its own, and (3) generalize the solutions to other performance problems. This paper attempts to tackle a JVM performance optimization from a different dimension and in a way that can be scaled to other common utilized resources, such as file systems, static contents, search engines, web services...etc. It shows how to build a framework that monitors the performance metrics to determine patterns leading to bottleneck incidents and then benchmark these performance metrics. The framework uses artificial neural network in its core to accomplish this first steps with immediate benefit of eliminating the need to a domain expert analyzing which of these metrics is more important or has more weight on constituting the bottleneck condition, and hence enable the system to deal with more ambiguous situations. The framework uses an analytics engine, to establish predictive patterns between the system bottleneck and library of factors to establish an early alert system and thus enhancing the weight of the bottleneck signal. Finally, the framework acts in defense when the deadlock signal is triggered from the learning and/or the analytics engine through streaming down concurrent transactions into a temporarily queuing data structure. We put our model into a test and built a simulation to quantify the added benefit of each component of our framework. The results are proven to demonstrate the immediate benefit of our framework and open doors for other future work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Cl!t¢9w 1 INTwODUCTION

Applications running on Java Virtual Machine (JVM) are extensively adopted in numerous industrial and commercial applications around the world. The main advantage in Java is the concept of parallel programing or multiple threading and the fact that it can run in different operating system without the need to change the coding syntax. In the commercial world, Java is mainly known for its web programing capabilities. While this is all true and considered redundant info, Java applications running on multiple JVM's on single or multiple servers usual end up exploiting common resources such as a database management system (DBMS). Database management systems are more complex, more rapidly evolving, and more essential to the conduct of business than those of even a few years ago. The result is an increasing need for techniques that assist in optimizing the overall performance of these database driven systems in order to enhance the system availability and the overall user experience in a world where website up times and mobile apps are now expected to have 99.99% around the clock availability. The need for new techniques that help these systems avoid bottleneck situations in a world where technical glitches translate into a negative user experience and therefor a lost business opportunity. The challenge is that relational databases were not designed to be scaled in distributed model, such as the cloud architecture, which leads the entire solution to halt or crash in many situations. In this paper we will use a customized database known as an e-store to simulate real-world commercial Java application such as an e-commerce store that rely heavily on DB integration.

There are many theoretical and practical proposed techniques that stopped short from providing a comprehensive solution to the current challenges stated above. This document roles off some of these proposals, and builds upon other ones, such as “A Queuing Model to Achieve Proper Elasticity for Cloud Cluster Jobs,” [1] which focuses on providing an understanding of the database behavior through the introduction of an analytical model based on queuing theory to determine at any given time the workload conditions and the minimum number of computing resources needed for executing query jobs on a cloud cluster. Another example of academic work that helped shaping the direction of our solution is the “Performance Modeling and Analysis of a Database Server with Write-Heavy Workload,” [2] which studies the performance anomalies dynamics that are difficult to monitor and control, through the development of a queuing based performance model for database servers with write-heavy workload. In both papers and in other papers such as “Solving Enterprise Applications Performance Puzzles - Queuing Models to the Rescue,” [3] the authors use a queuing model to rectify the system performance which we will prove is necessary and will constitute some portion of our work as explained in the subsequent sections of this document.

On the other hand, we also examined more conventional or widely adopted commercial solutions related to the same challenges, and we found that they mainly evolve around four main concepts listed below:

- Increase Memory
- Caching,
- Query optimization,
- Splitting databases

These solutions are not always the optimum from performance or cost prospective. For example, a caching solution ends up replicating the database into another database, which runs into technical issues of caching synchronization and validation and invalidation. The practice of query optimization constitutes another hurdle especially for highly integrated complicated queries that were evolving over time to serve multiple enterprise agents. Finally, the option of splitting the system database into two always introduces more complexities such as keeping the enterprise database systems synchronized.

This document discusses a cognitive or an AI alternative approach to autonomously enhance the system availability and optimize the performance of a JVM interacting with common computing resources, such as a database management system. An example of such a database driven system would a Java implementation of an online store deployed on a single or multiple servers utilizing highly paralyzed query jobs resulting from multiple and concurrent user sessions. The proposed solution will rely on machine learning practices in order to introduce an integrated cognitive framework. As shown in the below figure, the Cognitive Performance Application solution proposes three correlated phases or engines. These are

- Learning Engine,
- Analytics Engine,
- Queuing Engine.

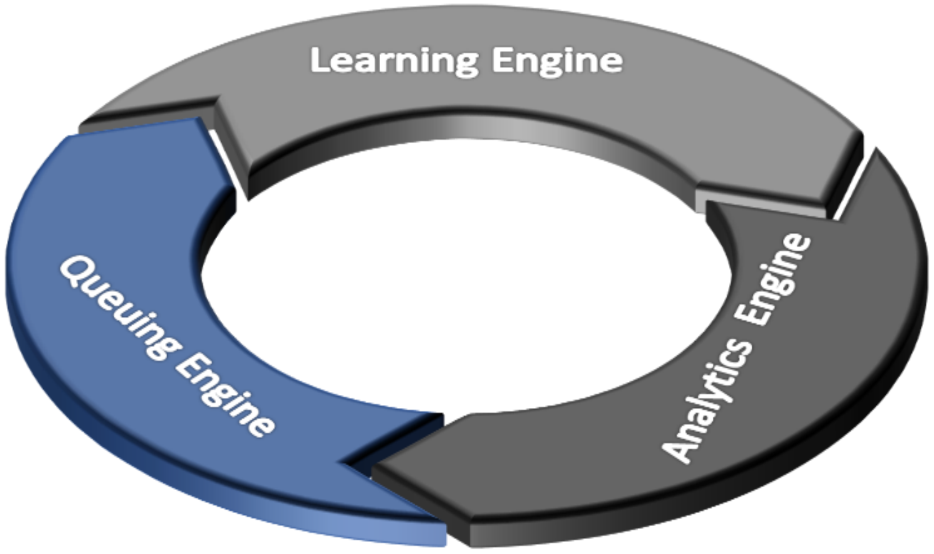


Figure 1 – CORRELATED COMPONENTS OF THE COGNITIVE PERFORMANCE APPLICATION

The first component would learn how to determine or detect a JVM performance bottleneck, by monitoring correlated parameters such as CPU utilization and query response times using neural network technology. [4] Meanwhile, solution package will also construct an advanced analytics-based alert mechanism to predict, based on historical performance records, the future bottleneck situations where the demand for the database and the other system computing resources is trending up. The high demand in a real world would have positive correlations with external library of factors such weather conditions or shopping seasonality. Finally, and based on the outcome of the learning and predictive components of this solution, a signal will be triggered for the system to rectify the bottleneck situation through building a temp queuing structure to stream down the system demand. The queuing component is elastic in nature will then dissolve on its own once the system recovers back to normal operation levels as the performance conditions improve. The process flow of the system and the different functionalities of the cognitive application components are demonstrated on the below graph, and are subject to further explanation in the subsequent segments of this document.

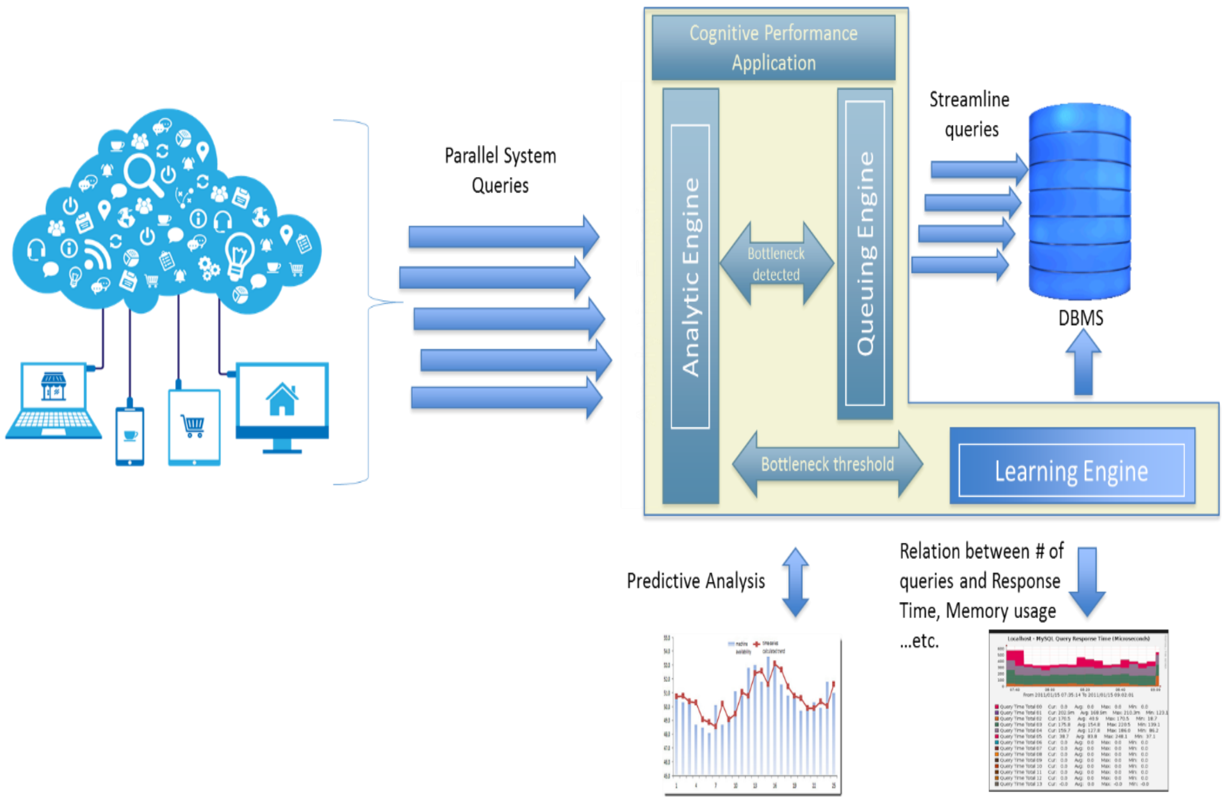


Figure 2 – *Cognitive Performance Application Diagram*

The uniqueness of this approach lies in the fact that it can be generalized to other performance problem dealing with high demand for common resources. Same approach also can be scaled horizontally to other common resources utilized by Java applications such file systems, static contents, search engines, web services…etc. Moreover, it widens the dimension of solving conventional performance challenges of database driven systems without further exhausting customary solutions, but rather shifting the focus up to the requestor layer to queue up computing jobs intelligently and only when needed to avoid bottleneck consequences. In addition, the proposed solution has a high viability for limited budget implementations, compared to other usual solutions that mandate investing on scaling the hardware specifications. Meanwhile the proposed solution has another advantage on areas like maintenance reduction and eliminating partially or totally interruption system admins, who would otherwise consistently monitor the system performance.

This document is structured on different chapters, the first of which focuses on building the components of a typical modern ecommerce store database system, based on a scaled down version of IBM WebSphere e-commerce database schema [5]. The sample store database records are populated with millions of customers, addresses, shopping carts…etc. database records. Also in the first chapter we list the system specifications under study, and then simulate the system demand with constant user load in order to take snapshots of the different performance parameters enhancing our understanding of the different correlations among them and when the system reaches a bottleneck. The document then moves to other chapters describing the three components of the Cognitive Performance Package and dives into more details regarding architecture decision, logic, code snippets, integration, statistical calculations …etc. Toward the end of this document we run different random simulations of the user load to test our solution and compare the system performance with and without our solution. We finally add another section concluding our findings and recommendations.

CI!t¢9W 2 CONSTANT LOAD SIMULATION

2.1. Simulation

A critical part of this development is to build a Java based framework to simulate a discrete system demand, as in the number of concurrent threads competing for the CPU time, which translate to number of active user sessions accommodated by the online ecommerce store server over a particular time range. To achieve this goal, the simulation model abstracts a queuing network model of a standalone system that receives and serves a group of parallel database queries (tasks) in an attempt to distil, from the mass of details that is the system itself, exactly those aspects that are essential to the system's behavior. Modeling provides a framework for congregating, organizing, evaluating, and understanding information about a current system subject to this study. Once a model has been defined through this abstraction process, it can be parameterized to reflect any of the alternatives under study, and then evaluated to determine its performance under this alternative. [6] The objective using this approach is to quantify a model, which accurately reflect the performance measures of the online store.

This simulation framework will further rely heavily on the integration between the online store and the database management system, in order to make it a query driven simulator system augmenting the likelihood of matching a typical deployment of an online store. The usage of Java multi-threading feature will make it conceivable to simulate an independent entity, or a user session browsing the system, while timing between independent database activity during certain time slots will be randomized using a normal distribution. In the queue-based system, the JVM connected to a DBMS is considered the service center where the work gets done. To accomplish a given task, it takes the JVM, a certain amount of time. If a task arrives faster than it can be processed, a queue builds and the response time grows as shown below.

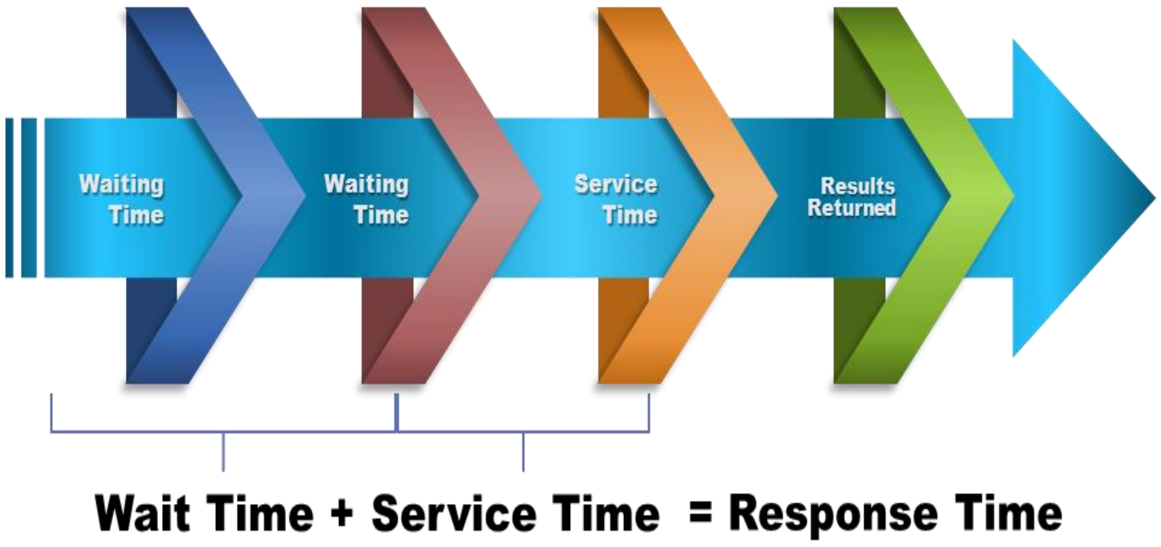


Figure 3 – TRANSACTION RESPONSE TIME

As the demand on the JVM mountains, the CPU utilization increases and it becomes more likely that a newly arriving query will have to wait since there are other queries ahead. In general, the response time degradation is more pronounced the busier the resource is. It is the responsibility of the Learning Engine to benchmark the performance metrics over a configurable period of time. In this application a number of threads will be measured against response times, memory usage, and CPU utilizations. The simulation framework will utilize different types of SQL transactions, each of which will result in different response times. The Learning Engine will be a supervised model that detects the bottleneck based on the previously mentioned system performance parameters. More of these details are covered in the next subsections.

2.2. Populating the database

To start simulating a real ecommerce solution, we will start by building and populating a similar ecommerce database. Below is the database schema diagram that is build to mimic a very known industrial ecommerce store, namely IBM WebSphere e-commerce. [5]

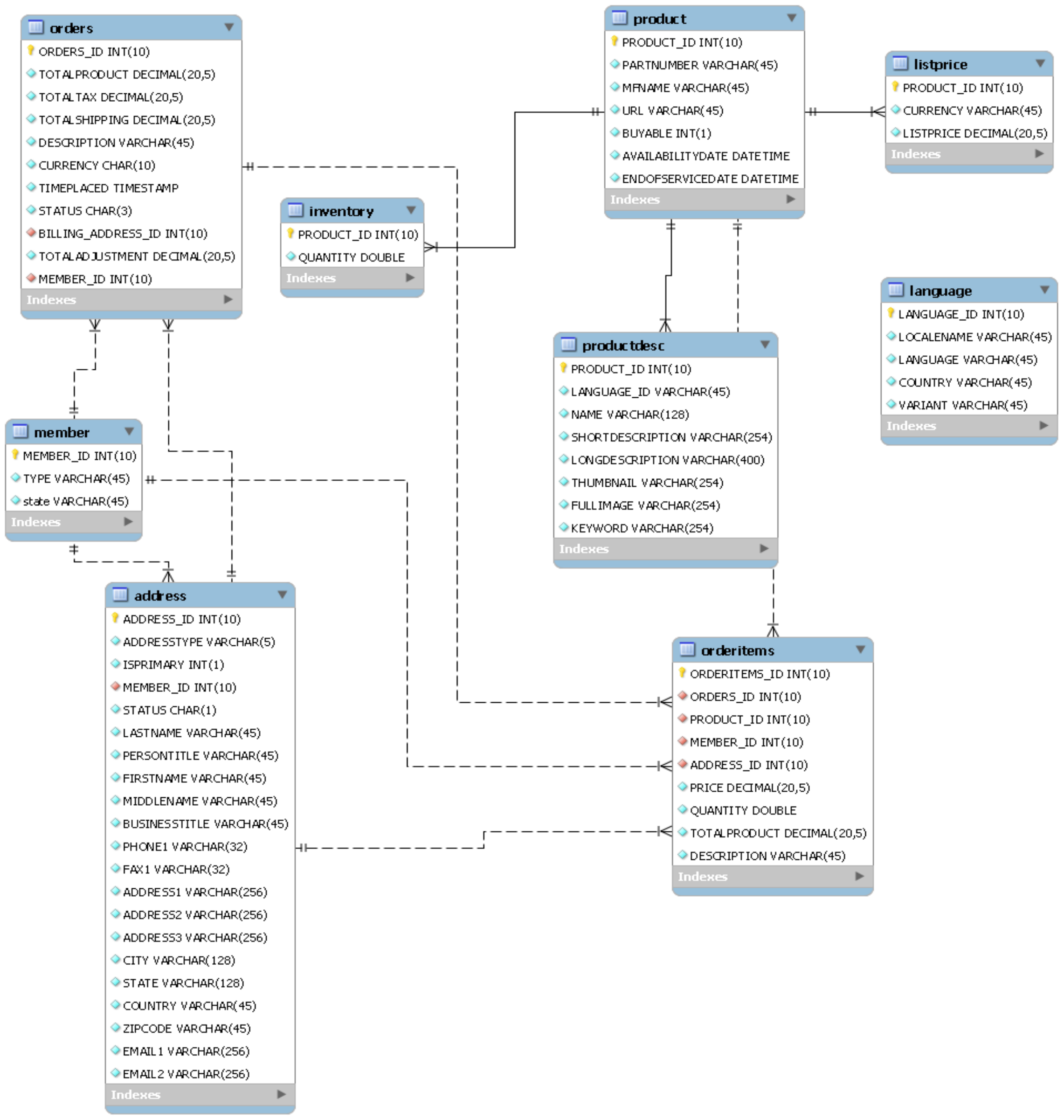


Figure 4 – ONLINE STORE DATABASE SCHEMA

A detailed description of each database table is covered in Appendix A. Worth mentioning that part of the efforts to build a simulator was to build Java classes that use JDBC driver to connect to the DBMS and utilize “insert” SQL statements. These classes would run for one time in order to initialize or populate hundreds of thousands of database records. All these classes are included in a Java package named. “edu.louisville.cs.db.” In this section, we will not go into detail describing each method in these classes, as they are insignificant to overall goal. However, a class diagram summarizing these classes is displayed below.

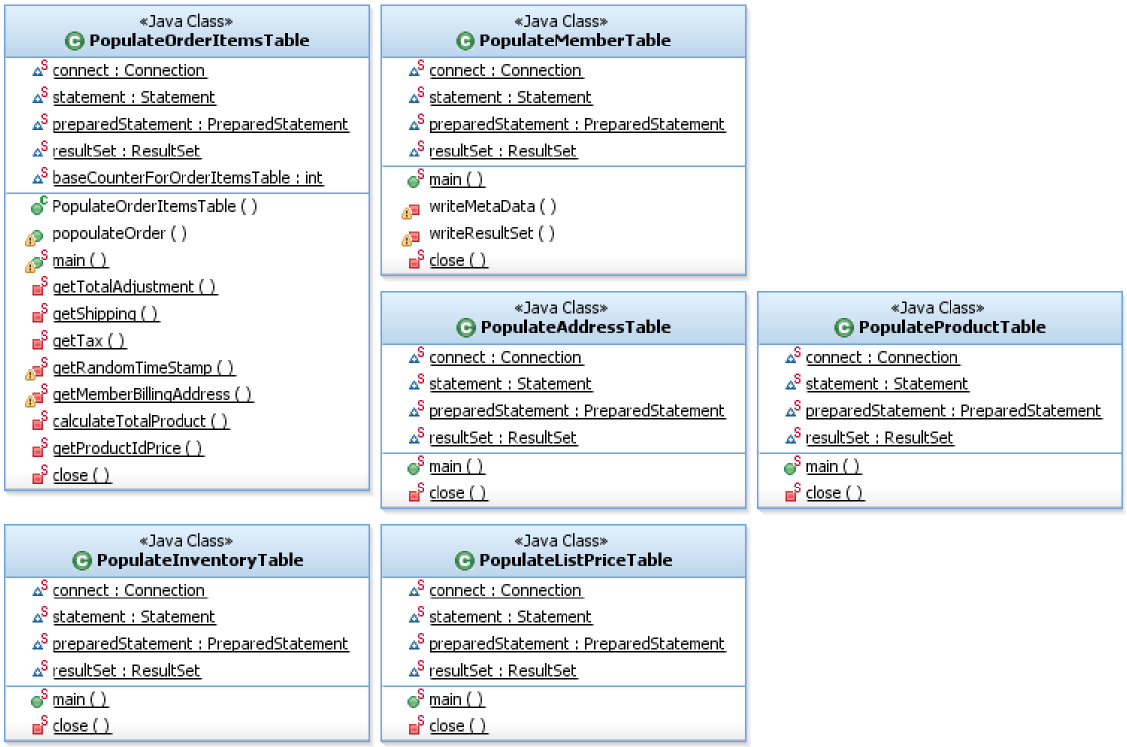


Figure 5 – CLASS DIAGRAM FOR JAVA OBJECT POPULATING DATABASE

2.3. System Specifications and Simulation

This section will cover how to simulate a snapshot of the system performance. We will start first by detailing the platform specification used for this study.

- The system is Windows Server 2008 based.
- Runs on four processors cores
- Has 6144 MB of RAM
- Has 53GB of hard disk
- Has MySQL database V.56
- Has JVM V.1.7

To simulate a demand of single server, we need to consider the application server topology. In real commerce implantation, the production environment consists of one or more application server(s). Each of which has single or multiple server nodes, each node has shared pooled of database connections to be distributed across the different user web sessions. The ecommerce site requires enough JDBC pool connections in order to insure that no threads need to wait for an available database connection. For consideration regarding the user experience, the number of database connections is set to be relative or equal to the number of allowed concurrent active web sessions. In a typical active web session, users would login and become authenticated against the database, search products and retrieve their information from database, and finally proceed to the checkout flow where the entire order transaction will again be recorded in the database. Through the user journey, there is only one database connection through which different database transactions (queries) are executed. In other words, to simulate the demand of an online store hosted on a singled server, the simulation system would utilize multi-concurrent threads. Each of these Java threads represents an active web session. Each thread will establish a new JDBC connection and would run number of queries under each connection.

2.4. Demand Simulation Snapshot and CPU Utilization

To study the demand of the system against the response time and the CPU utilization and in isolation from the heap allocation or the memory utilization, we will start by running 100-concurrent threads in order to record a snapshot of the system performance. In order understand the system behavior that could lead to a bottleneck situation in regard to only the CPU utilization and query response time and in isolation of the memory allocation, each of these 100 parallel threads will use a simple insert database query, to avoid substantial returned result, and hence avoid substantial memory allocation of the JVM heap. Same query will run for certain number under each database connection initialized in each thread. We will record the following performance metrics: number of threads in memory competing for the database resources, the system CPU utilization percentage, response time, start time of each thread, end time of each thread, JVM heap ratio, allocated heap memory in KB, Max Heap Memory in KB, and finally the number of threads served. Each of these metrics will be gathered and recorded toward the end of each thread execution. In addition, data summary will be calculated to include mean, standard deviation, median, maximum, and minimum data points. The data summary will be used later on to normalize the input to the neural network as descried in subsequent sections. The complete result of the snapshot run is shown in the below table.

Threads in the Memory	CPU Utilization	Response Time (MS)	Start Time	End Time	JVM Heap Ratio	Allocated Memory (KB)	Max Memory (KB)	Threads Served
100	55.90%	64.7394	9471.1332	9535.8726	0.00599	1518	253440	1
99	39.60%	98.8712	9471.4200	9570.2912	0.00599	1518	253440	2
98	28.53%	169.4794	9471.6857	9641.1651	0.01052	2667	253440	3
97	23.49%	222.4609	9472.3764	9694.8373	0.01401	3550	253440	4
96	23.83%	230.4046	9471.8583	9702.2629	0.01266	3208	253440	5
95	23.94%	237.8465	9471.2631	9709.1096	0.01250	3167	253440	6
94	22.17%	259.6832	9472.1675	9731.8507	0.01256	3184	253440	7
93	20.75%	289.4151	9472.0572	9761.4723	0.00599	1518	253440	8
92	20.32%	300.1542	9472.0468	9772.2010	0.00164	415	253440	9
91	20.11%	316.4318	9471.7506	9788.1824	0.01297	3288	253440	10
------	------	------	------	------	------	------	------	------
------	------	------	------	------	------	------	------	------
------	------	------	------	------	------	------	------	------
------	------	------	------	------	------	------	------	------
8	13.49%	1060.6872	9471.2123	10531.8995	0.01264	3204	253440	93
7	13.47%	1070.3242	9471.4714	10541.7956	0.01215	3079	253440	94
6	13.50%	1079.3157	9471.1389	10550.4546	0.00272	689	253440	95
5	13.45%	1081.9205	9471.2212	10553.1417	0.00738	1871	253440	96
4	13.47%	1086.5877	9471.3841	10557.9718	0.01087	2755	253440	97
3	13.27%	1085.9374	9472.1406	10558.0780	0.00808	2048	253440	98
2	13.41%	1090.0949	9471.7748	10561.8697	0.01283	3251	253440	99
1	13.26%	1094.3487	9472.2388	10566.5875	0.00599	1518	253440	100

Table 1-CPU Utilization demand simulation snapshot Summary

2.4.1. Demand Simulation Snapshot Run Summary

	CPU Utilization	Response Time	Heap Ratio
Min	13.21%	64.73	0.00164
Max	55.90%	1094.3	0.01742
Stand Dev	5.48%	728.04	0.00486
Average	16.10%	579.544	0.00796
Median	14.49%	579.544	0.00738

Table 2-CPU Utilization demand simulation summary

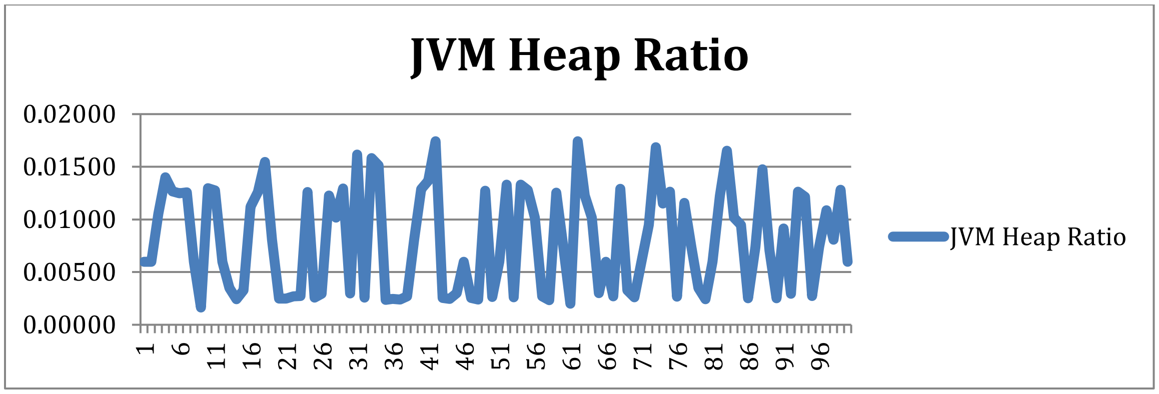


Figure 6 – JVM HEAP RATIO CHART

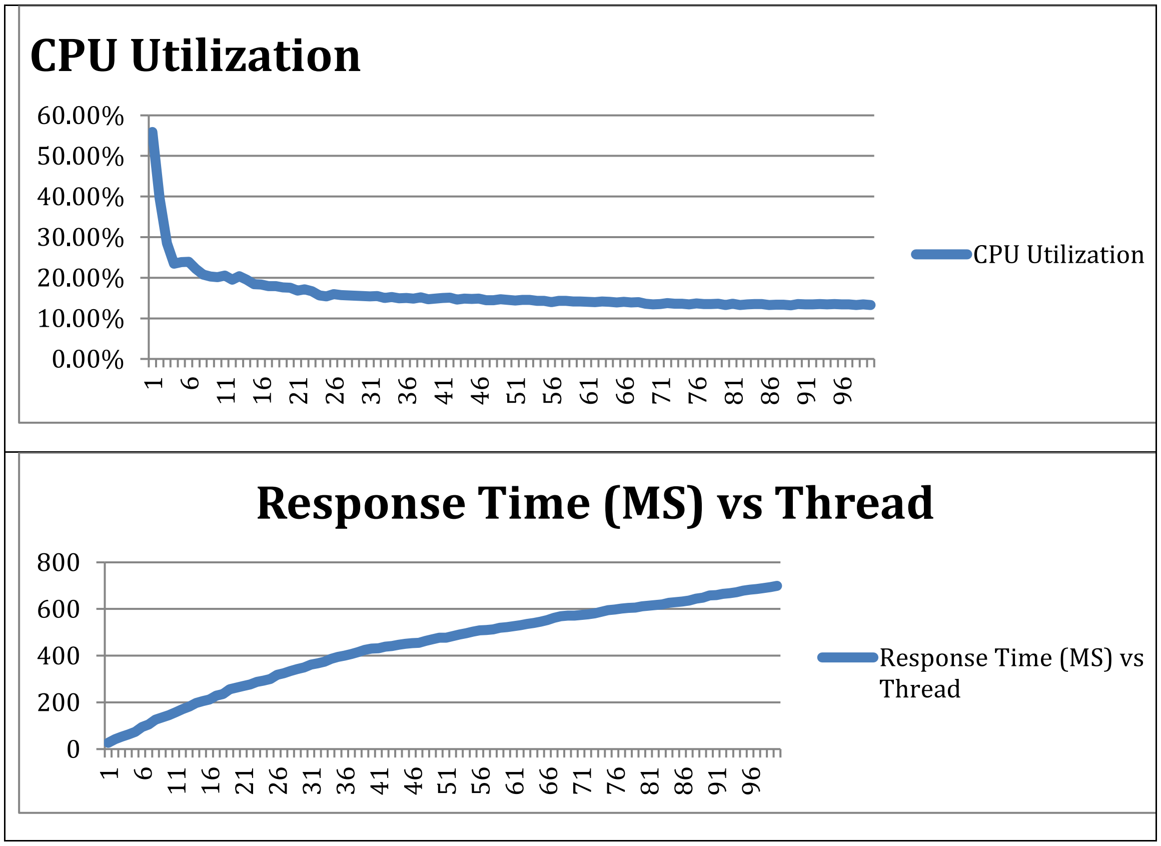


Figure 7 – CPU UTILIZATION AND RESPONSE CHART

From the previous records, we could easily spot a correlation between the number of threads, or web session, and the overall system performance. We can conclude with great degree of confidence that the system performance in terms of CPU utilization improves as the number of threads competing for the system services decreases. In other words, the system could reach a deadlock as more threads are competing for its resources. This fact, as apparent as it is, will justify our future work to rectify the system's bottleneck situation by including a queue component in order to manage or stream down elastically an overwhelming system demand once benchmarked and detected.

2.5. Demand Simulation Snapshot and Memory Utilization

To further study the bottleneck of a system and it is relation to a high memory allocation, or a high JVM heap utilization as in our case, we will run the same 100-concurrent threads in order to record a snapshot of the system performance. Each of these threads will use a search database query, where the returned result is big enough to cause higher object allocation in the heap. Fresh search query will run for certain number under each database connection included in each thread. We will record the same previous performance metrics: number of threads in memory competing for the database resources, CPU utilization, response time, start time of each thread, end time of each thread, JVM heap ration, allocated heap memory in KB, Max Heap Memory in KB, number of threads served. A data summary from average to standard deviation will also be included in a separate table.

Threads in the Memory	CPU Utilization	Response Time (MS)	Start Time	End Time	JVM Heap Ratio	Allocated Memory (KB)	Max Memory (KB)	Threads Served
100	100.00%	74.10382	1593.50065	1667.60447	0.96275	244	253.44	1
99	100.00%	95.75259	1593.64834	1689.40093	0.96275	244	253.44	2
98	100.00%	116.71327	1593.10269	1709.81596	0.76941	195	253.44	3
97	98.63%	134.75497	1593.06380	1727.81877	0.94697	240	253.44	4
96	93.77%	161.04976	1593.43684	1754.48660	0.80887	205	253.44	5
95	91.87%	176.59385	1593.44190	1770.03575	0.91935	233	253.44	6

94	91.02%	203.44151	1593.03835	1796.47986	0.70234	178	253.44	7
93	89.47%	219.6958	1593.60733	1813.30313	0.79309	201	253.44	8
92	88.08%	249.37769	1593.39249	1842.77018	0.78520	199	253.44	9
--------	--------	--------	--------	--------	--------	--------	--------	--------
--------	--------	--------	--------	--------	--------	--------	--------	--------
9	62.13%	1235.13727	1592.82260	2827.95987	0.72601	184	253.44	92
8	62.11%	1243.44796	1593.06401	2836.51197	0.71812	182	253.44	93
7	61.99%	1248.93478	1593.12268	2842.05746	0.79309	201	253.44	94
6	61.85%	1257.3786	1593.75104	2851.12964	0.59580	151	253.44	95
5	62.00%	1277.89931	1593.09050	2870.98981	0.82071	208	253.44	96
4	62.01%	1279.04924	1593.05784	2872.10708	0.61158	155	253.44	97
3	61.97%	1280.00535	1593.45456	2873.45991	0.69839	177	253.44	98
2	61.89%	1283.23441	1593.83750	2877.07191	0.67077	170	253.44	99
1	61.99%	1288.75338	1593.02577	2881.77915	0.69839	177	253.44	100

Table 3-Memory Utilization demand snapshot summary

2.5.1. Demand Simulation Snapshot Run Summary

	CPU Utilization	Response Time	Heap Ratio
Min	61.85%	74.10	0.58
Max	100.00%	1288.75	0.96
Stand Dev	10.04%	858.88	0.090417725
Average	69.46%	681.43	0.72
Median	65.06%	681.42	0.71

Table 4-Memory Utilization demand simulation summary

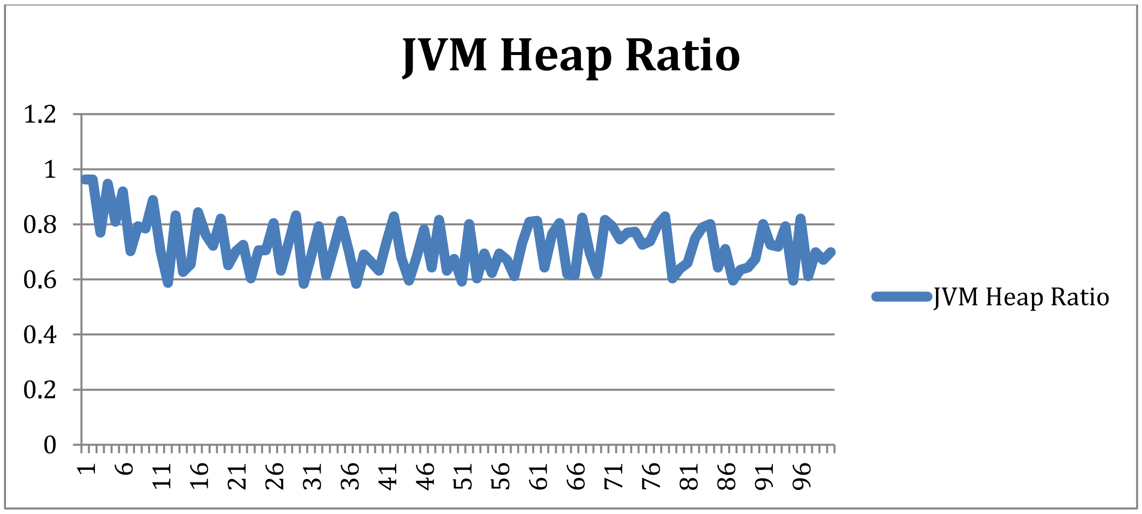
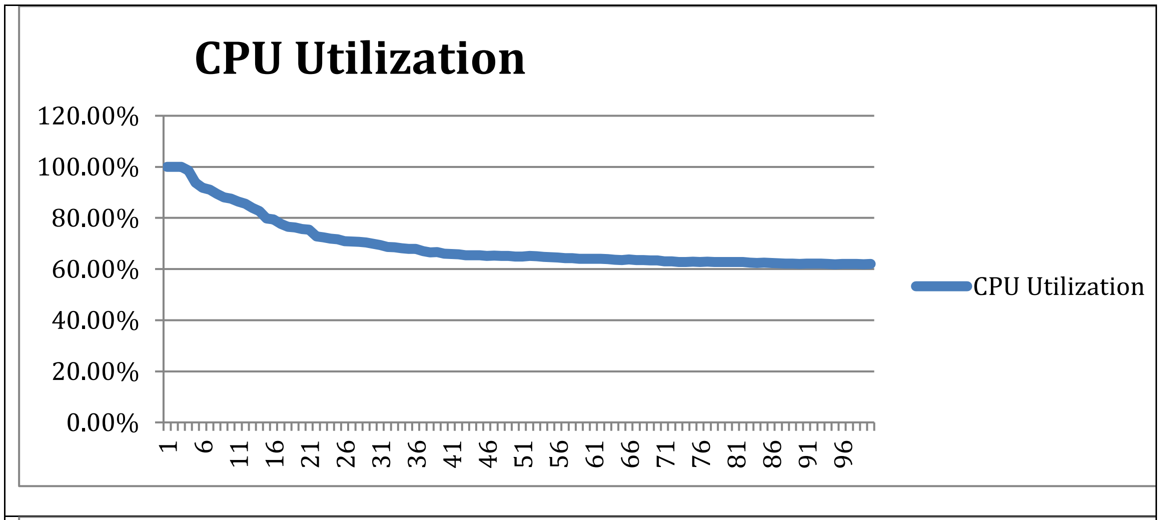


Figure 8 – JVM HEAP RATIO CHART



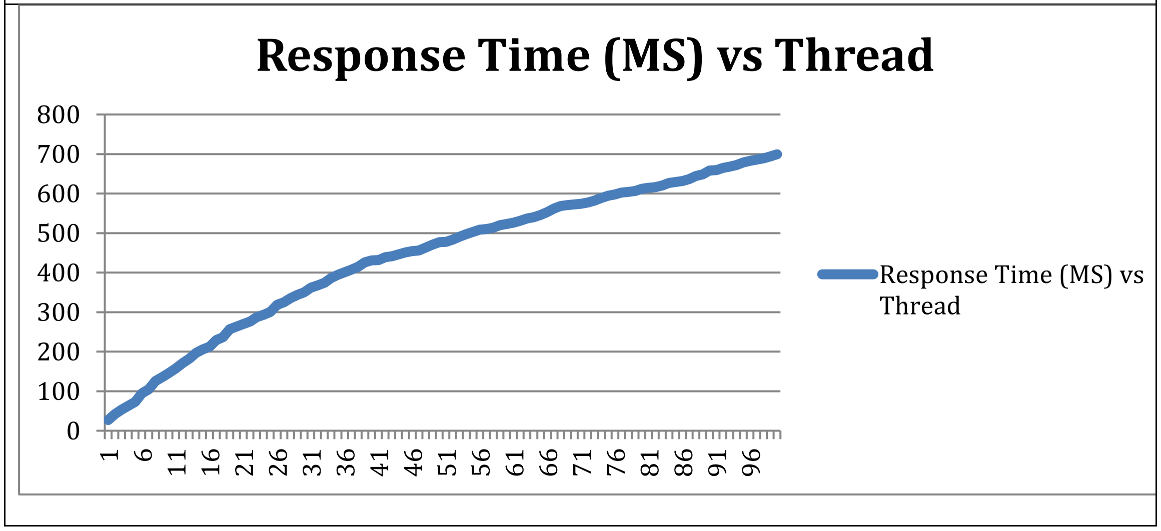


Figure 9 – CPU UTILIZATION AND RESPONSE CHARTS

From the previous results it shows that the high object allocation in the JVM heap is not as significant factor for the system to reach a bottleneck situation. This could be contributed to the fact that the JVM heap under this study only occupy smaller space of the system RAM or the hard disk storage and hence saturating the heap size is less significant on causing a system bottleneck. However, it is evident that larger object allocations lead to higher the CPU utilization. Also in real application a large heap size could lead to an out of memory outages. The result from this section and the previous section will justify the use of queuing system to rectify an overwhelming system demand and will also justify the fact that we monitor the heap utilization as a performance metric and added as a parameter to the learning engine to detect a bottleneck situation when it reaches more than 90%.

2.6. Simulation of real ecommerce user sessions

To simulate a real-life user web session on an ecommerce store in respect to the database load, we collected a real production data from a Business-to-Business (B2B) ecommerce store to understand the type of queries rendered through the user journey on the online store. We first started by studying the average user requests per day. The below data table was collected from a newly published "HP2B" online ecommerce store. The table below is usually collected on daily basis by a monitoring tool for the purpose of showing the page load time as a mean of collecting daily performance snapshot of the production system and to understand where the user may face potential issue while processing HTTP requests. The "Load Time(s)" column is an average of all the requests targeted the system on that specific day. The "Beacons" column shows the number of user requesting URL's for the associated webpages shown in the "Page Group" table.

Row	Page Group	Load Time (s)	Beacons
1	Home Page	7.15	4,164
2	Product Detail Page	10.57	2,476
3	Login Page	3.63	1,347
4	Search Page	2.63	765
5	Others	6.08	663
6	Product Listing Page	6.3	480
7	View Cart Page	4.67	372
8	Checkout Page	4.97	127

Table 5 -HP2B page demands

To better understand the above table, another pivot table and chart, shown below, are derived from the above data table, to plot the relation between the page type and the user request. From the below derived table we found that during the user journey from the login page to the checkout page, most users spend the majority of their web session times on the home page, interacting with their account details, more than they spend time buying or submitting an order.

Row	Page Group	Percentage
1	Home Page	40.06%
2	Product Detail Page	23.82%
3	Login Page	12.96%
4	Search Page	7.36%
5	Others	6.38%
6	Product Listing Page	4.62%
7	View Cart Page	3.58%
8	Checkout Page	1.22%

Table 6-HP2B page demand percentage per user

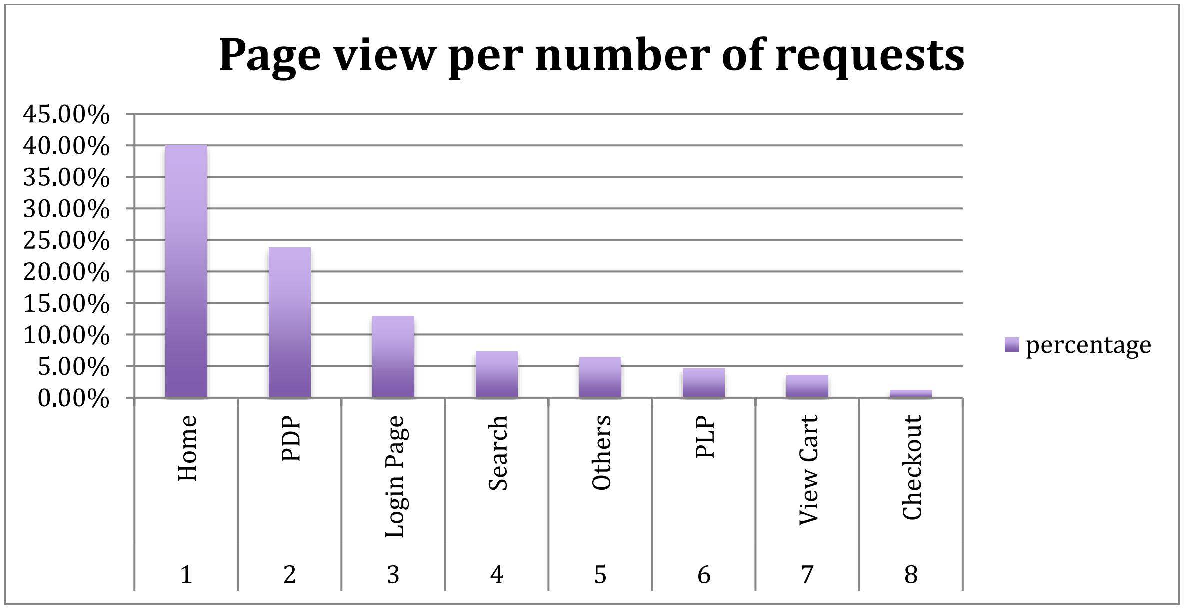


Figure 10 – *HP2B SITE PAGE VIEW STATS*

To map these results to the system under study, we draw a relation between each page and its association with the database tables. The home page and the login page account for around 53% of the web session load. And both are associated with the member and address tables. Product Detail Page (PDP) accounts for 25% of the web session load and is associated with the product and inventory tables. The search and the product listing page (PLP) accounts of about 10% of the session load and again are associated with the product and inventory tables. The view cart and the checkout pages account for 5% of the session load and is associated with the "order" table and so on. To simulate the database demand during the user web session, we will query the database, in each Java thread, number of times relative to the percentages of the user web session load calculated above and we will run a new snapshot as shown below. We expect the new reading to be in alignment of the previous readings specially those from the very last section with different variation.

Threads in the Memory	CPU Utilization	Response Time (MS)	Start Time	End Time	JVM Heap Ratio	Allocated Memory (KB)	Max Memory (KB)	Threads Served

100	100.00%	26.98033	41694.09321	41721.07354	23	60769	253440	1
99	93.33%	42.12236	41694.03538	41736.15774	16	41377	253440	2
98	86.17%	53.40489	41694.05878	41747.46367	23	59594	253440	3
97	83.04%	62.93324	41694.09243	41757.02567	16	41799	253440	4
96	80.77%	73.39834	41694.08217	41767.48051	23	59115	253440	5
95	79.92%	94.86686	41693.8797	41788.74656	17	43377	253440	6
94	79.19%	104.99724	41693.81618	41798.81342	20	50864	253440	7
93	76.78%	125.36705	41694.14626	41819.51331	19	49942	253440	8
92	77.11%	135.34169	41694.13327	41829.47496	25	64049	253440	9
91	76.70%	145.82959	41694.05341	41839.883	19	50449	253440	10
---------	---------	---------	---------	---------	---------	---------	---------	---------
---------	---------	---------	---------	---------	---------	---------	---------	---------
8	70.20%	667.73692	41693.9845	42361.72142	25	65244	253440	93
7	70.15%	672.01238	41693.97993	42365.99231	16	41288	253440	94
6	70.29%	678.72642	41694.01868	42372.7451	20	51975	253440	95
5	70.34%	682.65985	41693.89277	42376.55262	22	57195	253440	96
4	70.27%	685.50448	41693.89904	42379.40352	19	50501	253440	97
3	70.19%	689.25683	41693.98959	42383.24642	23	60560	253440	98
2	70.07%	693.91156	41694.00244	42387.914	23	59711	253440	99
1	69.94%	699.2557	41694.06585	42393.32155	25	63494	253440	100

Table 7 -HP2B simulations run summary

2.6.1. Demand Simulation Snapshot Run Summary

	CPU Utilization	Response Time	Heap Ratio
Min	69.94%	26.98	15.00000
Max	100.00%	699.255	25.00000
Stand Dev	4.42%	475.37	2.96573
Average	72.61%	363.11	20.18000
Median	71.40%	363.11	20.00000

Table 8 -HP2B simulations run summary

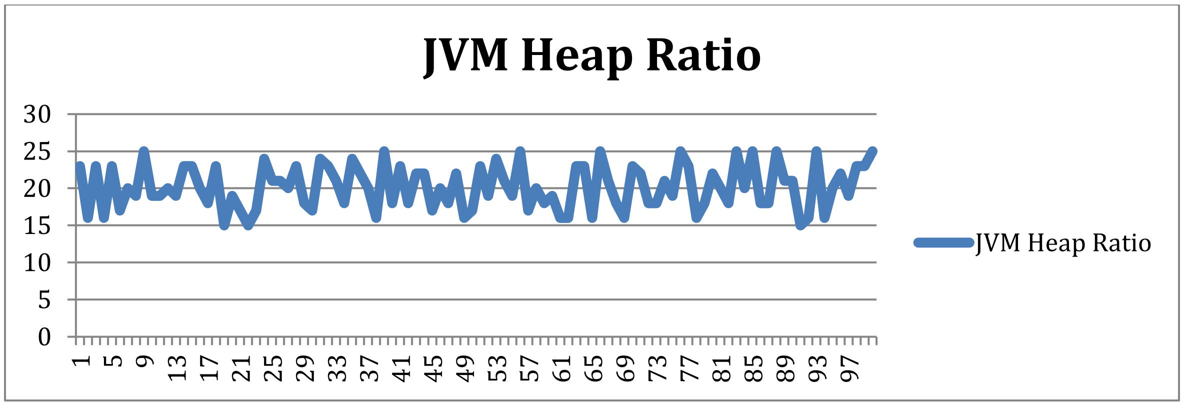


Figure 11 – JVM HEAP RATIO

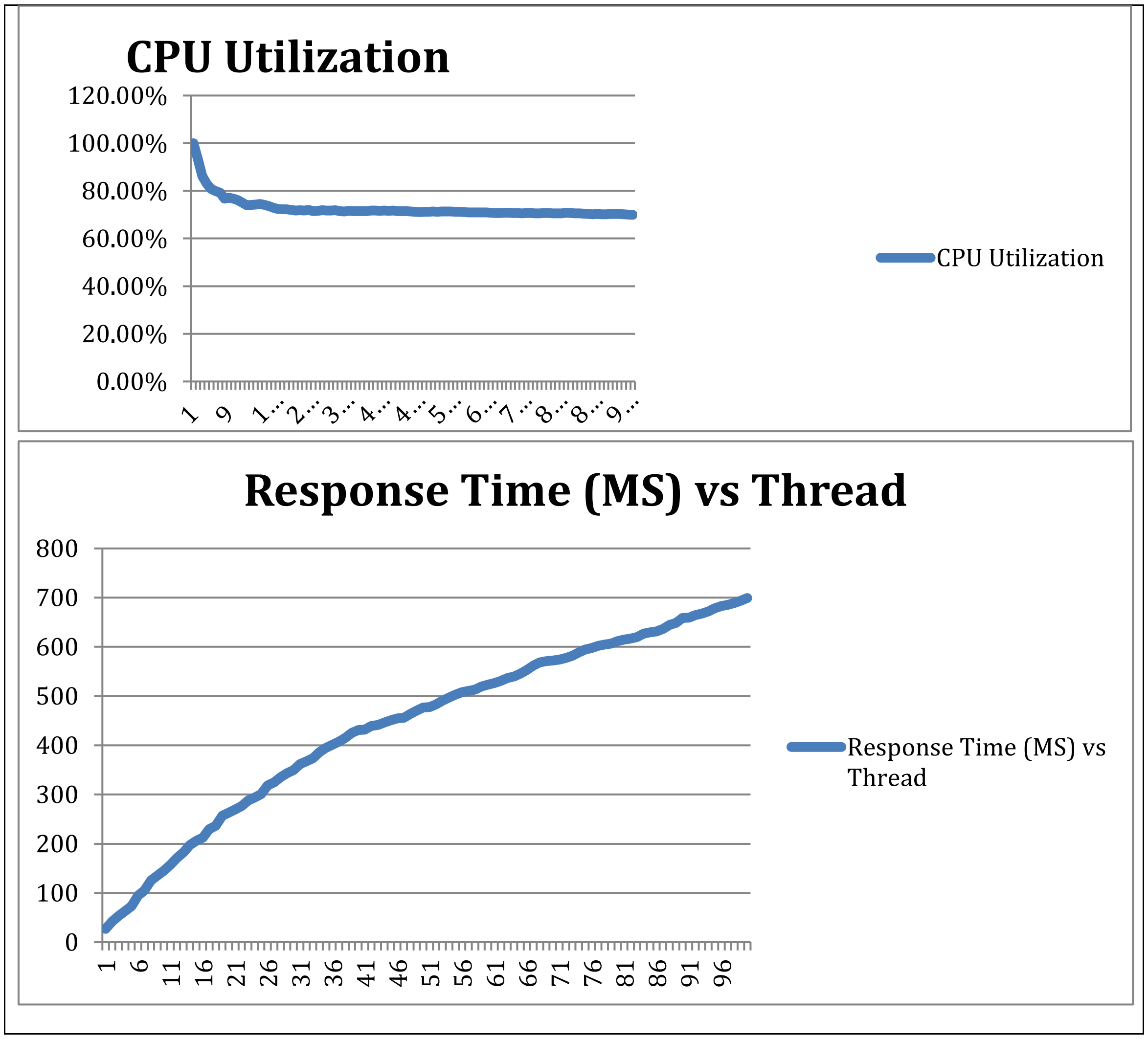


Figure 12 – CPU UTILIZATION AND RESPONSE TIME CHARTS

2.7. Summary

In this chapter, we took three snapshots of the system to understand the correlation between certain parameters and overall system performance. We also concluded that there is a direct relation or dependency between the number of active resources competing for the processer time, the heap utilization or the memory allocation, on one hand, and the CPU utilization and the response time, on the other hand. The results from these snapshots will constitute all of our future work in the next chapters.

Cl!t¢9w 3 LEARNING ENGINE

This engine monitors different system metrics and learns when to benchmark and detect a system bottleneck. The learning engine monitors the performance metrics of the requestor system, where the multi-threading Java application is running, and the provider system where the common resources run, i.e. the DBMS in our case. The Learning Engine will *continuously* monitor performance metrics such memory usage and CPU utilizations and will be able to *adapt* to any changes in the monitored system (the provider system) to accurately identify a bottleneck situation. Once this monitoring model is benchmarked, it will assess and quantify the relation between performance metrics and the occurrences of a bottleneck.

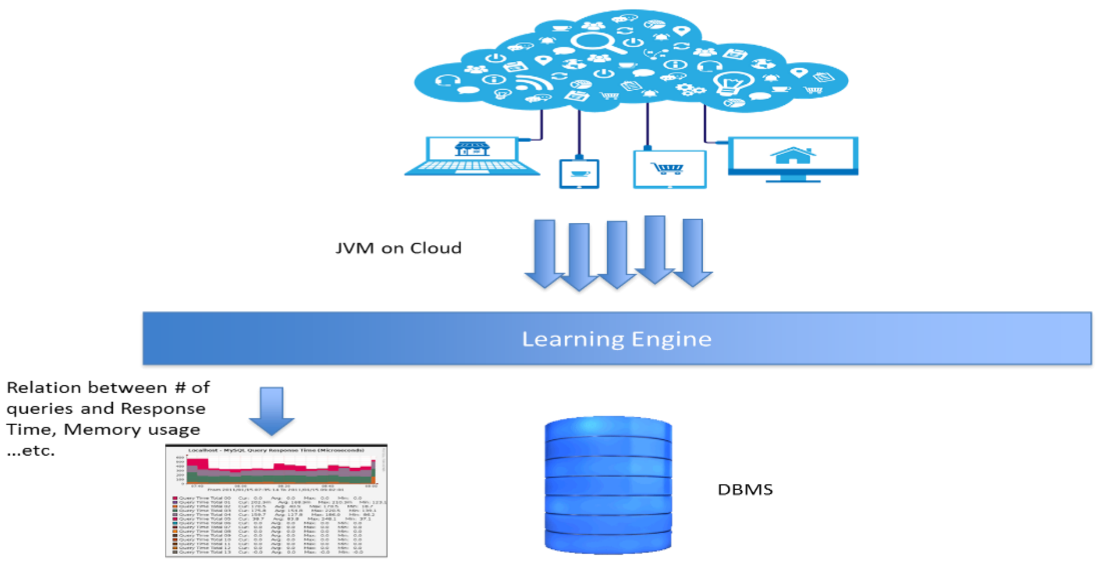

JVM on Cloud
Learning Engine
Relation between # of queries and Response Time, Memory usage ...etc.
DBMS

Figure 13 – LEARNING ENGINE

To achieve this goal, continuous learning and adaptability, learning engine will use a lightweight implementation of Artificial Neural Network (ANN). ANN will assign random interconnection weights for the performance metrics. i.e. response time, CPU usage, and memory utilizations, by applying a set of training or learning samples. The final effects of a learning process are tuned parameters of a network the will help point to a bottleneck situation. Moreover, it will be able to figure out the right weights of the performance input parameters in cases where an ambiguity surround whichever performance metrics has the most effect on the system performance. The lightweight is also a requirement for the Cognitive Performance Application not to constitute a performance burden on the host system.

3.1. Neural Network Type

The neural network implemented in our application is of a type *feedforward*. Meaning the flow of the signal is on one direction. The input signals are fed into the input layer and then forward to the next layer. The network learning is also a *supervised* because the objective is to map input data (X's) to an output (Y) through a learning function f.

F: X → Y. where Y data acts as the supervisor. [7]

The below diagram shows the neural network implemented.

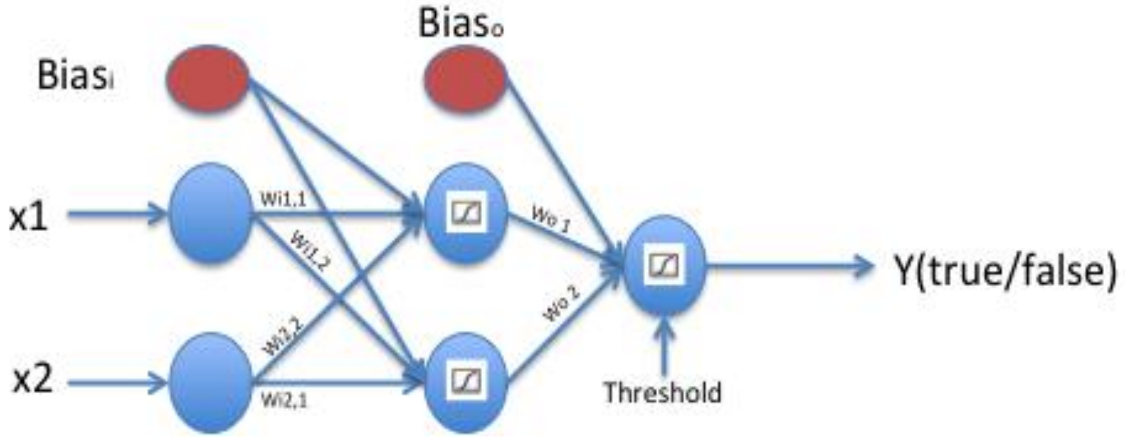


Figure 14 – IMPLEMENTED NEURAL NETWORK

In this neural network there is one input layer and one hidden layer following *monolayer* network architecture for simplicity and outmost performance.

3.2. Neural Network Input

The input data in our case will be the CPU utilization ranging percentage and the JVM heap ratio. Both inputs normalized and are ranging from 0 to 1. The output on the other hand will be either 0 or 1 as a classification of the system bottleneck, where 1 is true and 0 is false. Training data will be provided for the neural network at the time of initialization as in two-dimension array shown below.

```
float[][] trainingData = new float[][] {
{0.70f, 0.73f }, {1.0f, 0.81f }, {0.1f, 0.86f },
{0.90f, 0.95f }, {1.0f, 0.45f }, {0.2f, 0.70f },
{0.95f, 0.51f }, {0.6f, 0.89f }, {0.3f, 0.79f },
{0.93f, 0.51f }, {0.5f, 0.89f }, {0.4f, 0.79f },
{0.91f, 0.71f }, {0.89f, 1.0f }, {0.9f, 0.39f },
{0.92f, 0.1f },  {0.93f, 1.0f }, {0.94f, 0.4f },
{0.88f, 0.37f }, {0.87f, 0.87f },{0.86f, 0.76f}, };
```

The actual results of these learning input will be also provided as one dimensional array of zero’s and one’s as shown below.

```
int[] actuals = new int[]{
                        0,1,0, 1,1,0,
                        1,0,0, 1,0,0,
                        1,0,1, 1,1,1, 0,0,0};
```

Worth mentioning, that the input data in this above form satisfies both the vectorization and normalization prerequisites of artificial neural network input data. [8] Below code shows how input and output weights, and bias are randomly initialized.

```
   private void initWeights(int neurons, int dimension) throws ZeroNeuronsException,
ZeroInputDimensionException {
        if (neurons == 0)
            throw new ZeroNeuronsException();
        if (dimension == 0)
            throw new ZeroInputDimensionException();

        for (int i = 0;i<neurons;i++){
            this.bias[i] = Utils.randFloat(-0.5f, 0.5f);
            this.outWeights[i] = Utils.randFloat(-0.5f, 0.5f);
            for (int j = 0;j<dimension;j++) {
                this.inputWeights[j][i] = Utils.randFloat(-0.5f, 0.5f);
            }
        }
    }
```

3.3. Learning procedure

The below flow diagram summarizes the neural network systematic learning process.

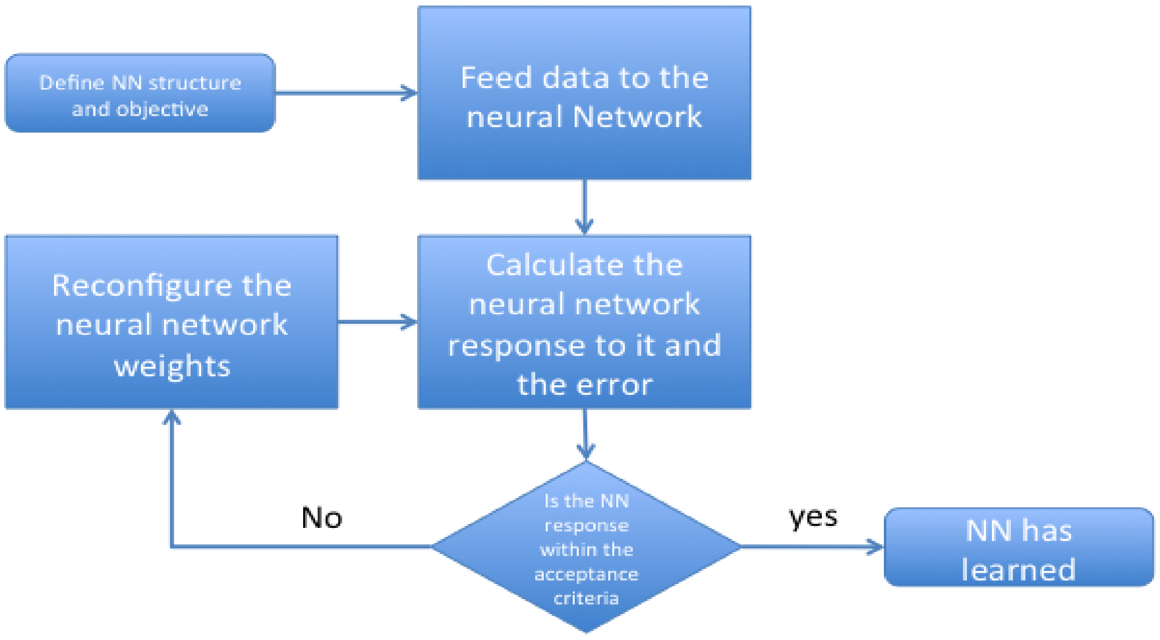

Define NN structure and objective
Feed data to the neural Network
Calculate the neural network response to it and the error
Reconfigure the neural network weights
Is the NN response within the acceptance criteria
No
yes
NN has learned

Figure 15 – NEURAL NETWORK SYSTEMATIC LEARNING

Learning procedure is iterative process and is controlled or stopped when reaching the maximum number of iterations (**private int maxEpochs= 10000;).** Another important parameter is the learning rate **(float learningRate = 0.05f;),** which dictates how strongly the neural network would vary in the weights' hyperspace. [8] ANN will check it is response or result against the input data and will calculate how far off the results **(float error = actual - fOut;).** Then it will use the "*Delta rule*", which is an algorithm based on the gradient descent method to account for nonlinearity, to determine how the input and output weights (W) and biases will be updated. The complete method implementation is shown below.

```
private void initLearn(float actual, float fOut, float[] expectedOutput, float[]
outWeights, float[][] inputWeights, float[] bias, float bOut, int neurons, float[]
input, int dimension) {
        float error = actual - fOut;
        float learningRate = 0.05f;
        float dv;
        float[] dw = new float[neurons];
        float[][] dwi = new float[dimension][neurons];
        float[] dbi = new float[neurons];
        float[] db = new float[neurons];


        // Modify out weights
        dv = fOut * (1-fOut) * error;
        for (int i = 0;i<neurons;i++){
            this.outWeights[i] = outWeights[i] + learningRate * dv * expectedOutput[i];
        }

        // Modify out bias out
        float dbOut = learningRate * dv * 1;
        this.bOut = (bOut + dbOut);

        // Modify input weights
        for (int i = 0;i<neurons;i++){
            dw[i] = expectedOutput[i] * (1 - expectedOutput[i]) * outWeights[i] * dv;
            for (int j = 0;j<dimension;j++){
                dwi[j][i] = learningRate * dw[i] * input[j];
                this.inputWeights[j][i] = inputWeights[j][i] + dwi[j][i];
            }
        }

        // Modify input bias
        for (int i = 0;i<neurons;i++){
            dbi[i] = expectedOutput[i] * (1 - expectedOutput[i]) * outWeights[i] * dv;
            db[i] = learningRate * dbi[i] * 1;
            this.bias[i] = bias[i] + db[i];
        }
 }
```

3.4. Sigmoid Activation Function

The activation function calculates a “weighted sum” of its input, adds a bias as below

$$\Sigma \textbf{ (weight * input) + bias}$$

This is implemented as shown below

```
        private float calculateFOut(float[] x){
                for (int i = 0;i<neurons;i++){
                    float sum = 0;
                    for (int j=0; j<dimension; j++){
                        sum = sum + (x[j] * wWeights[j][i]);
                    }
                    this.fOutArray[i] = ActivationFunction.activate(sum + bias[i]);
                }

                this.fOut = 0;
                for (int i = 0;i<neurons;i++){
                    this.fOut += fOutArray[i] * vWeights[i];
                }

                return ActivationFunction.activate(fOut + bOut);
        }
```

And then decides whether it should be "activated" or not using the output of the following equation:

$$\boldsymbol{A = \frac{1}{1 + e^{-x}}}$$

, which is implemented as shown below

```
        public class SigmoidFunction implements ActiviationFunction {
            @Override
            public float activate(float value) {
                return (float)(1/(1+Math.exp(-value)));
            }}
```

As the outcome of classification problem is binary, we also added "parser" to convert the output to only 1 if the result is above (0.5) and zero otherwise as shown below.

```
@Override
    public Integer parseResult(float result) {
        return (result < 0.5) ? 0 : 1;
    }
```

3.5. Neural Network Classification Method

Once the neural network has learned, or the final values of the weights and bias have been declared, we surround the method descried earlier "calculateFOut" with another wrapper to restrict the output to the only zero's and ones' as shown below.

```
@Override
    public Integer parseResult(float result) {
        return (result < 0.5) ? 0 : 1; }
```

For simplicity we also wrap the previous method with another wrapper to restrict the output to true and false corresponding to the ones and zeros concluded in the previous method. The true indicates a “deadlock” classification and false otherwise.

```
public boolean classify(double CPUUlization, double Heap) {
        float[] valueToPredict = new float[] {(float)CPUUlization, (float)Heap};

        if (result.classifyValue(valueToPredict)==1)
               return true;
        else return false;
    }
```

3.6. Neural Network Kernel

The below code is the heart of the learning engine. It initializes the neural network, set the number of neuron and the number of the hidden layers, and the activation method. It also randomly assigns values for its input and output weights and bias. Then loops number of times relative to the number of epochs parameter and calls both analyzer and the learner classes. The “analyzer” sums the weights and the input values, and then calculates the expected output values using the activation method. The “learner” calculates the error by subtracting the actual output from the expected output and updates the weights and bias using the delta rule. The code is shown below.

```
Public class NeuralNetworkThread implements Runnable {
        @Override
        public void run() {
            float quadraticError = 0;
            float MSE = 0;
            float[] f;
            int success = 0;
            for (int i = 0; i<maxEpochs; i++) {
                success = 0;
                for (int z = 0; z<nElements; z++) {
                    analyzer = new Analyzer (getRowElements(z), inputWeights, bias,
                               outWeights, bOut, neurons, activateFunction, dimension);
                    f = analyzer.getFOutArray();
                    fOut = analyzer.getFOut();
                    learner = new Learner (outputs[z], fOut, f, outWeights,
inputWeights,
                              bias, bOut, neurons, getRowElements(z), dimension);
                    outWeights = learner.getVWeights();
                    inputWeights = learner.getWWeights();
                    bias = learner.getBias();
                    bOut = learner.getBOut();
                    success = resultParser.countSuccesses(success, fOut, outputs[z]);
                    quadraticError += Math.pow(((outputs[z] - fOut)), 2);
                }
                quadraticError *= 0.5f;
            }

            float successPercentage = (success / (float)nElements) * 100;
            result = new Result(analyzer, resultParser, successPercentage,
quadraticError);
        }}
```

3.7. Mean Squared error

To show the how accurate the neural network becomes, we plotted the below diagram using the MSE data over the complete number of iterations. It is easy to note that MSE decreases as the number of epoch's increases to almost near a zero value.

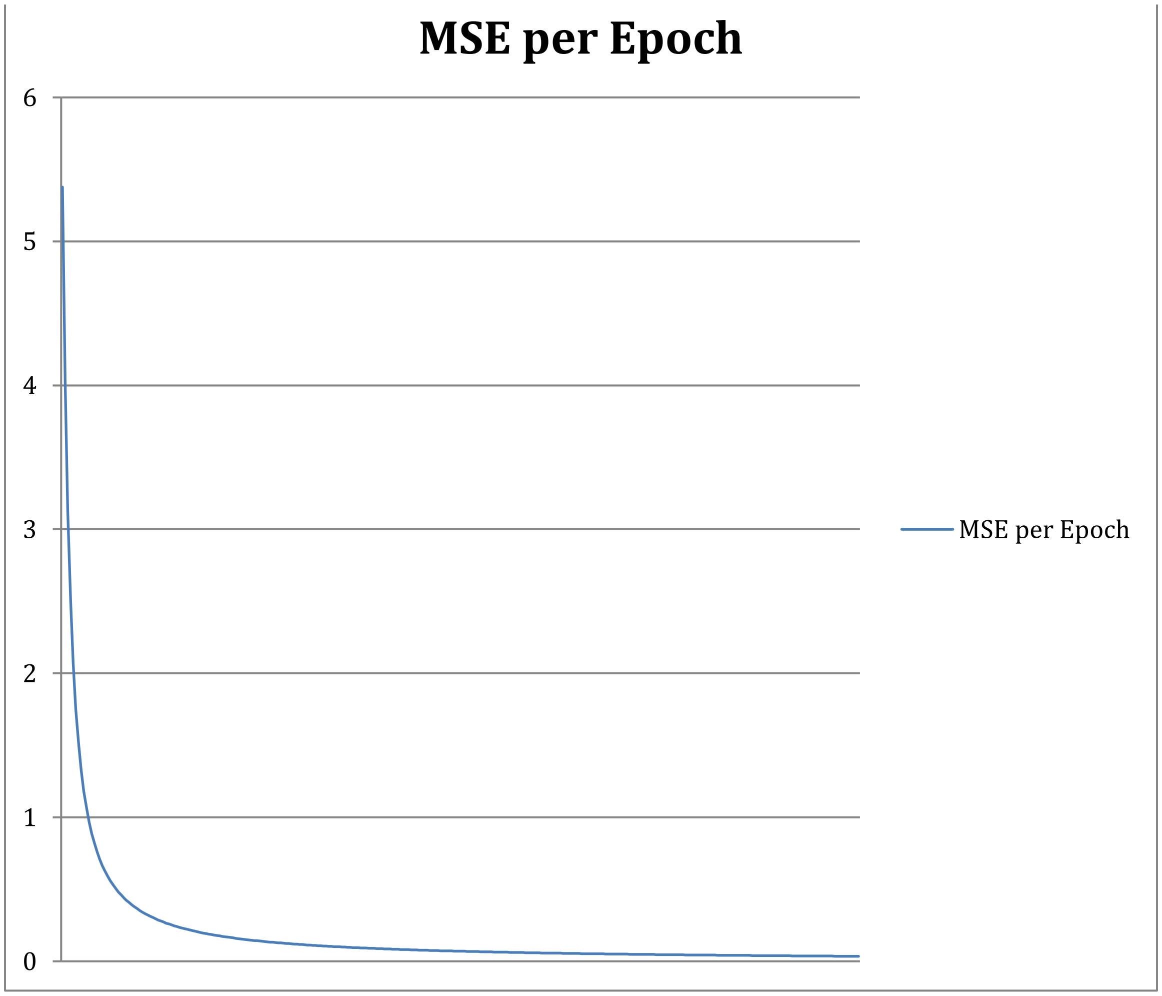


Figure 16 – MSE per Epoch

3.8. Learning Engine Class Diagram

The below graph a UML class diagram summarizing the structure of the learning engine.

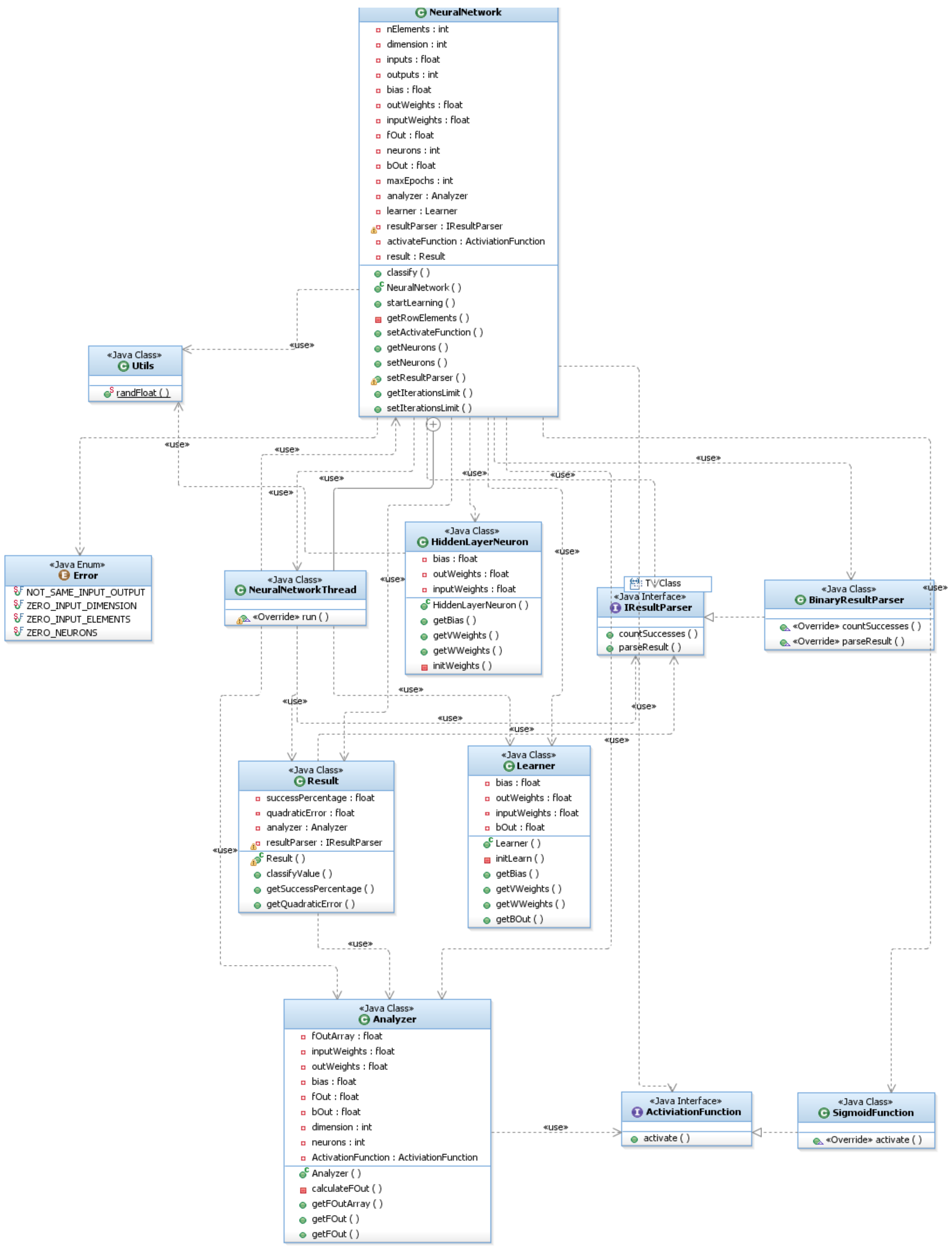


Figure 17 – LEARNING ENGINE CLASS DIAGRAM

C l ! t¢9w 4 ANALYTICS ENGINE

The second component of the Cognitive Performance Application (CPA) is the Analytics engine. The Analytics Engine is a core component of the system that will act primarily as an early alert mechanism that will predict the situation leading to a system deadlock. This component will work continuously in the backend and will keep autonomously correcting its predictions results with the ability to adapt to the different changes in the system specifications. The system will collect a library of factors, which feed through the application in order to establish a pattern of behavior pertaining to the overall performance. At this point, the application usually depends on a human interference for configuring and integrating the feeds of this library of factors into the applications. To establish such a pattern, the input of domain experts usually are needed to decide on the input parameters of the system performance predictive model in regard to forecasting the online commerce store future demand. In real world the demand of a particular eCommerce store could be a subject of multiple correlated relations with metrics that can be collected externally or internally from:

- Current user on the store
- Commerce stores database, such as the trending of daily average of order transactions number
- Weather company, such as feeds of daily temperatures, or the mark and end of weather seasons
- Social media buzz, such as twitter
    - We can also extract other factors such as political stability, social rest…and other materics to predict global demand.

- ...etc.

The previously mentioned feeds can be integrated in many ways such as be implementing a scheduled service-oriented architecture (SOA) calls, where services are provided to the other components through a communication protocol over a network, or alternatively through an Extract, Transform, and Load (ETL) architecture. The architecture for the CPA will be flexible to be extended to different patterns of feeds i.e. either SOA or ETL.

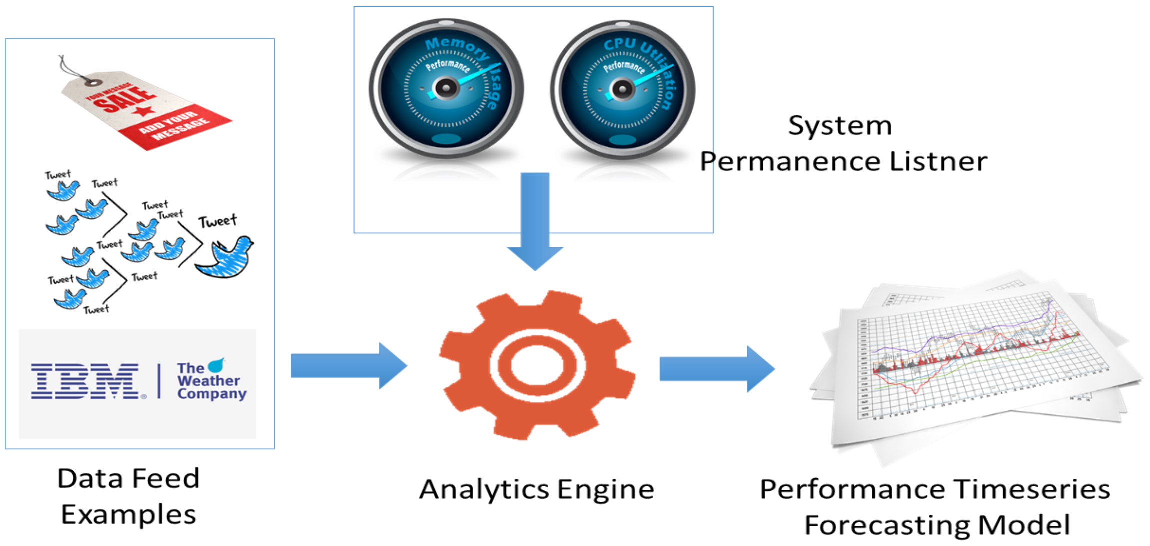

Memory Usage
CPU Utilization
System Permanence Listner
Data Feed Examples
Analytics Engine
Performance Timeseries Forecasting Model

Figure 18 – ONE FLAVOR OF THE ANALYTICS ENGINE

In our implementation, we will use the internal user load to predict the future demand in terms of the CPU utilization. If the future CPU utilization if above certain threshold the analytics engine will trigger a signal, in combination of the learning engine signal, to the queue system to take the necessary rectifying methods.

4.1. Factory Design Pattern

Learning engine will be based on a Factory design pattern. Factory design pattern is used to create objects or Class in Java and it provides loose coupling and high cohesion. Factory pattern encapsulate object creation logic which makes it easy to change it later when you change how object gets created or you can even introduce new object with just change in one class. The obvious reason for this architecture decision is that we will have different analytics models that we will choose among them to initialize our learning engine. These analytics models are

1- Linear Regression Model
2- Multiple Linear Regression Model
3- Polynomial Linear Regression Model

In our learning engine we will implement a parent interface for the previous sub models and we use declaration method to initialize the best fit model after processing the input data as shown in the code below.

```
PredictingModel forecaster = Predictor.getBestForecast( observedData );
```

Using the factory pattern will enable the analytic engine to choose among the different predictive models. Likewise, will make it a "scalable" to add new patterns in the future

4.2. Best Fit Model Criteria

The choice among the predictive models will be based on how a good fit they are determined by the following calculation criteria

- Akaike Information Criteria (AIC)
- Arithmetic mean of the errors (bias)
- Mean Absolute Deviation (MAD)
- Mean Absolute Percentage Error (MAPE)
- Mean square of the errors (MSE)
- Sum of Absolute Errors (SAE)

The calculation and the implementation of earlier analytical criteria are shown below

```
    /**
     * A method to calculate the various accuracy indicators when
     * applying the given DataSet to the current forecasting model.
    **/
protected void calculateAccuracyIndicators( DataSet dataSet )
    {
        // Note that the model has been initialized
        initialized = true;

        // Reset various helper summations
        double sumErr = 0.0;
        double sumAbsErr = 0.0;
        double sumAbsPercentErr = 0.0;
        double sumErrSquared = 0.0;

        // Obtain the forecast values for this model
        DataSet forecastValues = new DataSet( dataSet );
        predict( forecastValues );

        // Calculate the Sum of the Absolute Errors
        Iterator<DataPoint> it = dataSet.iterator();
        Iterator<DataPoint> itForecast = forecastValues.iterator();
        while ( it.hasNext() )
            {
                // Get next data point
                DataPoint dp = it.next();
                double x = dp.getDependentValue();

                // Get next forecast value
                DataPoint dpForecast = itForecast.next();
                double forecastValue = dpForecast.getDependentValue();

                // Calculate error in forecast, and update sums appropriately
                double error = forecastValue - x;
                sumErr += error;
                sumAbsErr += Math.abs( error );
                sumAbsPercentErr += Math.abs( error / x );
                sumErrSquared += error*error;
            }

        // Initialize the accuracy indicators
        int n = dataSet.size();
        int p = getNumberOfPredictors();

        accuracyIndicators.setAIC( n*Math.log(2*Math.PI)
                + Math.log(sumErrSquared/n)
                + 2 * ( p+2 ) );
        accuracyIndicators.setBias( sumErr / n );
        accuracyIndicators.setMAD( sumAbsErr / n );
        accuracyIndicators.setMAPE( sumAbsPercentErr / n );
        accuracyIndicators.setMSE( sumErrSquared / n );
        accuracyIndicators.setSAE( sumAbsErr );
    }
}
```

4.3. Predictive Analytics Models

Once the feeds and the means of integration are established, the predictive analytic model needs to be activated to forecast the future demand. Again, there are many techniques or methods that can be used to establish such model. Many machine learning techniques can be utilized to establish the system demand predictive model such as regression, clustering, classifications, decision trees/rules, k-nearest neighbors…etc [9]. This document will focus only on three solid predictive models. These analytics models are

- Linear Regression Model
- Multiple Linear Regression Model
- Polynomial Linear Regression Model

The detail of each will be discussed in the subsequent sections.

4.3.1. Linear Regression Model

An example of these techniques is a regression predicative model. Linear regression was developed in the field of statistics and is studied as a model for understanding the relationship between input and output numerical variables, and later it has been brought on to be a branch machine learning. The idea of regression model is to find the best-fitting line among historic data points. [10] This will be done through calculating the slope and the intercept as in the below equation

Y = intercept + slope * X

Where Y in our application would be the CPU utilization, and X is the number of the user session. X is the independent variable that will be used to predict the value of Y. The intercept and the slope are calculated as

$$intercept = \frac{\left((\sum y)(\sum x^2) - (\sum x)(\sum xy)\right)}{(n(\sum x^2) - (\sum x)^2)}$$

$$slope = \frac{\left(n(\sum xy) - (\sum x)(\sum y)\right)}{(n(\sum x^2) - (\sum x)^2)}$$

We could use other example of the independent variable such in time unite, such as a day or an hour, average temperature, competitor price and so on, but we will stick to the earlier definition for simplicity. The given representation is a linear equation, making predictions as simple as solving the equation for a specific set of inputs.

The implementation of regression predictive model can be programmed using a Java program (class) the will be part of the Cognitive Performance Application's analytics engine. The calculation of the liner regression is carried out in the init method as shown in the below code snippet. which is part of the LinearRegressionModel class.

```
public void init( DataSet dataSet )
    {
        int n = dataSet.size();
        double sumX  = 0.0;
        double sumY  = 0.0;
        double sumXX = 0.0;
        double sumXY = 0.0;

        Iterator<DataPoint> it = dataSet.iterator();
        while ( it.hasNext() )
            {
                DataPoint dp = it.next();

                double x = dp.getIndependentValue( independentVariable );
                double y = dp.getDependentValue();

                sumX  += x;
                sumY  += y;
                sumXX += x*x;
                sumXY += x*y;
            }

        double xMean = sumX / n;
        double yMean = sumY / n;

        slope = (n*sumXY - sumX*sumY) / (n*sumXX - sumX*sumX);
        intercept = yMean - slope * xMean;

    }
```

Once the model has been established, the calculation for the slope and the intercept has been completed we use the below method to predict the value for CPU utilization of the next future user session.

```
public double predict( DataPoint dataPoint )
    {

        double x = dataPoint.getIndependentValue( independentVariable );
        double forecastValue = intercept + slope*x;

        dataPoint.setDependentValue( forecastValue );

        return forecastValue;
    }
```

4.3.2. Multiple Linear Regression Model

A multiple variable linear regression model essentially attempts to put a hyperplane through the data points. Mathematically, assuming the independent variables are x_i and the dependent variable is Y. This hyperplane can be represented as in the below equation:

$$Y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3 + ...b_i * x_i$$

- Where the b_i are the coefficients of the regression. b_0 is called the intercept.

Let each of the i predictor variables, x_1, x_2, … x_i, have n levels. Levels can be expressed in the following way:

$$y_1 = b_0 + b_1 * x_{11} + b_2 * x_{12} + b_3 * x_{13} + ...b_i * x_{1i}$$
$$y_2 = b_0 + b_1 * x_{21} + b_2 * x_{22} + b_3 * x_{23} + ...b_i * x_{2i}$$
$$\ldots\ldots$$
$$y_n = b_0 + b_1 * x_{n1} + b_2 * x_{n2} + b_3 * x_{n3} + ...b_i * x_{ni}$$

The system of (n) equations shown previously can be represented in matrix notation as follows:

$$Y = Xb$$

For the special case of one variable the above matrix can be expressed as below:

$$\begin{bmatrix} n & \Sigma x \\ \Sigma x & \Sigma x^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} \Sigma y \\ \Sigma xy \end{bmatrix}$$

For two variables the above matrix can be expressed as below:

$$\begin{bmatrix} n & \Sigma x_1 & \Sigma x_2 \\ \Sigma x_1 & \Sigma {x_1}^2 & \Sigma x_1 x_2 \\ \Sigma x_2 & \Sigma x_1 x_2 & \Sigma {x_2}^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} \Sigma y \\ \Sigma x_1 y \\ \Sigma x_2 y \end{bmatrix}$$

and so on. The previous illustrated calculation for multi regression model is carried out in the init method as shown in the below code snippet, which is part of the MultipleLinearRegressionModel class.

```
    public void init( DataSet dataSet ){


            String varNames[] = dataSet.getIndependentVariables();

            // If no coefficients have been defined for this model,
            //  use all that exist in this data set
            if ( coefficient == null )
                setIndependentVariables( varNames );

            int n = varNames.length;
            double a[][] = new double[n+1][n+2];

            // Iterate through dataSet
            Iterator<DataPoint> it = dataSet.iterator();
            while ( it.hasNext() )
                {
                    // Get next data point
                    DataPoint dp = it.next();

                    // For each row in the matrix, a
                    for ( int row=0; row<n+1; row++ )
                        {
                            double rowMult = 1.0;
                            if ( row != 0 )
                                {
                                    // Get value of independent variable, row
                                    String rowVarName = varNames[row-1];
                                    rowMult = dp.getIndependentValue(rowVarName);
                                }

                            // For each column in the matrix, a
                            for ( int col=0; col<n+2; col++ )
                                {
                                    double colMult = 1.0;
                                    if ( col == n+1 )
```

```
                                        {
                                            // Special case, for last column
                                            //  use value of dependent variable
                                            colMult = dp.getDependentValue();
                                        }
                                    else if ( col > 0 )
                                        {
                                            // Get value of independent variable, col
                                            String colVarName = varNames[col-1];
                                            colMult =
    dp.getIndependentValue(colVarName);
                                        }

                                    a[row][col] += rowMult * colMult;

                                }
                        }
                }

            // Solve equations to derive coefficients
            double coeff[] = Utils.GaussElimination( a.length(), a );

            // Assign coefficients to independent variables
            intercept = coeff[0];
            for ( int i=1; i<n+1; i++ )
                coefficient.put( varNames[i-1], new Double(coeff[i]) );

            // Calculate the accuracy indicators
            calculateAccuracyIndicators( dataSet );
        }
```

Once the multi regression problem is expressed in a matrix form we can implement a Gaussian elimination on the given matrix to find the values of the coefficients b_0, b_1, b_2 … b_n, as shown in the code snippet below, which is part of a common utility class.

```
static double[] GaussElimination( int n, double a[][] )
    {
        // Forward elimination
        for ( int k=0; k<n-1; k++ )
            {
                for ( int i=k+1; i<n; i++ )
                    {
                        double qt = a[i][k] / a[k][k];
                        for ( int j=k+1; j<n+1; j++ )
                            a[i][j] -= qt * a[k][j];
                        a[i][k] = 0.0;
                    }
            }

        double x[] = new double[n];
        // Back-substitution
        x[n-1] = a[n-1][n] / a[n-1][n-1];
        for ( int k=n-2; k>=0; k-- )
            {
                double sum = 0.0;
                for ( int j=k+1; j<n; j++ )
                    sum += a[k][j]*x[j];
                x[k] = ( a[k][n] - sum ) / a[k][k];
            }
        return x;
    }
```

Once the model has been established, or that the y-intercept and the other coefficients have been calculated, we can use the below method to predict the value for CPU utilization, or the independent value, of the next future iteration.

```
Public double predict( DataPoint dataPoint )
   {

       double forecastValue = intercept;

       Iterator< Map.Entry<String,Double> > it = coefficient.entrySet().iterator();
       while ( it.hasNext() )
           {
               Map.Entry<String,Double> entry = it.next();

               // Get value of independent variable
               double x = dataPoint.getIndependentValue( (String)entry.getKey() );

               // Get coefficient for this variable
               double coeff = ((Double)entry.getValue()).doubleValue();
               forecastValue += coeff * x;
           }

       dataPoint.setDependentValue( forecastValue );

       return forecastValue;
   }
```

4.3.3. Polynomial Regression Model

When the relation between the independent variable (x) and the dependent variable (y) is not linear is not linear but rather show as "curve" we can use a single variable polynomial regression model, which essentially attempts to put a polynomial line through the data points. Mathematically, assuming the independent variable is x and the dependent variable is y, then this line can be represented as:

$$y = b_0 + b_1*x^1 + b_2*x^2 + b_3*x^3 + \ldots + b_m*x^m$$

*Where "m" is <u>order</u> of the polynomial equation. [11]

Without getting into much details of how to drive the mathematical equation for the polynomial regression, which is beyond the scope of this document, the problem can represent itself in a matrix form as shown below.

$$\begin{bmatrix} n & \Sigma x_i & \dots & \Sigma x_i{}^m \\ \Sigma x_1 & \Sigma x_i{}^2 & \dots & \Sigma x_i{}^{m+1} \\ \dots & \dots & \dots & \dots \\ \Sigma x_i{}^m & \Sigma x_i{}^{m+1} & \dots & \Sigma x_i{}^{2m} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \dots \\ b_m \end{bmatrix} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \\ \dots . \\ \Sigma x_i{}^m y_i \end{bmatrix}$$

This matrix presentation is carried out in the init method as shown in the below code snippet, which is part of the PolynomialRegressionModel class.

```
public void init( DataSet dataSet )
    {
        double a[][] = new double[order][order+1];

        for ( int i=0; i<order; i++ ) {
                for ( int j=0; j<order; j++ ){
                        int k = i + j;

                        Iterator<DataPoint> it = dataSet.iterator();
                        while ( it.hasNext() )
                            {
                            DataPoint dp = it.next();

                            double x = dp.getIndependentValue( independentVariable );

                            a[i][j] = a[i][j] + Math.pow(x,k);
                            }
                    }

                Iterator<DataPoint> it = dataSet.iterator();
                while ( it.hasNext() )
                    {
                        DataPoint dp = it.next();

                        double x = dp.getIndependentValue( independentVariable );
                        double y = dp.getDependentValue();

                        a[i][order] += y*Math.pow(x,i);
                    }
            }

        coefficient = Utils.GaussElimination( order, a );

        // Calculate the accuracy indicators
        calculateAccuracyIndicators( dataSet );

  }
```

Once the single polynomial regression problem is expressed in a matrix form we can implement a Gaussian elimination on the given matrix to find the values of the coefficients a_0, a_1, a_2 … a_n, as shown in the previous section. And once the polynomial regression model calculation has been completed, we can use the below method to predict the value for CPU utilization, or the independent value, of the next future iteration.

```
public double predict( DataPoint dataPoint )
    {

        double x = dataPoint.getIndependentValue( independentVariable );
        double forecastValue = 0.0;
        for ( int i=0; i<order; i++ )
            forecastValue += coefficient[i] * Math.pow(x,i);

        dataPoint.setDependentValue( forecastValue );

        return forecastValue;
    }
```

4.3.4. Models Comparison

Once the above models are established and calculated, we use an evaluation criteria class to specify how to compare the different models based on the values of Akaike Information Criteria (AIC), Arithmetic mean of the errors (bias), Mean Absolute Deviation (MAD), Mean Absolute Percentage Error (MAPE), Mean square of the errors (MSE), Sum of Absolute Errors (SAE.) we can also use them all in our comparison as shown in the code below which is part of the Predictor class.

```
 static boolean compare( PredictingModel model1,
                                       PredictingModel model2,
                                       EvaluationCriteria evalMethod )
    {
        // Special case. Any model is better than no model!
        if ( model2 == null )
            return true;

        double tolerance = 0.00000001;

        // Use evaluation method as requested by user
        if ( evalMethod == EvaluationCriteria.BIAS )
            return ( model1.getBias() <= model2.getBias() );
        else if ( evalMethod == EvaluationCriteria.MAD )
            return ( model1.getMAD() <= model2.getMAD() );
        else if ( evalMethod == EvaluationCriteria.MAPE )
            return ( model1.getMAPE() <= model2.getMAPE() );
        else if ( evalMethod == EvaluationCriteria.MSE )
            return ( model1.getMSE() <= model2.getMSE() );
        else if ( evalMethod == EvaluationCriteria.SAE )
            return ( model1.getSAE() <= model2.getSAE() );
        else if ( evalMethod == EvaluationCriteria.AIC )
            return ( model1.getAIC() <= model2.getAIC() );

        // Default evaluation method is a combination
        int score = 0;
        if ( model1.getAIC()-model2.getAIC() <= tolerance )
            score++;
        else if ( model1.getAIC()-model2.getAIC() >= tolerance )
            score--;

        if ( model1.getBias()-model2.getBias() <= tolerance )
            score++;
        else if ( model1.getBias()-model2.getBias() >= tolerance )
            score--;

        if ( model1.getMAD()-model2.getMAD() <= tolerance )
            score++;
        else if ( model1.getMAD()-model2.getMAD() >= tolerance )
            score--;

        if ( model1.getMAPE()-model2.getMAPE() <= tolerance )
            score++;
        else if ( model1.getMAPE()-model2.getMAPE() >= tolerance )
            score--;

        if ( model1.getMSE()-model2.getMSE() <= tolerance )
            score++;
        else if ( model1.getMSE()-model2.getMSE() >= tolerance )
            score--;

        if ( model1.getSAE()-model2.getSAE() <= tolerance )
            score++;
        else if ( model1.getSAE()-model2.getSAE() >= tolerance )
            score--;


        return ( score > 0 );
    }
}
```

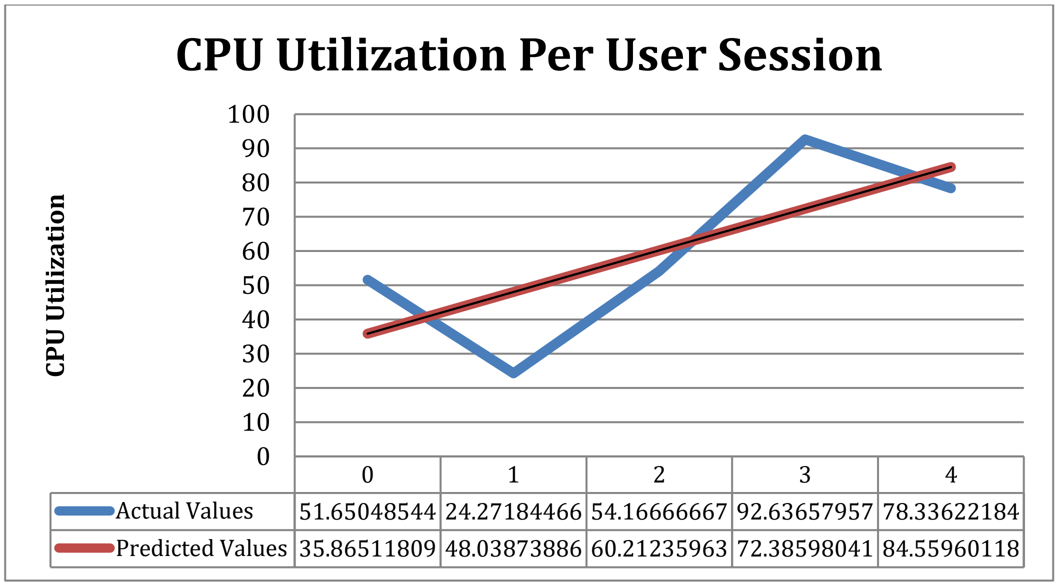
4.4. Run Example

The below example will show how the analytic system predict the value of the CPU utilization for the next User session based on the CPU utilization values recorded for the immediate preceding five user sessions as shown in the below table.

User Session #	CPU Utilization
0	51.6504854368932
1	24.271844660194176
2	54.166666666666664
3	92.63657957244655
4	78.33622183708839
5	?

Table 9 –CPU Per User Session

The below graph is a visual presentation of these data point.



	0	1	2	3	4
Actual Values	51.65048544	24.27184466	54.16666667	92.63657957	78.33622184
Predicted Values	35.86511809	48.03873886	60.21235963	72.38598041	84.55960118

Figure 19 – ANALYTICS ENGINE EXAMPLE RUN

The analytical engine decided that the best model is the Analytics Engine is linear regression model with the following equation:

$$y=35.86511809212924+12.173620771264279*x$$

, and the predicted value of the CPU utilization for the next run will be predicted value:

96.73322194845063

as shown in the system logs captured below.

```
Data set: (
  (x=0.0,dependentValue=51.6504854368932)
  (x=1.0,dependentValue=24.271844660194176)
  (x=2.0,dependentValue=54.166666666666664)
  (x=3.0,dependentValue=92.63657957244655)
  (x=4.0,dependentValue=78.33622183708839)
)
bestModel: linear regression model with the following equation:
  y=35.86511809212924+12.173620771264279*x
count---5
predicted value: 96.73322194845063
```

4.5. Analytics Engine Class Diagram

The below graph a UML class diagram summarizing the structure of the learning engine.

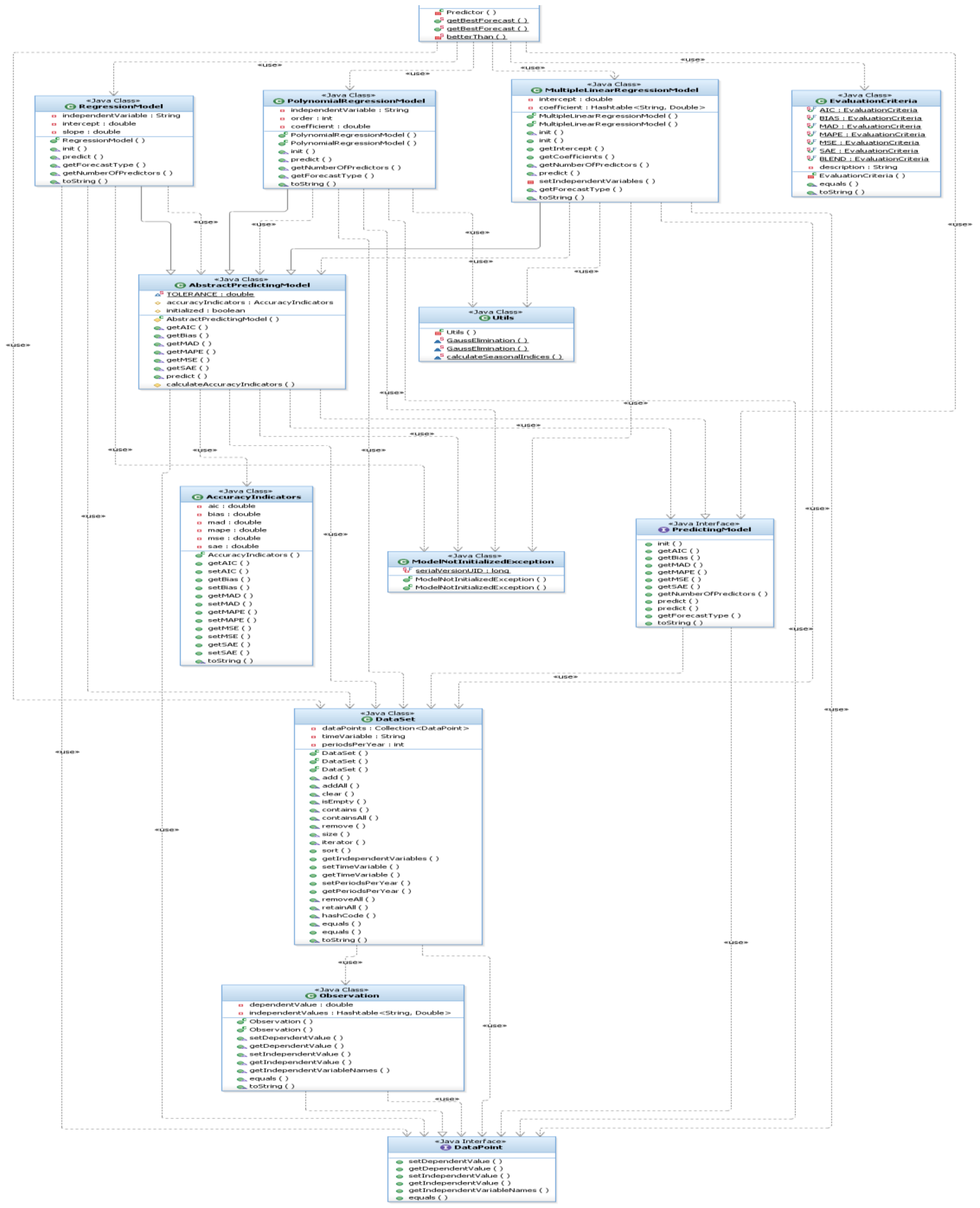


Figure 20 – Analytics engine UML Class diagram

4.6. Summary

The Analytics Engine is a component of the Cognitive Performance Application that establishes a pattern between library of factors and the system performance gauges to act as an early alert system signaling a potential deadlock. In this chapter we explained the architecture decision of using a factory design pattern to make the model scalable for future updates. We also have shown the three different models that constitute the core of our engine, namely

- Linear Regression Model
- Multiple Linear Regression Model
- Polynomial Linear Regression Model

Moreover, we have explained how the system will compare and choose between these model based on the accuracy of the their calculation using the following criteria

- Akaike Information Criteria (AIC)
- Arithmetic mean of the errors (bias)
- Mean Absolute Deviation (MAD)
- Mean Absolute Percentage Error (MAPE)
- Mean square of the errors (MSE)
- Sum of Absolute Errors (SAE)

Cl!t¢9W 5 QUEUING ENGINE

The Queuing engine is the last component of the Cognitive Performance Application. The engine will have two execution modes.

- A pass-through
- Queuing mode.

The learning engine or the analytics engine will signal the mode of execution for the queening engine. When the pass-through signal is flagged, all database transactions will go uninterrupted from the JVM to the DBMS. When the queuing mode is signaled, the database queries will join a First-Come-First-Serviced (FCFS) queue structure. The database queries in the queue structure will be concurrently executed after adding a trifling delay to each in order to stream down the flow and reduce the performance load on the JVM and DBMS. The queuing engine will iterate between the two execution modes based on the signal given.

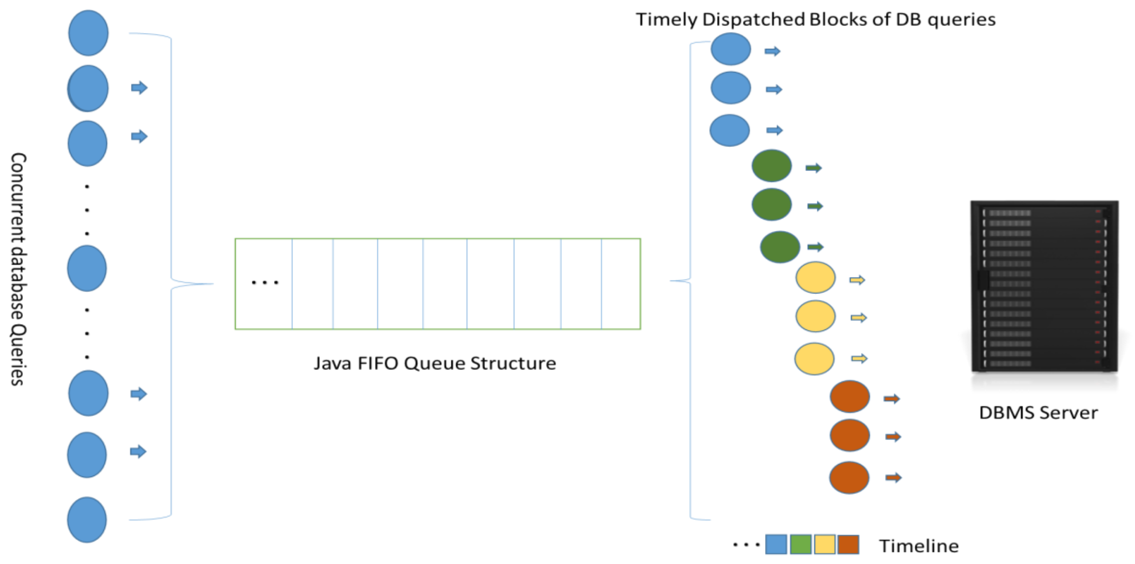

Concurrent database Queries
Java FIFO Queue Structure
Timely Dispatched Blocks of DB queries
DBMS Server
Timeline

Figure 21 – Java Queuing Engine

We will use native library called BlockingQueue.Java.util.concurrent.BlockingQueue, which is a Java Queue that support operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element. Java BlockingQueue doesn’t accept null values and throw NullPointerException when storing null value in the queue. Java BlockingQueue implementations are thread-safe. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control. Java BlockingQueue interface is part of Java collections framework and it’s primarily used for implementing producer consumer problem. We don’t need to worry about waiting for the space to be available for producer or object to be available for consumer in BlockingQueue because it is handled by implementation classes of BlockingQueue. Java provides several BlockingQueue implementations such as ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue etc. While implementing producer consumer problem in BlockingQueue, we will use LinkedBlockingQueue implementation.

```java
package cpu.cs.louisville.edu;


import Java.sql.Connection;
import Java.sql.Statement;
import Java.text.NumberFormat;
import Java.util.Random;
import Java.util.concurrent.BlockingQueue;
import Java.util.concurrent.LinkedBlockingDeque;


public class UserSession implements Runnable {

        static Connection connect = null;
        static Statement statement = null;

        private int threadNumber;
        private boolean runTimeMode = true;
        private int counterRunRound;

        public UserSession(int threadNum, boolean runTime, int counterRunNum ){
                this.threadNumber = threadNum;
                this.runTimeMode = runTime;
                this.counterRunRound = counterRunNum;
        }

        NumberFormat format = NumberFormat.getInstance();

        static long maxMemory ;
        static long allocatedMemory;
        static long heapRatio;

    private BlockingQueue<SessionQueries> concurrentLinkedQueue = new
LinkedBlockingDeque<SessionQueries>(100);

    public void dequeueItem() {
       if (!concurrentLinkedQueue.isEmpty()) {
         //  System.out.println("Queue size: " + concurrentLinkedQueue.size());
           try {
                       concurrentLinkedQueue.take().runQueries(connect, statement,
format, maxMemory, allocatedMemory,
                                      heapRatio, this.threadNumber,
this.counterRunRound);
               } catch (Exception e) {}
       } else {
          System.out.println("Queue Empty ");
       }
    }

    private void enqueueItem(SessionQueries item) throws Exception {
    //   System.out.println("Enqueueing item ");
       concurrentLinkedQueue.put(item);
    }

    public int getQueueSize() {
       if (!concurrentLinkedQueue.isEmpty()) {
           return concurrentLinkedQueue.size();
       } else {
           return 0;
       }
    }

  /**
   *
   */
    public void run(){
       SessionQueries session = new SessionQueries();

            int numberOfUserSession = ( new Random().nextInt(100)) ;
            System.out.println("number Of User Session "+numberOfUserSession);
            for (int count = 0 ; count < numberOfUserSession ; count++){
                // if ( this.runTimeMode == true ){
                  // StaticRecord st = new StaticRecord();
                       if (StaticListener.isDeadLock() == false){

                          try {
                                                        session.runQueries(connect,
statement, format, maxMemory, allocatedMemory, heapRatio, count, this.counterRunRound);
                                                } catch (Exception e) {
                                                        e.printStackTrace();
                                                }
                       }else {
                                                try {
                                                        enqueueItem(session);
                                                } catch (Exception e) {
                                                        e.printStackTrace();
                                                }
                       }
             }
     }
}// end of UserSession
```

This queue structure will be used to rectify the system deadlock situation when the number of queries overwhelms the system and cause the CPU utilization to reach its peak

C l ! t¢9w 6 RANDOM LOAD SIMULATION

In this section, a system run is descried for a certain amount of duration, five minutes to be exact. During the five minutes range, the simulator program triggers a random number of threads, user sessions, at constant rate, 12 seconds to be exact. Also for this run the Java heap size is being reduced to help the system reach a bottleneck situation and hence study the recovery mechanism.

6.1. Discrete vs. Continuous

The simulator only triggers random number of threads, sampling user loads, every twelfth seconds during a five-minute range and hence, by definition, is classified as a discrete process. The choice of using a discrete run was to assemble the actual user load on an online system. Otherwise, the simulator would have triggered the number of the users in a loop structure such as the Java language “for” or “while” loops continuously for the same time range, or five minutes as in our case.

6.2. Pseudo-random Number Sampling

The simulator triggers random number of threads, every twelfth seconds during a five-minute range. The random number of threads is generated using a Java Pseudo-random generator utility. The Java number sampling or non-uniform pseudo-random variate generation is the numerical method of generating random numbers distributed according to a normal distribution of certain mean and standard deviation.

6.2.1. Normalization

In order to get to mean and the average we have studied the user load on the hp ecommerce store for four consecutive working weeks as shown below. We then normalized the data to make it is values between 0 and 1 using the following equation:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Then we calculated the mean and standard deviation as shown in the table below.

Day	#	Number of users	Normalized
18-Sep	1	231	0.401993355
19-Sep	2	197	0.289036545
20-Sep	3	231	0.401993355
21-Sep	4	131	0.069767442
22-Sep	5	286	0.584717608
25-Sep	6	374	0.877076412
26-Sep	7	110	0
27-Sep	8	282	0.571428571
28-Sep	9	229	0.395348837
2-Oct	10	411	1
3-Oct	11	220	0.365448505
4-Oct	12	411	1
5-Oct	13	333	0.740863787
6-Oct	14	284	0.57807309
8-Oct	15	192	0.272425249
9-Oct	16	218	0.358803987
10-Oct	17	242	0.438538206
11-Oct	18	269	0.528239203
12-Oct	19	188	0.259136213
Mean		**254.6842105**	**0.48067844**
Standard deviation		**83.15183744**	**0.276251952**

Min		**110**	**0**
Max		**411**	**1**

Table 10 -HP2B daily user load

The Java number sampling or non-uniform pseudo-random variate generation is the numerical method of generating random numbers distributed according to a normal distribution of mean 0.48067844 and standard deviation 0.276251952, derived from the above calculation. The choice of normal distribution is more appropriate because if the sample size is sufficiently large, as in our case, the sampling distribution of the sample mean approximates the normal distribution. This holds true despite the distribution of the population from which the sample were collected according to the central limit theorem.

6.3. Results of First Run

The first run does not entail any performance enhancement mechanism. The first run is aimed to observe and collect the as-is system performance metrics to be compared later on after the addition of the learning, analytics, and queue components. The first run resulted in the execution of a total of 1004 threads or user sessions. The first run is triggered over a five minutes time range, within which a random number of user sessions is rendered every 12 seconds using Java random number generator and normally distributed. Performance metrics such as the

- CPU Utilization percentage,
- Thread response time in seconds,
- Allocated memory in KB
- Heap ratio

are collected and recorded toward the end of each thread execution. A summary of the run data is shown in the below table and the further subsequent sections.

CPU Usage	*Response Time (s)*	*Start Time*	*End Time*	*Thread Number*	*Heap Ratio*	*Allocated Memory (KB)*	*Max Memory (KB)*
100.00%	6.82298798	2051670.832	2051677.655	1	0.03	6.17	178.176
100.00%	2.10095347	2051677.677	2051679.778	2	0.08	15.473	178.176
100.00%	2.75671768	2051679.779	2051682.536	3	0.11	20.408	178.176
100.00%	3.88481427	2051682.537	2051686.422	4	0.09	16.331	178.176
100.00%	3.52589645	2051691.545	2051695.07	5	0.09	16.496	178.176
100.00%	2.88591846	2051695.072	2051697.958	6	0.10	17.844	178.176
100.00%	2.55195492	2051697.959	2051700.511	7	0.09	16.577	178.176
100.00%	3.08366706	2051700.512	2051703.596	8	0.08	15.64	178.176
100.00%	2.90243103	2051708.703	2051711.605	9	0.07	12.811	178.176
……	……	……	……	……	……	……	……
……	……	……	……	……	……	……	……
……	……	……	……	……	……	……	……
……	……	……	……	……	……	……	……
100.00%	2.45123124	2054600.353	2054602.804	996	0.64	115.208	178.176
100.00%	14.5856296	2054590.063	2054604.648	997	0.79	141.898	178.176
100.00%	32.38739141	2054591.418	2054623.806	998	0.63	112.641	178.176
100.00%	8.17268885	2054613.735	2054621.908	999	0.48	85.554	178.176
100.00%	12.50983365	2054608.014	2054620.523	1000	0.58	103.689	178.176
100.00%	11.90430417	2054607.993	2054619.898	1001	0.53	95.513	178.176
100.00%	14.20815774	2054605.491	2054619.699	1002	0.52	93.001	178.176
100.00%	11.01713206	2054608.006	2054619.023	1003	0.47	83.931	178.176
54.47%	3.48059529	2054627.697	2054631.178	1004	0.63	112.828	178.176

Table 11 –First Run Summary

6.3.1. CPU Utilization

The below graph and table shows the system CPU utilization recorded toward the end of the execution of each thread, or a user session, along with a data statics summary including the average, max, min, and standard deviation.

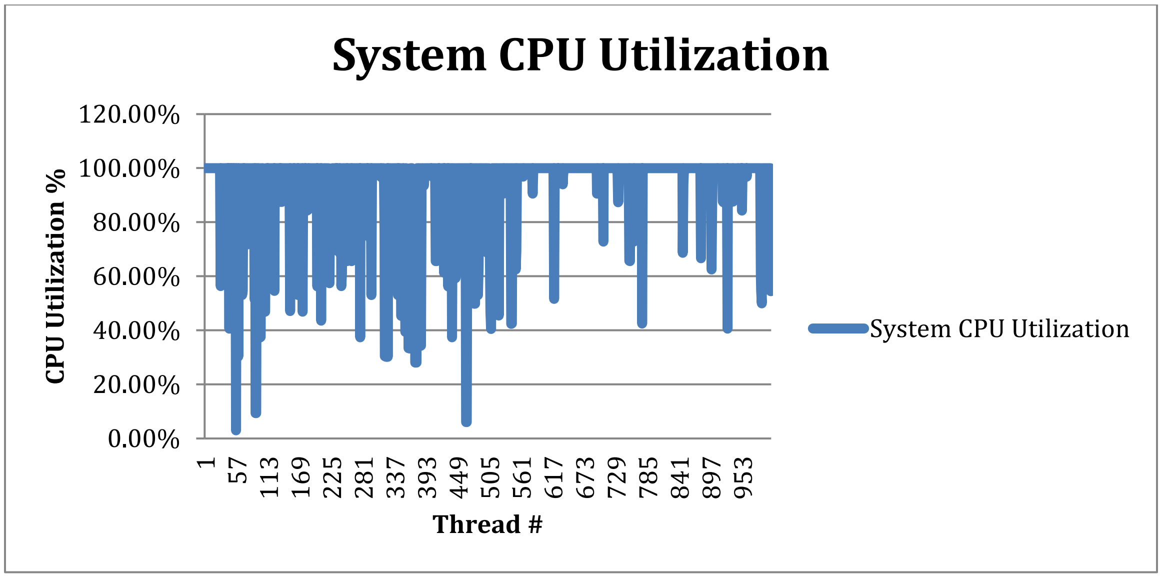


Figure 22 – System CPU utlilization charts

STATS Analysis	Value
Average	**94.15%**
Max	**100.00%**
Min	**3.01%**
StdDev	**0.158067891**

Table 12 –First Run CPU data Summary

6.3.2. Response Time Per Thread

The below graph records each thread response time calculated the difference between the thread start time and the thread end time. The response time of each query will be measured using the nanoTime() method to allow for max precision. This method provides nanosecond precision, but not necessarily nanosecond accuracy. For example, to measure how long some database transaction takes to execute:

```
long startTime = System.nanoTime();    // ... the code being measured ...
long estimatedTime = System.nanoTime() - startTime;
```

The subsequent table displays a data statics summary including the average, max, min, and standard deviation for all the user sessions involved in the run.

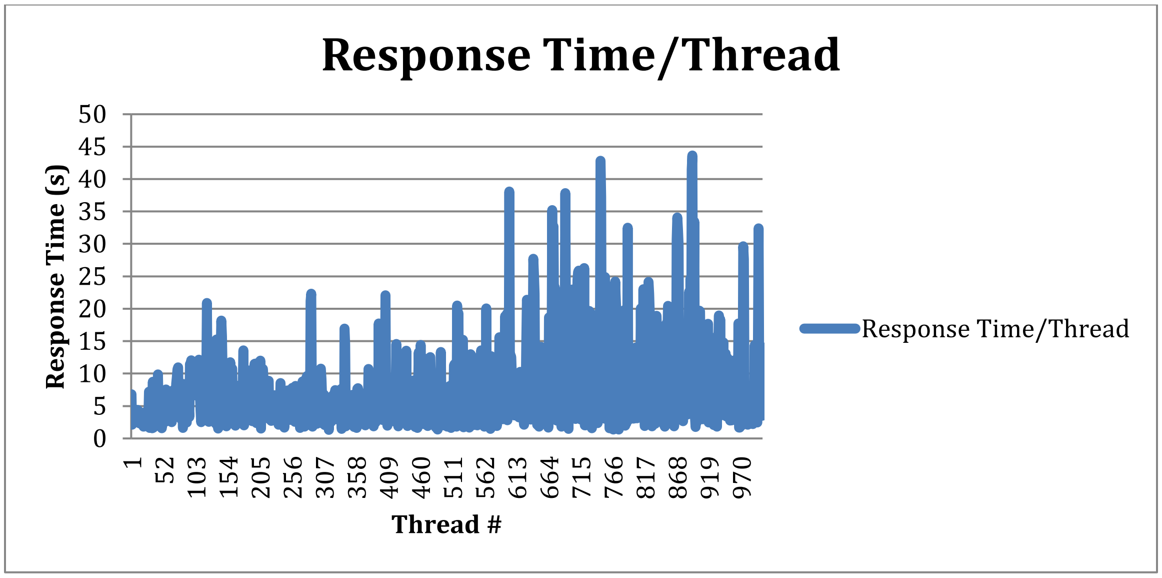


Figure 23 – RESPONSE TIME IN SECONDS PER THREAD

STATS Analysis	Value
Average	**8.09**
Max	**43.63**
Min	**1.33**
StdDev	**6.35**

Table 13 –First run thread response time summary

6.3.3. Heap Ratio

The below graph shows the JVM heap ratio recorded toward at the end of the execution of each thread, or a user session, by retrieving the JVM allocated memory divided my max memory variable. The subsequent table displays the heap ratio data statics summary including first Quartile, second Quartile (median), and third Quartile.

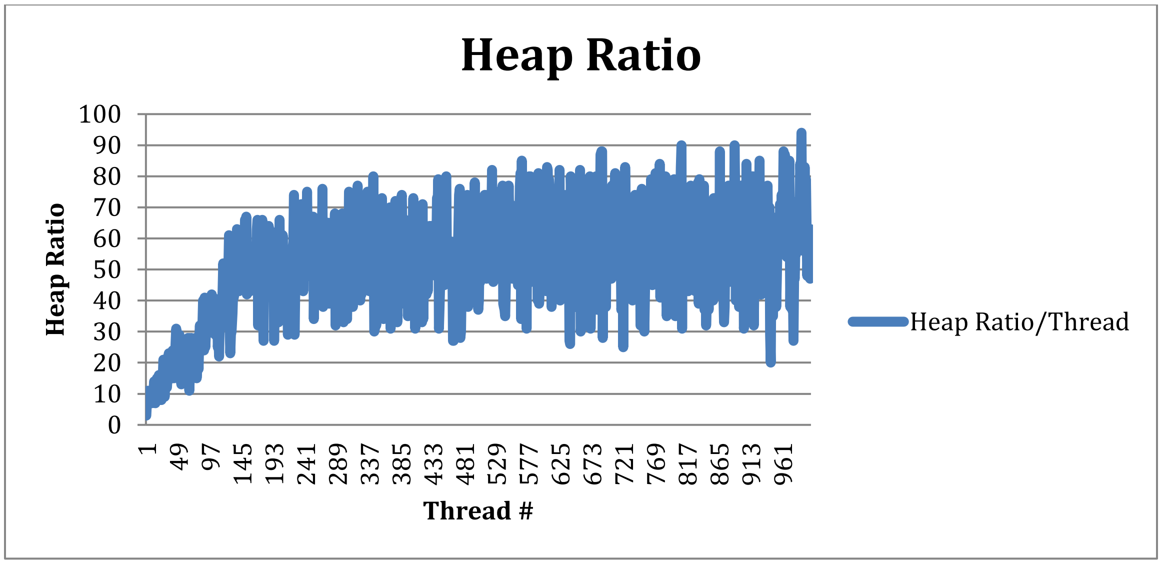


Figure 24 – HEAP RATIO PER THREAD

STATS Analysis	Value
First Quartile	**43**
Second Quartile (median)	**55**
Third Quartile	**63.75**

Table 14 –First run heap ratio summary

6.4. Results of Second Run

The second run resulted the execution 940 threads or user session. The second run is similar to the first one in that it is also triggered over exactly five minutes time range, within which a random number of user sessions is rendered every 12 seconds using Java random number generator. The main difference from the first run is that it includes two extra major components to enhance the performance. These are the learning engine and the queuing engine. These two components are responsible mainly for defining and detecting the system bottleneck and introducing a recovering mechanism. The system resorts to the recovering mechanism if and only a bottleneck is detected and then alters back to normal execution mode once the bottleneck conditions are void. The details of these two extra components will be covered on different chapters of this document. A summary of the run data is shown in the below table.

CPU Usage	*Response Time (s)*	*Start Time*	*End Time*	*Thread Number*	*Heap Ratio*	*Allocated Memory (KB)*	*Max Memory (KB)*
71.45%	6.36287852	2068323.678	2068330.041	1	4	8.059	178.176
44.42%	3.37606177	2068340.181	2068343.557	2	4	7.989	178.176
37.73%	3.18759926	2068348.452	2068351.64	3	5	9.113	178.176
41.13%	2.90369656	2068362.034	2068364.938	4	6	11.366	178.176
80.76%	4.02605219	2068369.837	2068373.863	5	8	14.83	178.176
42.29%	3.09899373	2068378.888	2068381.987	6	5	10.162	178.176
52.84%	3.21760867	2068386.854	2068390.072	7	11	20.212	178.176
31.19%	2.3927622	2068400.266	2068402.659	8	10	18.542	178.176
71.45%	6.36287852	2068323.678	2068330.041	1	4	8.059	178.176
……	……	……	……	……	……	……	……
……	……	……	……	……	……	……	……
……	……	……	……	……	……	……	……
100.00%	9.21452238	2071222.498	2071231.713	931	56	101.22	178.176
100.00%	14.98215555	2071222.504	2071237.486	932	49	88.945	178.176
100.00%	16.79426994	2071222.509	2071239.303	933	63	113.543	178.176
0.00%	5.0810955	2071244.65	2071249.731	934	75	134.55	178.176
100.00%	39.96265167	2071222.531	2071262.494	935	61	109.058	178.176
100.00%	37.2201013	2071222.525	2071259.745	936	61	109.112	178.176
100.00%	34.5797734	2071222.52	2071257.1	937	40	71.511	178.176
100.00%	32.23581489	2071222.515	2071254.75	938	46	82.62	178.176
100.00%	6.77268899	2071244.659	2071251.432	939	88	158.063	178.176

Table 15 –Second Run Summary

The subsequent sections show more detailed analysis on the performance metrics collected through the run.

6.4.1. CPU Utilization

The below graph and table shows the system CPU utilization recorded toward at the end of the execution of each thread, or a user session, along with a data statics summary including the average, max, min, and standard deviation.

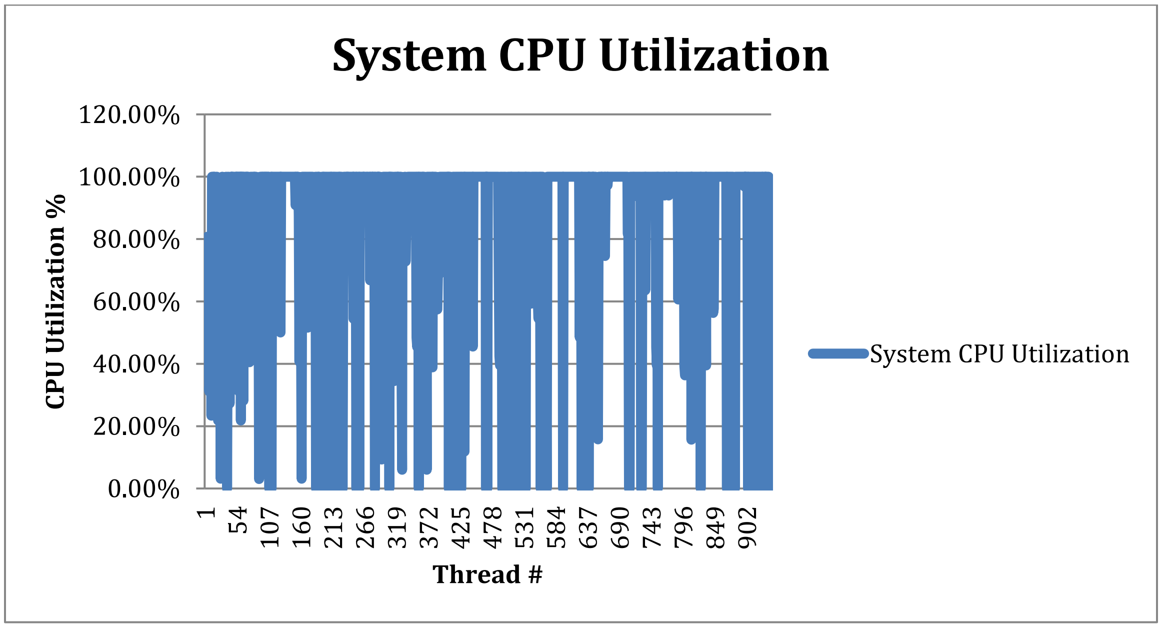


Figure 25 – System CPU utlilization charts

STATS Analysis	Value
Average	**87.80%**
Max	**100.00%**
Min	**0.00%**
StdDev	**0.268636399**

Table 16 –Second run CPU data summary

The second run shows significant improvement in the overall CPU utilization. Also it shows that the added components managed to decrease the frequency of which the system hits the bottleneck. These results are visualized and calculated in the above graph and table.

6.4.2. Response Time Per Thread

The below graph records each thread response time calculated by recording the thread start time and end time then retrieving the difference as the thread response time. The subsequent table displays a data statics summary including the average, max, min, and standard deviation for all the user sessions involved in the run.

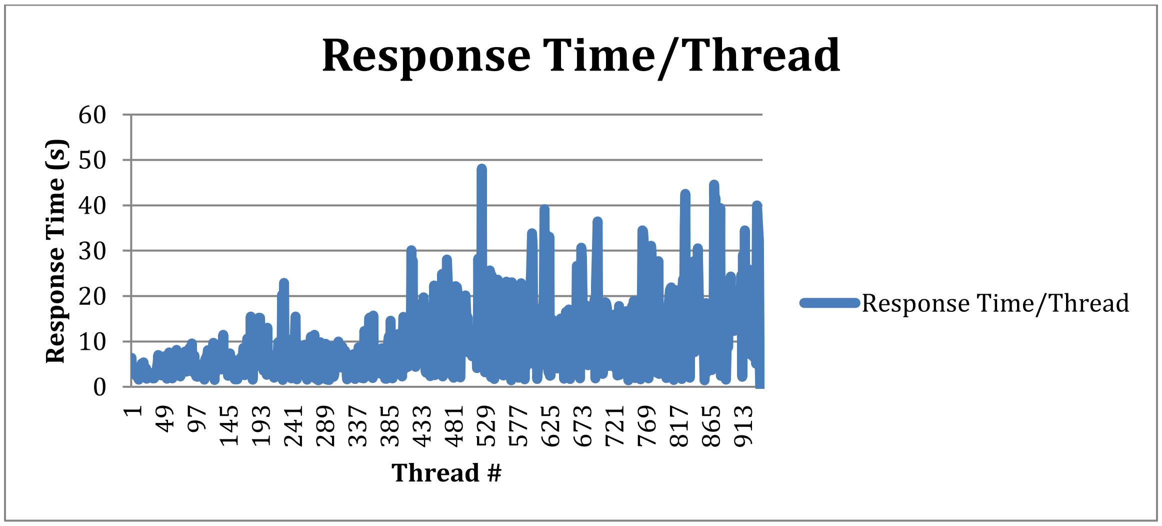


Figure 26 – *Response Time in seconds per thread*

STATS Analysis	Value
Average	**9.97**
Max	**48.09**
Min	**1.40**
StdDev	**7.58**

Table 17 –Second run thread response time data summary

The second run shows slight change in the overall response time. However, the threadresponse time is shown to be a little longer due to the introductory of the queuing component as a recovery mechanism. The stacking of the threads to the queuing engine, which will be detailed in subsequent sections, during the system peak times, contributed to the increase of some of the individual thread's response time, and hence increasing the overall response times of the user sessions as recorded in the above table.

6.4.3. Heap Ratio Per Thread

The below graph shows the JVM heap ratio recorded toward at the end of the execution of each thread, or a user session, by retrieving the JVM allocated memory divided my max memory variable. The subsequent table displays the heap ratio data statics summary including the first Quartile, second Quartile (median), and third Quartile.

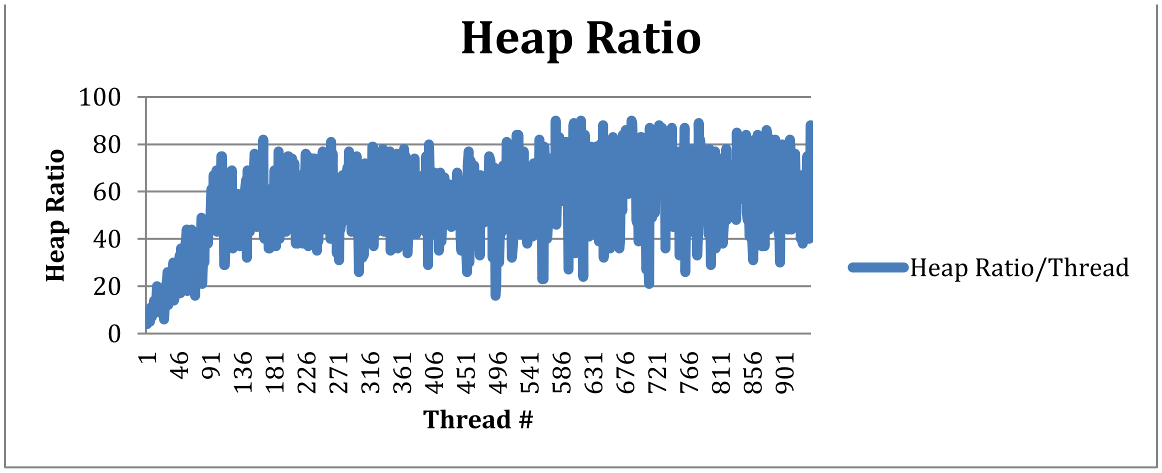


Figure 27 – HEAP RATIO PER THREAD

STATS Analysis	Value
First Quartile	**45**
Second Quartile (median)	**58**
Third Quartile	**68**

Table 18 –Second run heap ratio data summary

The heap ratio also shows to be similar to the first run with a slight or insignificant increase resulted from the addition of the recovery components.

6.5. Results of Third Run

The third run resulted in the execution of a total of 905 threads, user sessions. The second run is similar to the first one in that it is also triggered over a five minutes time range, within which a random number of user sessions is rendered every 12 seconds using Java random number generator. The main difference from the first and second runs is that it includes, along with the learning and the queuing components, a third component. This new component is the analytic engine, which is mainly responsible for predicting the possibility of the system reaching a bottleneck based on the historic runs. The details of all these components; learning, queuing and analytics, will be covered in subsequent sections. A summary of the run data is shown in the below table.

CPU Usage	*Response Time (s)*	*Start Time*	*End Time*	*Thread Number*	*Heap Ratio*	*Allocated Memory (KB)*	*Max Memory (KB)*
91.64%	5.6798846	2082177.806	2082183.486	1	2	5.301	178.176
40.05%	3.51932159	2082198.698	2082202.217	2	8	14.28	178.176
50.00%	3.64850281	2082212.432	2082216.081	3	9	17.44	178.176
53.82%	3.09170858	2082221.019	2082224.111	4	13	23.777	178.176
39.59%	3.13410433	2082229.13	2082232.265	5	12	21.952	178.176
38.55%	3.14383253	2082237.258	2082240.402	6	8	15.233	178.176
36.50%	3.01136807	2082250.672	2082253.684	7	8	15.481	178.176
61.89%	3.59677626	2082258.63	2082262.227	8	6	11.363	178.176
42.49%	2.19742104	2082267.364	2082269.561	9	14	25.296	178.176
……	……	……	……	……	……	……	……
……	……	……	……	……	……	……	……
……	……	……	……	……	……	……	……
96.80%	9.71586053	2085075.284	2085085	896	62	111.818	178.176
100.00%	2.90107402	2085089.062	2085091.963	897	76	136.73	178.176
100.00%	7.89125783	2085089.532	2085097.423	898	43	76.935	178.176
100.00%	2.7990394	2085100.152	2085102.951	899	57	101.806	178.176
100.00%	7.5139068	2085091.722	2085099.236	900	56	101.097	178.176
100.00%	17.28572346	2085097.036	2085114.322	901	69	123.635	178.176
100.00%	15.42726171	2085097.027	2085112.455	902	78	140.132	178.176

100.00%	10.02460508	2085096.904	2085106.928	903	60	107.945	178.176
100.00%	3.3531615	2085107.334	2085110.687	904	64	114.335	178.176

Table 19 –Third run data summary

The subsequent sections show more detailed analysis on the performance metrics collected through the run.

6.5.1. CPU Utilization

The below graph and table shows the system CPU utilization recorded toward at the end of the execution of each thread, or a user session, along with a data statics summary including the average, max, min, and standard deviation.

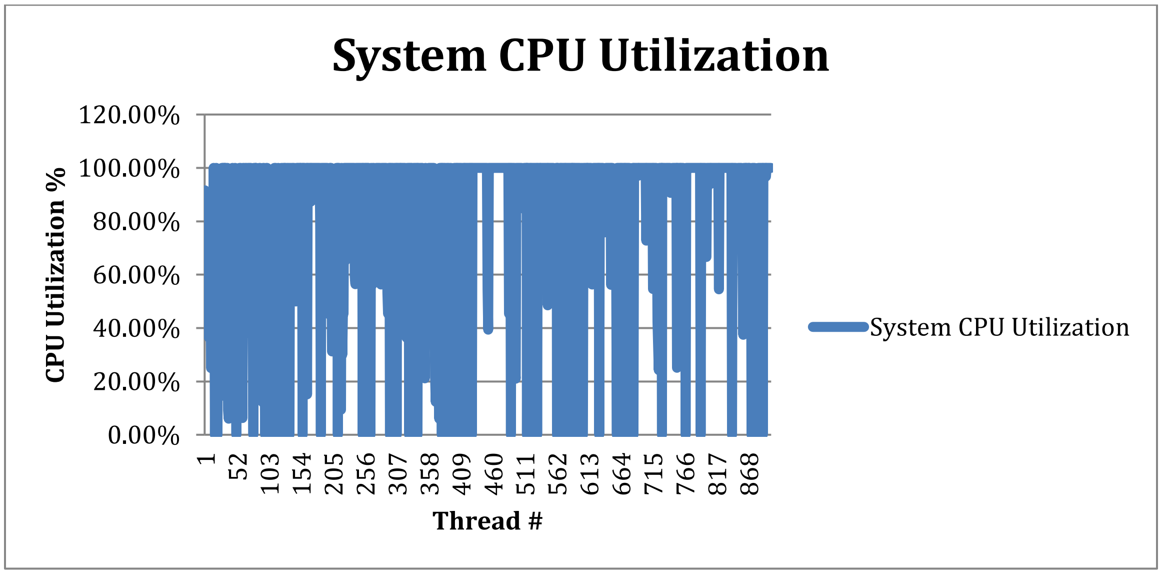


Figure 28 – *System CPU utlilization charts*

STATS Analysis	Value
Average	**84.60%**
Max	**100.00%**
Min	**0.00%**
StdDev	**0.293183864**

Table 20 –Third run CPU utilization data summary

The second run shows meaningful another improvement in the overall CPU utilization. Also, it shows that the added component managed to further decrease the frequency of which the system hits the bottleneck. These results can be inspected visually or mathematically from the above graph and table.

6.5.2. Response Time Per Threads

The below graph records each thread response time calculated by recording the thread start time and end time then retrieving the difference as the thread response time. The subsequent table displays a data statics summary including the average, max, min, and standard deviation for all the user sessions involved in the run

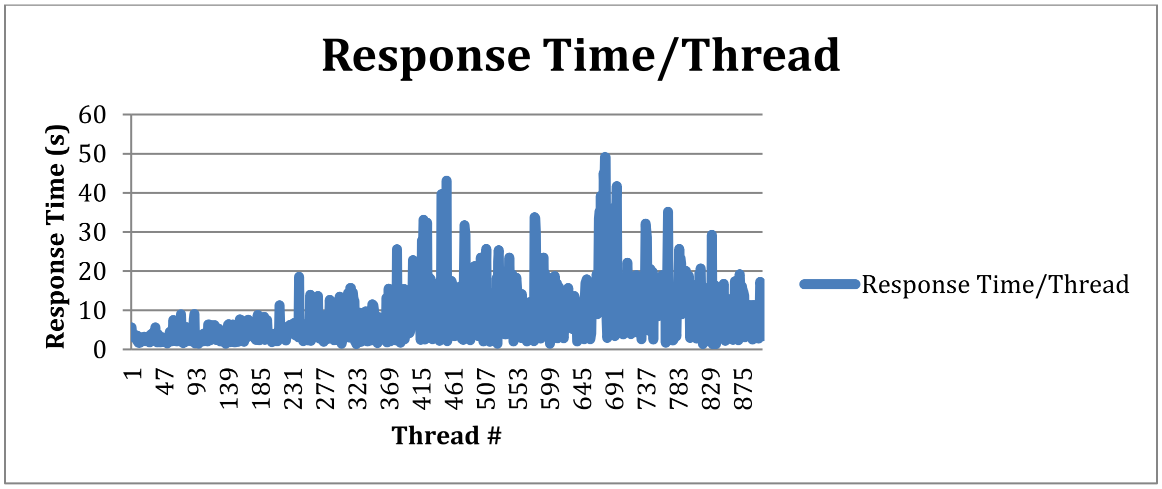


Figure 29 – RESPONSE TIME IN SECONDS PER THREAD

STATS Analysis	Value
Average	**9.02**
Max	**49.14**
Min	**1.38**
StdDev	**7.22**

Table 21 –Third run response time data summary

The third run shows a better overall response time from the second run. However, the thread response time shows to be little longer than the first run.

6.5.3. Heap Ratio Per Thread

The below graph shows the JVM heap ratio recorded toward at the end of the execution of each thread, or a user session, by retrieving the JVM allocated memory divided my max memory variable. The subsequent table displays the heap ratio data statics summary including the first Quartile, second Quartile (median), and third Quartile.

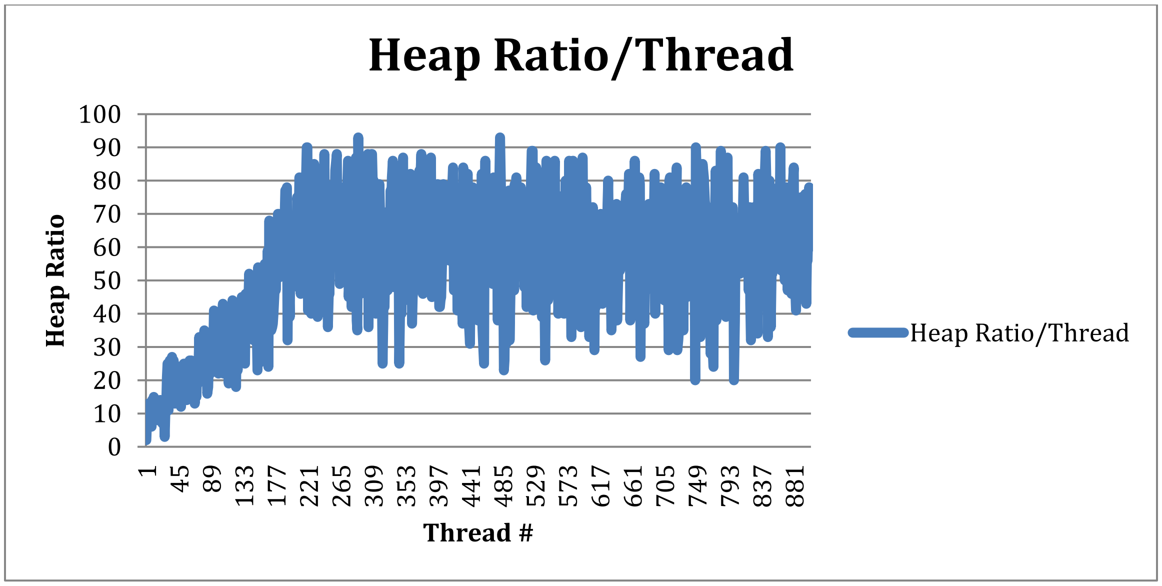


Figure 30 – Heap ratio per thread

STATS Analysis	Value
First Quartile	**42**
Second Quartile (median)	**58**
Third Quartile	**70**

Table 22 –Third run heap ratio data summary

The heap ratio also shows to be more similar to the first run with a slight or insignificant increase resulted from the addition of the recovery components.

CI!t¢9W 7 CONCLUSION

We have built an API that can be attached to a JVM or multiple JVM's hosting web applications heavy on the integration with DMBS, such as an ecommerce or an online store application. The API, Cognitive Performance Application, monitors the JVM performance and learns to detect its bottleneck. The application also has predictive engine that can forecast the next bottleneck occurrence. Once a bottleneck is detected or forecasted, the application uses a temp queue structure to stream down the demand intelligently and only when needed in order to avoid the system from bottlenecking. The application utilizes feed forward ANN, to learn from the performance parameters how to detect the bottleneck. The use of ANN in our case was to emphasize generalization on the design and to demo how the ANN can weight the most determining factors leading to bottleneck in case of ambiguity or in case of absence of domain expert. The application also uses regression models to predict the next bottleneck in order to help the system become a proactive one. We have run number of test to examine the different components of the performance API. We ran a test without invoking the different components of the API with certain hardware specifications managing to bottleneck the system very frequently with an average CPU utilization of 94.15%. Then we ran a second test on the same conditions, but with the addition of two components of the API, the learning engine and the queue engine. We noticed a significant improvement in terms of the system not bottlenecking as frequently as before and with an overall CPU utilization of 87.80%. We concluded with a third test running on the same conditions as the first and the seconds tests, and fully deployed the three components of the application; learning, analytics, and queuing engines. The test shows further improvements with a better average CPU utilization of 84.60%

REFERENCES

[1] Khaled Saleh, “A Queueing Model to Achieve Proper Elasticity for Cloud Cluster Jobs”, Department of Electrical and Computer Engineering, Khalifa University, Sharjah, UAE, 2009

[2] Manfred Dellkrantz, Maria Kihl, and Anders Robertsson,‘ Performance Modeling and Analysis of a Database Server with Write-Heavy Workload’, Department of Automatic Control, Lund University 2 Department of Electrical and Information Technology, Lund University, 2012

[3] Leonid Grinshpan, “Solving Enterprise Applications Performance Puzzles - Queuing Models to the Rescue”, Wiley-IEEE Press, ISBN 978-1-118-06157-2, 2012

[4] Mukarram A. Tahir, “Java Implementation Of Neural Networks”, BookSurge Publishing. ISBN-10: 1419665359; September 27, 2007

[5] International Business Machines Corporation (IBM), “http://www-01.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/landing/wc_welcome.html”, WebSphere Commerce Version 8, 2015

[6] Edward D. Lazowska, John Zahorjan, G. Scott Graham, Kenneth C. Sevcik, ‘Quantitative System Performance’, chapter 1. An Overview of Queueing Network Modelling, 1984by Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

[7] Fabio M Soares, Alan M.F. Souza, “Neural Network Programming with Java,” Packt Publishing, ISBN: 978-1-78588-090-2, January 2016

[8] Adam Gibson, Josh Patterson (August 2017), “Deep Learning A Practitioner's Approach” O'Reilly Media

[9] Mehmed Kantardzic, “Data Mining: Concepts, Models, Methods, and Algorithms 2nd edition,” IEEE Press & John Wiley, ISBN: 978-0-470-89045-5, August 2011.

[10] Yan, Xin, “Linear Regression Analysis: Theory and Computing”, World Scientific Book, pp. 1–2, ISBN 9789812834119, Jun 2009

[11] Jianqing Fan, Irene Gijbels, “Local Polynomial Modelling and Its Applications: Monographs on Statistics and Applied Probability 1st Edition” ISBN-13: 978-0412983214, Chapman & Hall ,1997

[12] John Fox & Sanford Weisberg, ‘Time-Series Regression and Generalized Least Squares in R’, revision: 11, November 2010

[13] Shajulin Benedict, “Performance issues and performance analysis tools for HPC cloud applications: a survey”, Benedict, S. Computing 95: 89. doi:10.1007/s00607-012-0213-0, 2013

[14] Ken Hygh; Ruth Willenborg; Stacy Joines, “Performance Analysis for Java Web Sites”, SBN-13: 978-0201844542, 2003

[15] Henry H. Liu, “Java Performance and Scalability: A Quantitative Approach” CreateSpace Independent Publishing Platform, ISBN-10: 1482348012, February 2, 2013

[16] Edward D. Lazowska, John Zahorjan, G. Scott Graham, Kenneth C. Sevcik, ‘Quantitative System Performance’, chapter 1. An Overview of Queuing Network Modeling, 1984 by Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632.

[17] E. Altman; M. Arnold; R. Bordawekar; R. M. Delmonico; N. Mitchell; P. F. Sweeney, “Observations on tuning a Java enterprise application for performance and scalability”, IBM Journal of Research and Development, 2010

[18] Kuo-Yi Chen, J. Morris Chang, Ting-Wei Hou, “Multithreading in Java: Performance and Scalability on Multicore Systems”, IEEE TRANSACTIONS ON COMPUTERS, VOL. 60, NO. 11, NOVEMBER 2011

[19] R. A. Sciampacone; V. Sundaresan; D. Maier; T. Gray-Donald, “Exploitation of multicore systems in a Java virtual machine ”, IBM Journal of Research and Development archive. Volume 54 Issue 5, DOI:10.1147/JRD.2010.2057911, September 2010

[20] Guillermo L. Taboada, Sabela Ramos, Roberto R. Expósito, Juan Touriño, Ramón Doallo, “Java in the High Performance Computing arena: Research, practice and experience”, www.elsevier.com/locate/scico, 2012

[21] Viktor Mauch;Marcel Kunze;Marius Hillenbrand, “High performance cloud computing”, Future Generation Computer Systems Journal, Vol. 29, August 2013
[22] Salah, K,“A Queueing Model to Achieve Proper Elasticity for Cloud Cluster Jobs,” Services Transactions on Cloud Computing, ISSN 2326-7550, Vol. 1, No. 1, July-September 2013, pp. 53-64
[23] Daniel Jaschob; Michael Riffle, “JobCenter: an open source, cross-platform, and distributed job queue management system optimized for scalability and versatility,” Source Code for Biology and Medicine, DOI 10.1186/1751-0473-7-8, July 2012
[24] Ethem Arkin; Bedir Tekinerdogan; Kayhan M. İmre, “Systematic approach for deriving feasible mappings of parallel algorithms to parallel computing platforms,” Wiley Online Library, DOI: 10.1002/cpe.3821, 22, March 2016
[25] J. Tai, J. Zhang, J. Li, W. Meleis, and N. Mi, "Ara: Adaptive resource allocation for cloud computing environments under bursty workloads," in Proc. IEEE Performance Computing and Communications Conference (IPCCC), pp. 1-8, 2011.
[26] Michele Chambers, Thomas W Dinsmore, “Advanced Analytics Methodologies: Driving Business Value with Analytics,“ Pearson FT Press. ISBN-10: 0-13-349860-3, Sep 22, 2014
[27] David M. Lane, ‘Introduction to Statistics Online Edition’, chapter 14, An Interactive eBook, David Lane, 2013
[28] Phillips, Judah, “Building a Digital Analytics Organization: Create Value by Integrating Analytical Processes, Technology, and People into Business Operations (FT Press Analytics) 1st Edition,” Pearson, ISBN-10: 0133372782, 2013
[29] Cohen, J., Cohen P., West, S.G., & Aiken, L.S, “Applied multiple regression/correlation analysis for the behavioral sciences. (2nd ed.)”, Hillsdale, NJ: Lawrence Erlbaum Associates. ISBN-10: 0805822232, 2003
[30] Draper, N.R.; Smith, H., “Applied Regression Analysis (3rd ed.)”, John Wiley. ISBN 0-471-17082-8, 1998
[31] Osama Abuzeid, Anas Al-Rabadi, Hashem Alkhaldi. “Recent advancements in fractal geometric-based nonlinear time series solutions to the micro-quasistatic thermoviscoelastic creep for rough surfaces in contact,” Mathematical Problems in Engineering, Volume 2011, Article ID 691270, 2011
[32] Priestley, M. B, “Spectral Analysis and Time Series,” Academic Press. ISBN 978-0-12-564901-8, 1981
[33] Bi Ruixiang, "Research on the standard database of performance evaluation of financial expenditure," International Conference on Artificial Intelligence and Education (ICAIE), Hangzhou, pp. 261-265. doi: 10.1109/ICAIE.2010.5641482, 1981
[34] Shuigeng Zhou; Songmao Zhang; George Karypis, “Advanced Data Mining and Applications” 8th International Conference, ADMA 2012, Nanjing, China, December 15-18, 2012
[35] Jason Brownlee, “How to Use Dropout with LSTM Networks for Time Series Forecasting”, http://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting, Deep Learning, April 28, 2017
[36] Wanbo Zheng; Mengchu Zhou; Lei Wu; Yunni Xia; Xin Luo; Shanchen Pang; Qingsheng Zhu; Yanqing Wu, “Percentile Performance Estimation of Unreliable IaaS Clouds and Their Cost-Optimal Capacity Decision,” IEEE Access v5: 2808-2818, 2017
[37] Luis Miralles-Pechuán; Dafne Rosso; Fernando Jiménez; Jose M. García, “A methodology based on Deep Learning for advert value calculation in CPM, CPC and CPA networks”, DOI 10.1007/s00500-016-2468-4, Springer-Verlag Berlin Heidelberg, 2016
[38] Peifeng Niu; Yunpeng Ma; Mengning Li; Shanshan Yan; Guoqiang Li,” A Kind of Parameters Self-adjusting Extreme Learning Machine”, Neural Processing Letters, DOI: 10.1007/s11063-016-9496-z, December 2016
[39] Zhenyun Zhuang; Cuong Tran; Haricharan Ramachandra; Badri Sridharan, “Ensuring High-Performance of Mission-Critical Java Applications in Multi-tenant Cloud Platforms”, Cloud Computing (CLOUD), IEEE 7th International Conference, 2014
[40] Stéphane Guerrier, Roberto Molinari, and Yannick Stebler, “Theoretical Limitations of Allan Variance-based Regression for Time Series Model Estimation”, IEEE SIGNAL PROCESSING LETTERS, VOL. 23, NO. 5, MAY 2016

[41] Yinxiao Huang; Xiaohong Chen; Wei Biao Wu, “Recursive Nonparametric Estimation for Time Series”, IEEE Transactions on Information Theory, DOI: 10.1109/TIT.2013.2292813, 2014
[42] Vlad Mihalcea, “High-Performance Java Persistence 1st edition”, VLAD MIHALCEA, ISBN-973022823X, October 12, 2016
[43] N D Lewis, “Neural Networks for Time Series Forecasting with R” CreateSpace Independent Publishing Platform, ISBN-10: 1544752954, March 27, 2017
[44] Trevor Hastie, Robert Tibshirani, Jerome Friedman, “The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics) 2nd Edition” Springer ISBN-10: 0387848576, 2016
[45] Brendan Gregg, “Systems Performance: Enterprise and the Cloud 1st Edition”, Prentice Hall, October 26, 2013
[46] G. David Garson, “Generalized Linear Models & Generalized Estimating Equations”, Statistical Associates Blue Book Series 26, ASIN: B009434OUQ, 2013
[47] Jason W. Osborne, “Regression & Linear Modeling: Best Practices and Modern Methods 1st” ASIN: B01DWC39GO, SAGE Publications, Inc., April 19, 2016
[48] Judah Phillips, ”Ecommerce Analytics: Analyze and Improve the Impact of Your Digital Strategy (FT Press Analytics)”, ISBN-13: 978-0134177281, Pearson FT Press; 1 edition, April 14, 2016
[49] Cesar Perez Lopez, “SPSS. Predictive Models”, CreateSpace Independent Publishing Platform, July 8, 2013
[50] Brady T. West, Kathleen B. Welch, Andrzej T Galecki, “ Linear Mixed Models: A Practical Guide Using Statistical Software,” hapman and Hall/CRC; 2 edition, July 17, 2014
[51] Naone, Erica, “The New Big Data”, MIT Technology Review. August 22, 2011
[52] Joe Anne Legge, Statistics Canada, “Power from Data! (12-004-X)”, www.statcan.gc.ca/edu/power-pouvoir/ch9/scatter-nuages/5214827-eng.htm, 2011
[53] Richard Lowry, ‘Concepts and Applications of Inferential Statistics’, Richard Lowry 1998-2017
[54] Li B., Sun X., Zhou Y.,” A hierarchical model for regression test selection and cost analysis of Java programs”, APSEC 2010 17th Asia Pacific Software , 2010

APPENDIX A

A.1. Estore Schema Create Statement

```
CREATE DATABASE `estore` /*!40100 DEFAULT CHARACTER SET utf8
*/;
```

A.2. Address Table Create Statement

A.2.1. Description

This table stores the addresses of users or organizations in the WebSphere Commerce system. The addresses can be the members' own addresses or for their friends, associates, or clients, and so on. Some columns here replace columns used in previous versions.

Column Name	Column Type	Description
ADDRESS_ID	BIGINT NOT NULL	Unique reference number, internally generated. This is a primary key, replacing the `SASHNBR` column in the `SHADDR` table used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
ADDRESSTYPE	CHAR(5)	The purpose of the address. Valid values are: S (`shipto`), B (`billto`), and `SB` (both `shipto` and `billto`). If this is unspecified when creating a new address, the business logic will default to `SB`.
MEMBER_ID	BIGINT NOT NULL	Foreign key to the MEMBER table for the member who owns this address.
ADDRBOOK_ID	BIGINT NOT NULL	Foreign key to the ADDRBOOK table for the address book to which this address belongs, replacing the `SASHNBR` column in the `SHADDR` table used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.

ORGUNITNAME	VARCHAR(128)	Name of the organizational unit to which this address information applies, if this is a business address. See comment on ADDRESS.ORGNAME column.
FIELD3	VARCHAR(64)	Compare with the BCFIELD3 column in the BUCONT table.
BILLINGCODE	VARCHAR(17)	A code to identify the shipping or billing information.
BILLINGCODETYPE	CHAR(2)	The code designating the system or method of code structure used for billing.
STATUS	CHAR(1)	Specifies the status of the address. Valid values are P (permanent or current) and T (temporary or historical). This column replaces `SAADRFLAG` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
ORGNAME	VARCHAR(128)	If this is a business address, the name of the organization for this address. This is not the name of the organization that owns the address book with this address. This column replaces `SAREPCOM` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
ISPRIMARY	INTEGER	A user or organization can have multiple addresses of each ADDRESSTYPE and one of them to be designated as primary. For example, one of the shipping addresses can be designated as the primary shipping address. Valid values are 1 (primary address) or 0 (non-primary address).
LASTNAME	VARCHAR(128)	Last name of the person to which this address applies, replacing `SALNAME` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.

PERSONTITLE	VARCHAR(50)	Title of the person to which this address applies, replacing `SATITLE` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite. LDAP uses a length of 50. Valid values are Dr, Prof, Rev, Mr, Mrs, Ms, and N (not provided). The default is N.
FIRSTNAME	VARCHAR(128)	First name of the person to which this address applies, replacing `SAFNAME` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
MIDDLENAME	VARCHAR(128)	Middle name or initials of the person to which this address applies, replacing `SAMNAME` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
BUSINESSTITLE	VARCHAR(128)	The business title. For example, Manager or Chief Executive Officer. LDAP uses a length of 128 characters for title attribute in `ePerson`. Compare with the BUCONT.BCTITLE column.
PHONE1	VARCHAR(32)	The primary phone number, replacing `SAPHONE1` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
FAX1	VARCHAR(32)	The primary fax number 1, replacing `SAFAX` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
PHONE2	VARCHAR(32)	The secondary phone number, replacing `SAPHONE2` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.

ADDRESS1	VARCHAR(100)	Address line 1, replacing `SAADDR1` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
FAX2	VARCHAR(32)	The secondary fax number.
NICKNAME	VARCHAR(254) NOT NULL	The nickname or identifier of the address, replacing `SANICK` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite. Among all the addresses owned by a member, their nicknames must be unique.
ADDRESS2	VARCHAR(50)	Address line 2, replacing `SAADDR2` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
ADDRESS3	VARCHAR(50)	Address line 3, replacing `SAADDR3` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
CITY	VARCHAR(128)	The city, replacing `SACITY` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
STATE	VARCHAR(128)	State or province, replacing `SASTATE` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
COUNTRY	VARCHAR(128)	The country or region, replacing `SACNTRY` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
ZIPCODE	VARCHAR(40)	ZIP or postal code, replacing `SAZIPC` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
EMAIL1	VARCHAR(256)	The primary e-mail address, replacing `SAEMAIL1` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.

EMAIL2	VARCHAR(256)	The secondary e-mail address.
PHONE1TYPE	CHAR(3)	Phone type for the primary phone number, such as `TTY` for a teletypewriter for people who have a hearing impairment or `PHN` for a standard phone. This column replaces `SADPHTYP`.
PHONE2TYPE	CHAR(3)	Phone type for the secondary phone number, such as `TTY` for a teletypewriter for people who have a hearing impairment or `PHN` for a standard phone. This column replaces `SAEPHTYP`.
PUBLISHPHONE1	INTEGER	Specifies whether or not the primary phone number is listed. Valid values are 1 (listed) or 0 (unlisted).
PUBLISHPHONE2	INTEGER	Specifies whether or not the secondary phone number is listed. Valid values are 1 (listed) or 0 (unlisted).
BESTCALLINGTIME	CHAR(1)	The best time to call, replacing `SABTCALL` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite. Valid values are D (daytime) and E (evening).
PACKAGESUPPRESSION	INTEGER	Specifies whether or not to include package inserts. Valid values are 1 (include) or 0 (do not include).
LASTCREATE	TIMESTAMP	The date and time the row was created, replacing `SASTMP` used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.
OFFICEADDRESS	VARCHAR(128)	Desktop delivery or office address associated with a shipping address.

SELFADDRESS	INTEGER NOT NULL DEFAULT 0	Specifies whether or not the address belongs to the member (user or organization). The address that belongs to a member is the one that was entered as part of registration. Valid values are 1 (the address belongs to the member) or 0 (the address does not belong to the user; it may belong to a friend, associate, or relative). A member can have only one permanent self address.
FIELD1	VARCHAR(64)	Customizable. This column replaces the SAFIELD1 column in the SHADDR table (used in previous versions of WebSphere Commerce or WebSphere Commerce Suite) and accepts up to 3 characters.
FIELD2	VARCHAR(64)	Customizable. This column replaces the SAFIELD2 column in the SHADDR table (used in previous versions of WebSphere Commerce or WebSphere Commerce Suite) and accepts up to one character.
TAXGEOCODE	VARCHAR(254)	Tax code based on geographical region, used for integration with Taxware.
SHIPPINGGEOCODE	VARCHAR(254)	Shipping code based on geographical region. This value is used for integration with Taxware.

Address Table

A.2.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`address`;

CREATE TABLE  `estore`.`address` (

  `ADDRESS_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,

  `ADDRESSTYPE` varchar(5) NOT NULL,

  `ISPRIMARY` int(1) unsigned NOT NULL,

  `MEMBER_ID` int(10) unsigned NOT NULL,

  `STATUS` char(1) NOT NULL,

  `LASTNAME` varchar(45) NOT NULL,
  `PERSONTITLE` varchar(45) NOT NULL,
  `FIRSTNAME` varchar(45) NOT NULL,
  `MIDDLENAME` varchar(45) NOT NULL,
  `BUSINESSTITLE` varchar(45) NOT NULL,
  `PHONE1` varchar(32) NOT NULL,
  `FAX1` varchar(32) NOT NULL,
  `ADDRESS1` varchar(256) NOT NULL,
  `ADDRESS2` varchar(256) NOT NULL,
  `ADDRESS3` varchar(256) NOT NULL,
  `CITY` varchar(128) NOT NULL,
  `STATE` varchar(128) NOT NULL,
  `COUNTRY` varchar(45) NOT NULL,
  `ZIPCODE` varchar(45) NOT NULL,
  `EMAIL1` varchar(256) NOT NULL,
  `EMAIL2` varchar(256) NOT NULL,
  PRIMARY KEY (`ADDRESS_ID`),
  KEY `FK_address_1` (`MEMBER_ID`),
  CONSTRAINT `FK_address_1` FOREIGN KEY (`MEMBER_ID`) REFERENCES `member` (`MEMBER_ID`) ON
DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC;
```

A.3. Inventory Table Create Statement

A.3.1. Description

Each row of this table contains a quantity amount representing the inventory for a particular CatalogEntry. The CatalogEntry is available to be shipped from a FulfillmentCenter on behalf of a Store. This table cannot be used in conjunction with Available To Promise (ATP) inventory allocation. It is used only when ATP inventory is not enabled (refer to the INVENTORYSYSTEM column of the STORE table).

Column Name	Column Type	Description
CATENTRY_ID	BIGINT NOT NULL	The CatalogEntry.

QUANTITY	DOUBLE NOT NULL DEFAULT 0	The quantity amount, in units indicated by `QUANTITYMEASURE`.
FFMCENTER_ID	INTEGER NOT NULL	The FulfillmentCenter.
STORE_ID	INTEGER NOT NULL	The Store.
QUANTITYMEASURE	CHAR(16) NOT NULL DEFAULT 'C62'	The unit of measurement for QUANTITY.
INVENTORYFLAGS	INTEGER NOT NULL DEFAULT 0	Bit flags, from low to high order, indicating how QUANTITY is used: 1 = noUpdate. The default UpdateInventory task command does not update QUANTITY. 2 = noCheck. The default CheckInventory and UpdateInventory task commands do not check QUANTITY.
OPTCOUNTER	SMALLINT	Reserved for IBM internal use.

Inventory Table

A.3.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`inventory`;
CREATE TABLE  `estore`.`inventory` (
  `PRODUCT_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `QUANTITY` double NOT NULL,
  PRIMARY KEY (`PRODUCT_ID`),
  CONSTRAINT `FK_inventory_1` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product` (`PRODUCT_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.4. Language Table Create Statement

A.4.1. Description

Each row of this table represents a language. Our Commerce system supports multiple languages and is translated into ten languages by default. Using the predefined ISO codes users can add other supported languages.

Name	Type	Description
LANGUAGE_ID	INTEGER NOT NULL	The language ID (primary key of this table) for each language. Foreign key to the tables that contain language-dependent information. The following is the list of language components (language ID codes) of the locale: -1 = English (US) -2 = French -3 = German -4 = Italian -5 = Spanish -6 = Brazilian Portuguese -7 = Simplified Chinese -8 = Traditional Chinese -9 = Korean

LOCALENAME	CHAR (16) NOT NULL	A Java locale used to represent a political, geographical, or cultural region that has a distinct language and customs. The following is the list of locale used for formatting:
		en_US = United States
		fr_FR = France
		de_DE = Germany
		it_IT = Italy
		es_ES = Spain
		pt_BR = Brazil
		zh_CN = China
		zh_TW = Taiwan
		ko_KR = Korea
		ja_JP = Japan
		ru_RU = Russian
		ro_RO = Romanian
		pl_PL = Polish

LANGUAGE	CHAR (5)	Language component of the locale.
		NOTE: A value must be entered for this column:
		en = English (US)
		fr = French
		de = German
		it = Italian
		es = Spanish
		pt = Brazilian Portuguese
		zh = Simplified Chinese
		zh = Traditional Chinese
		ko = Korean
		ja = Japanese
		ru = Russian
		ro = Romanian
		pl = Polish
		
		To choose another language component, refer to the existing ISO codes.

COUNTRY	CHAR (5)	Country or region component of the locale.
		NOTE: A value must be entered for this column:
		US = United States
		FR = France
		DE = Germany
		IT = Italy
		ES = Spain
		BR = Brazil
		CN = China
		TW = Taiwan
		KR = Korea
		JP = Japan
		RU = Russia
		RO = Romania
		PL = Poland
		
		To choose another country or region, refer to the existing ISO codes.

VARIANT	CHAR (10)	Variant component of the locale. Used to specify the locale encoding character set
ENCODING	VARCHAR (32)	The character encoding value that the browser uses to display the page in the selected language.

MIMECHARSET	VARCHAR (32)	The MIME character encoding value that the notification system uses to encode a message for the selected language.
OPTCOUNTER	SMALLINT	The optimistic concurrency control counter for the table.

Language Table

A.4.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`language`;

CREATE TABLE  `estore`.`language` (

 `LANGUAGE_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,

 `LOCALENAME` varchar(45) NOT NULL,

 `LANGUAGE` varchar(45) NOT NULL,

 `COUNTRY` varchar(45) NOT NULL,

 `VARIANT` varchar(45) NOT NULL,

 PRIMARY KEY (`LANGUAGE_ID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.5. Listprice Table Create Statement

A.5.1. Description

Each row of this table represents a ListPrice in a particular currency for each Catalog Entry

Name	Type	Description
CATENTRY_ID	BIGINT NOT NULL	The CatalogEntry.
CURRENCY	CHAR (3) NOT NULL	The Currency of the ListPrice. This is a currency code as per ISO 4217 standards.
LISTPRICE	DECIMAL (20,5) NOT NULL	The amount of the ListPrice.
OID	VARCHAR (64)	Reserved for IBM internal use.
OPTCOUNTER	SMALLINT	The optimistic concurrency control counter for the table.

Listprice Table

A.5.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`listprice`;

CREATE TABLE  `estore`.`listprice` (

 `PRODUCT_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
 `CURRENCY` varchar(45) NOT NULL,
 `LISTPRICE` decimal(20,5) NOT NULL,
 PRIMARY KEY (`PRODUCT_ID`),
 CONSTRAINT `FK_listprice_1` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product` (`PRODUCT_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

A.6. Member Table Create Statement

A.6.1. Description

Stores the list of members (participants) of the WebSphere Commerce system. A member is either a user, an organizational entity or a member group.

Name	Type	Description
MEMBER_ID	BIGINT NOT NULL	ID for the member, a unique primary key for this table.

TYPE	CHAR (3) NOT NULL	The type of member as follows:
		O = OrgEntity
		U = User
		G = MemberGroup

STATE	INTEGER	The registration approval status as follows: 0 = pending approval 1 = approved 2 = rejected 3 = pending email activation Null = approval is not applicable (approved)
OPTCOUNTER	SMALLINT	The optimistic concurrency control counter for the table.

Member Table

A.6.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`member`;
CREATE TABLE  `estore`.`member` (
 `MEMBER_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
 `TYPE` varchar(45) NOT NULL,
 `state` varchar(45) NOT NULL,
 PRIMARY KEY (`MEMBER_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.7. OrderItems Table Create Statement

A.7.1. Description

Each row of this table represents an order item in an order

Name	Type	Description
ORDERITEMS_ID	BIGINT NOT NULL	Generated unique key.
STOREENT_ID	INTEGER NOT NULL	The store entity the order (this order item is part of) is part of. This is normally a store unless STATUS is Q, in which case it is normally a store group.
ORDERS_ID	BIGINT NOT NULL	The order of which this order item is part.
TERMCOND_ID	BIGINT	The TermAndCondition, if known, that determined the price for this order item.
TRADING_ID	BIGINT	The TradingAgreement, if known, that determines the TermAndCondition objects (including how the price is determined) that apply to this order item.
ITEMSPC_ID	BIGINT	The specified item to be allocated from available inventory and shipped to the customer.
CATENTRY_ID	BIGINT	The catalog entry, if any, of the product being purchased.
PARTNUM	VARCHAR (64)	The part number of the catalog entry(CATENTRY.PARTNUMBER) for the product.
SHIPMODE_ID	INTEGER	The shipping mode, if still known.
FFMCENTER_ID	INTEGER	The fulfillment center, if known, from which the product will ship.
MEMBER_ID	BIGINT NOT NULL	The customer of the order item (which is the same as the customer of the order).
ADDRESS_ID	BIGINT	The shipping `address`, if any, for this order item.
ALLOCADDRESS_ID	BIGINT	The shipping address used when inventory for this order item was allocated or backordered.

PRICE	DECIMAL (20,5)	The price for the nominal quantity of the product (CATENTSHIP.NOMINALQUANTITY).
LINEITEMTYPE	CHAR (4)	If specified, indicates the type of the order item. ALT = the order item represents an alternative item (might not be exactly what the customer requested).
STATUS	CHAR (1) NOT NULL	The status for the order item. It may not be the same as the status in the order.
OUTPUTQ_ID	BIGINT	Reserved for IBM internal use.
INVENTORYSTATUS	CHAR (4) NOT NULL DEFAULT 'NALC'	The allocation status of inventory for this order item: NALC Inventory is not allocated nor on back-order. BO Inventory is on back-order. ALLC Inventory is allocated. FUL Inventory has been released for fulfillment. AVL Inventory is available.
LASTCREATE	TIMESTAMP	The time this order item was created.
LASTUPDATE	TIMESTAMP	The most recent time this order item was updated. Changing inventory allocation related information does not cause this timestamp to be updated (refer to the `LASTALLOCUPDATE` column).
FULFILLMENTSTATUS	CHAR (4) NOT NULL DEFAULT 'INT'	The fulfillment status of the order item: INT = not yet released for fulfillment. OUT = released for fulfillment. SHIP = shipment confirmed. HOLD = a temporary status between INT and OUT status.

LASTALLOCUPDATE	TIMESTAMP	The most recent time inventory was checked (for unallocated order items), allocated, or backordered, for this order item.
OFFER_ID	BIGINT	The offer, if any, and if it still exists, from which PRICE was obtained.
TIMERELEASED	TIMESTAMP	The time this order item was released for fulfillment.
TIMESHIPPED	TIMESTAMP	The time this order item was manifested for shipment.
CURRENCY	CHAR (10)	The currency of order item monetary amounts other than `BASEPRICE`. This is the same as the currency of the order, ORDERS.CURRENCY. This is a currency code according to ISO 4217 standards.
COMMENTS	VARCHAR (254)	Comments from the customer, such as a greeting for a gift.
TOTALPRODUCT	DECIMAL (20,5) DEFAULT 0	PRICE times QUANTITY.
QUANTITY	DOUBLE NOT NULL	The result of multiplying QUANTITY by CATENTSHIP.NOMINALQUANTITY must be a multiple of CATENTSHIP.QUANTITYMULTIPLE. And it represents the actual quantity being purchased, in the unit of measurement specified by CATENTSHIP.QUANTITYMEASURE.
TAXAMOUNT	DECIMAL (20,5)	The total sales taxes associated with this order item, in the currency specified by CURRENCY.
TOTALADJUSTMENT	DECIMAL (20,5) DEFAULT 0	The total of the monetary amounts of the order item adjustments for this order item, in the currency specified by CURRENCY. This column also includes all kinds of shipping charge adjustments like discount, coupon, shipping adjustment and surcharge.

SHIPTAXAMOUNT	DECIMAL (20,5)	The total shipping taxes associated with this order item, in the currency specified by CURRENCY.
ESTAVAILTIME	TIMESTAMP	An estimate of when sufficient inventory will be available to fulfill this order item. This estimate does not include the shipping offset.
FIELD1	INTEGER	Customizable.
DESCRIPTION	VARCHAR (254)	A mnemonic description of the order Item, suitable for display to the customer. This field is usually NULL when `CATENTRY_ID` is not NULL, since in that case the CatalogEntry description can be displayed.
FIELD2	VARCHAR (254)	Customizable.
ALLOCATIONGROUP	BIGINT	Reserved for IBM internal use.
SHIPCHARGE	DECIMAL (20,5)	The base shipping charge associated with the order item, in the currency specified by CURRENCY, it is the shipping charge before any adjustments. The shipping charge adjustment will be persisted in the `ORDERITEMS.TOTALADJUSTMENT` column with other adjustments including discount and surcharge. The total shipping charge is the sum of base shipping charge in `ORDERITEMS.SHIPCHARGE` column and the shipping charge adjustment in `ORDERITEMS.TOTALADJUSTMENT` column.
BASEPRICE	DECIMAL (20,5)	If PRICE was converted from a currency different from the order item currency, `BASEPRICE` is the price that was converted to determine the PRICE.
BASECURRENCY	CHAR (3)	The currency of `BASEPRICE`.
TRACKNUMBER	VARCHAR (64)	Reserved for IBM internal use.
TRACKDATE	TIMESTAMP	Reserved for IBM internal use.

PREPAREFLAGS	INTEGER NOT NULL DEFAULT 0	Contains the following bit flags indicating special processing associated with this order item: 1 generated The order item was generated during a previous execution of the OrderPrepare command. The next time the OrderPrepare command is run, it first removes all generated order items so that they can be re-generated if and as appddcable. 2 priceOverride The price of the order item has been manually entered and will not be changed by customer commands. 4 fulfillmentCenterOverride The fulfillment center has been manually specified and will not be changed by customer commands. 8 directCalculationCodeAttachment CalculationCodes may be directly attached to the order item. The default CalculationCodeCombineMethod will not look for direct attachments unless this flag is true.

		16 shippingChargeByCarrier The contract for this order item indicates that no shipping charge will be calculated by WebSphere Commerce. It may be calculated and charged by the carrier on fulfillment.
		32 quotation The order item was obtained from a quotation. The price will not be automatically refreshed by customer commands.
		64 notConfigured Price lookup and inventory allocation for this order item is not done using the component items found in the OICOMPddST table. This flag does not need to be set for order items whose CONFIGURATIONID column value is null.
		128 autoAdd This order item, such as a free gift, was automatically added to the order by the OrderCalculate command. This flag is for information-display purposes only and does not affect price calculations.

		256 hasPersonalizationAttributes The order item contains personalization attributes. 512 skipRepricing The order item was marked as not to be repriced. For example, it is set as skipRepricing when a quote is generated into an order if the quote has not expired. 1024 shippingAdjusted The order items shipping charge is manually adjusted. 2048 priceRefreshed The order items price has already been refreshed.
CORRELATIONGROUP	BIGINT	Normally this is the same as `ORDERITEMS_ID`, except: 1. When an order item is split by the `AllocateInventory` task command, the newly created `OrderItem` inherits the `CORRELATIONGROUP` value from the original order item. 2. when the `PREPAREFLAGS` column indicates "quotation", the OrderItem inherits the `CORRELATIONGROUP` value from the corresponding OrderItem in the parent order.

PROMISEDAVAILTIME	TIMESTAMP	When an order is placed (using the OrderProcess command), this would be set to EstAvailTime. After that it would normally not be updated, although a CSR could manually update this to reflect a verbal commitment made to the customer.
SHIPPINGOFFSET	INTEGER NOT NULL DEFAULT 0	An estimate of how many seconds it will take to ship this item once the order is placed and inventory has been allocated.
NEEDEDQUANTITY	INTEGER NOT NULL DEFAULT 0	Quantity needed for fulfillment. If CATENTRY is not NULL, this is QUANTITY times CATENTSHIP.NOMINALQUANTITY, converted from CATENTSHIP.QUANTITYMEASURE to BASEITEM.QUANTITYMEASURE, divided by BASEITEM.QUANTITYMULTIPLE and rounded to the nearest integer.
ALLOCQUANTITY	INTEGER NOT NULL DEFAULT 0	Quantity allocated or backordered for this order item. The quantity in BASEITEM.QUANTITYMEASURE units can be calculated by multiplying this value by BASEITEM.QUANTITYMULTIPLE, for the base item of the specified item indicated by ITEMSPC_ID.
ALLOCFFMC_ID	INTEGER	The fulfillment center from which inventory for this order item is allocated or backordered.
ORDRELEASENUM	INTEGER	The associated order release, if any.
CONFIGURATIONID	VARCHAR (128)	The identifier that is provided by an external product configurator. This identifier represents a list of order item components that are stored in the OICOMPLIST table.

SUPPLIERDATA	VARCHAR (254)	This column is not populated or used by WebSphere Commerce. The value for this column can be generated and used by the external system that originates an order. This value can be returned with a quotation, and sent when an order is placed on an external system. For example, this column can be used to contain a supplier distribution center ID. This field is not supported within the Order Noun for component services.
SUPPLIERPARTNUMBER	VARCHAR (254)	The supplier part number, if known. Suitable for display to the customer. This field is not supported within the Order Noun for component services.
AVAILQUANTITY	INTEGER	If specified, indicates the quantity available for purchase.
ISEXPEDITED	CHAR (1) NOT NULL DEFAULT 'N'	Indicates that the item should receive expedited handling in fulfillment. Y indicates that it should receive expedited handling. N indicates that it should not (this is the default).
OPTCOUNTER	SMALLINT	The optimistic concurrency control counter for the table.
REQUESTEDSHIPDATE	TIMESTAMP	The ship date that has been requested for the order item. When no date (null) is specified, the ship date will be whenever the item is available to be shipped. When a date is specified, the ship date will be no sooner than the requested date. This is because a backorder could delay the ship date beyond the requested date. See the definition of `ALLOCATIONOFFSET` and `MAXFOOFFSET` in the STORE table.
TIECODE	SMALLINT	TieCode used for items that are shipping at the same time (Ship Together).

OrderItem Table

A.7.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`orderitems`;
```

```
CREATE TABLE `estore`.`orderitems` (
 `ORDERITEMS_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
 `ORDERS_ID` int(10) unsigned NOT NULL,
 `PRODUCT_ID` int(10) unsigned NOT NULL,
 `MEMBER_ID` int(10) unsigned NOT NULL,
 `ADDRESS_ID` int(10) unsigned NOT NULL,
 `PRICE` decimal(20,5) NOT NULL,
 `QUANTITY` double NOT NULL,
 `TOTALPRODUCT` decimal(20,5) NOT NULL,
 `DESCRIPTION` varchar(45) NOT NULL,
 PRIMARY KEY (`ORDERITEMS_ID`),
 KEY `FK_ORDERITEMS_1` (`ORDERS_ID`),
 KEY `FK_orderitems_2` (`PRODUCT_ID`),
 KEY `FK_orderitems_3` (`MEMBER_ID`),
 KEY `FK_orderitems_4` (`ADDRESS_ID`),
 CONSTRAINT `FK_ORDERITEMS_1` FOREIGN KEY (`ORDERS_ID`) REFERENCES `orders` (`ORDERS_ID`),
 CONSTRAINT `FK_orderitems_2` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product`
(`PRODUCT_ID`),
 CONSTRAINT `FK_orderitems_3` FOREIGN KEY (`MEMBER_ID`) REFERENCES `member` (`MEMBER_ID`),
 CONSTRAINT `FK_orderitems_4` FOREIGN KEY (`ADDRESS_ID`) REFERENCES `address` (`ADDRESS_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.8. Order Table Create Statement

A.8.1. Description

Each row in this table represents an order in a store

Name	Type	Description
ORDERS_ID	BIGINT NOT NULL	Generated unique key.
ORMORDER	CHAR (30)	A merchant-assigned order reference number, if any.
ORGENTITY_ID	BIGINT	The immediate parent organization ID of the creator.
TOTALPRODUCT	DECIMAL (20,5) DEFAULT 0	The sum of ORDERITEMS.TOTALPRODUCT for the OrderItems in the Order.

TOTALTAX	DECIMAL (20,5) DEFAULT 0	The sum of ORDERITEMS.TAXAMOUNT for the OrderItems in the Order.
TOTALSHIPPING	DECIMAL (20,5) DEFAULT 0	The sum of ORDERITEMS.SHIPCHARGE for the OrderItems in the Order.
TOTALTAXSHIPPING	DECIMAL (20,5) DEFAULT 0	The sum of ORDERITEMS.SHIPTAXAMOUNT for the OrderItems in the Order.
DESCRIPTION	VARCHAR (254)	A mnemonic description of the order, entered by the customer, suitable for display to the customer.
STOREENT_ID	INTEGER NOT NULL	The store entity the order is part of. This is normally a store unless STATUS is Q, in which case it is normally a store group.
CURRENCY	CHAR (10)	The currency for monetary amounts associated with this order. This is the currency code according to ISO 4217 standards.
LOCKED	CHAR (1)	Reserved for IBM internal use.
TIMEPLACED	TIMESTAMP	The time this order was processed by the OrderProcess command.
LASTUPDATE	TIMESTAMP	The time this order was most recently updated.
SEQUENCE	DOUBLE NOT NULL DEFAULT 0	Can be used by a user interface to control the sequence of orders in a list.
STATUS	VARCHAR (3)	The status of the order. This is similar to order states. For specific values of status and their descriptions, refer to the order states page, but omit states 'Y' and 'Z'.
MEMBER_ID	BIGINT NOT NULL	The customer that placed the order.
FIELD1	INTEGER	Customizable.
ADDRESS_ID	BIGINT	This column is deprecated. In versions WC 5.6.1 and earlier, this column contained the billing address. In version WC 6.0 and later, the billing address is part of payment data and is stored in PPCEXTDATA table as encrypted data.

FIELD2	DECIMAL (20,5)	Customizable.
PROVIDERORDERNUM	INTEGER	Reserved for IBM internal use.
SHIPASCOMPLETE	CHAR (1) NOT NULL DEFAULT 'Y'	Reserved for IBM internal use.
FIELD3	VARCHAR (254)	Customizable.
TOTALADJUSTMENT	DECIMAL (20,5) DEFAULT 0	The sum of ORDERITEMS.TOTALADJUSTMENT for the order items in the order. This column also includes all kinds of shipping charge adjustments like discount, coupon, shipping adjustment and surcharge.
ORDCHNLTYP_ID	BIGINT	Reserved for IBM internal use.
COMMENTS	VARCHAR (254)	Comments from the customer.
NOTIFICATIONID	BIGINT	Notification identifier referring to the rows in the NOTIFY table that store notification attributes. These attributes override the defaults for notifications related to this order.
TYPE	CHAR (3)	For an order, TYPE indicates whether it is a regular order (ORD), private requisition list order (PRL), shareable requisition list order (SRL), standing order (STD), quotation order (QUT), quote (QOT), profile order (QUK), recurring order (REC), or subscription (SUB).
OPTCOUNTER	SMALLINT	The optimistic concurrency control counter for the table.
EDITOR_ID	BIGINT	Stores the ID of the person editing the order.
BUSCHN_ID	INTEGER	The Business Channel ID of the order.
SOURCEID	BIGINT	This field indicates the sources from which this order came, if any. This field is relevant for orders that were generated from a previously saved quote, for example. In that situation, the QuoteId would be saved here.

EXPIREDATE	TIMESTAMP	This entry is only relevant when the row refers to a quote, type "QOT". This field then indicates the expiration date for that quote.
BLOCKED	SMALLINT DEFAULT 0	Indicates if this order has a block placed against it or not.
OPSYSTEM_ID	INTEGER	This field indicates the Order Processing system for the order.
TRANSFERSTATUS	SMALLINT	Order transferring status: 0, it is in the initial status. 1, it is in the transferring status. 2, it is in the transferred status. null, the same as 0.
BUYERPO_ID	BIGINT	The Buyer PO Number Id related to an order

Order Table

A.8.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`orders`;

CREATE TABLE  `estore`.`orders` (

  `ORDERS_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,

  `TOTALPRODUCT` decimal(20,5) NOT NULL,

  `TOTALTAX` decimal(20,5) NOT NULL,

  `TOTALSHIPPING` decimal(20,5) NOT NULL,

  `DESCRIPTION` varchar(45) NOT NULL,

  `CURRENCY` char(10) NOT NULL,

  `TIMEPLACED` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

  `STATUS` char(3) NOT NULL,

  `BILLING_ADDRESS_ID` int(10) unsigned NOT NULL,

  `TOTALADJUSTMENT` decimal(20,5) NOT NULL,

  `MEMBER_ID` int(10) unsigned NOT NULL,

  PRIMARY KEY (`ORDERS_ID`),

  KEY `FK_ORDERS_1` (`MEMBER_ID`),

  KEY `FK_orders_2` (`BILLING_ADDRESS_ID`),

  CONSTRAINT `FK_ORDERS_1` FOREIGN KEY (`MEMBER_ID`) REFERENCES `member` (`MEMBER_ID`),

  CONSTRAINT `FK_orders_2` FOREIGN KEY (`BILLING_ADDRESS_ID`) REFERENCES `address`
(`ADDRESS_ID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.9. Product Table Create Statement

A.9.1. Description

This table holds the information related to a catalog entry. Examples of catalog entries include products, items, packages, and bundles.

Name	Type	Description
CATENTRY_ID	BIGINT NOT NULL	The internal reference number of the catalog entry.
MEMBER_ID	BIGINT NOT NULL	The reference number that identifies the owner of the catalog entry. Along with the PARTNUMBER, these columns are a unique index.
ITEMSPC_ID	BIGINT	The specified item that this catalog entry relates to. This column should only be populated for catalog entries that are of type "ItemBean", "PackageBean", or "DynamicKitBean". Specified items are used for fulfillment.
CATENTTYPE_ID	CHAR (16) NOT NULL	Identifies the type of catalog entry. Foreign key to the CATENTTYPE table. The supported default types are: ProductBean, ItemBean, PackageBean, BundleBean and DynamicKitBean.
PARTNUMBER	VARCHAR (64) NOT NULL	The reference number that identifies the part number of the catalog entry. Along with the MEMBER_ID, these columns are a unique index.
MFPARTNUMBER	VARCHAR (64)	The part number used by the manufacturer to identify this catalog entry.
MFNAME	VARCHAR (64)	The name of the manufacturer of this catalog entry.
MARKFORDELETE	INTEGER NOT NULL	Indicates if this catalog entry has been marked for deletion: 0 = No. 1 = Yes.
URL	VARCHAR (254)	The URL to this catalog entry, which can be used as a download URL for soft goods.
FIELD1	INTEGER	Customizable.

FIELD2	INTEGER	Customizable.
LASTUPDATE	TIMESTAMP	Indicates the last time the catalog entry was updated.
FIELD3	DECIMAL (20,5)	Customizable.
ONSPECIAL	INTEGER	This flag identifies if this catalog entry is on special.
ONAUCTION	INTEGER	This flag identifies if this catalog entry is on auction.
FIELD4	VARCHAR (254)	Customizable.
FIELD5	VARCHAR (254)	Customizable.
BUYABLE	INTEGER	Indicates whether this catalog entry can be purchased individually: 1=yes and 0=no.
OID	VARCHAR (64)	Reserved for IBM internal use.
BASEITEM_ID	BIGINT	The base item to which this catalog entry relates. This column should only be populated for catalog entries that are of type ProductBean, PackageBean or DynamicKitBean. Base items are used for fulfillment.
STATE	CHAR (1) DEFAULT '1'	Reserved for IBM internal use.
STARTDATE	TIMESTAMP	The date when this catalog entry is introduced. This column is for your interpretation and information only.
ENDDATE	TIMESTAMP	The date when this catalog entry is withdrawn. This column is for your interpretation and information only.
RANK	DOUBLE	Reserved for IBM internal use.
OPTCOUNTER	SMALLINT	The optimistic concurrency control counter for the table.
AVAILABILITYDATE	TIMESTAMP	The date that the product is available to customers for purchase. This column is for your interpretation and information only.

LASTORDERDATE	TIMESTAMP	The final date that the product is available for order. This is the latest date a customer can order the product. This column is for your interpretation and information only.
ENDOFSERVICEDATE	TIMESTAMP	The date when the product is no longer in service. For example, this date can be the date that a software manufacturer stops providing upgrades for a version of their product. This column is for your interpretation and information only.
DISCONTINUEDATE	TIMESTAMP	The date when the product is discontinued. For example, this date can be the date the manufacturer stops producing the product, or the date your store stops selling the product. It is recommended that this column be used consistently to avoid confusion. This column is for your interpretation and information only.
UP_MFNAME	VARCHAR (64)	The equivalent value of the MFNAME column in upper case characters. This column is used only for DB2 (LUW) database-types to enhance performance of text-based searches issued from Management Center.
UP_MFPARTNUMBER	VARCHAR (64)	The equivalent value of the MFPARTNUMBER column in upper case characters. This column is used only for DB2 (LUW) database-types to enhance performance of text-based searches issued from Management Center.
UP_PARTNUMBER	VARCHAR (64)	The equivalent value of the PARTNUMBER column in upper case characters. This column is used only for DB2 (LUW) database-types to enhance performance of text-based searches issued from Management Center.

Product Table

A.9.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`product`;
```

```
CREATE TABLE `estore`.`product` (
 `PRODUCT_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
 `PARTNUMBER` varchar(45) NOT NULL,
 `MFNAME` varchar(45) NOT NULL,
 `URL` varchar(45) NOT NULL,
 `BUYABLE` int(1) unsigned NOT NULL,
 `AVAILABILITYDATE` datetime NOT NULL,
 `ENDOFSERVICEDATE` datetime NOT NULL,
 PRIMARY KEY (`PRODUCT_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.10. ProductDesc Table Create Statement

A.10.1. Description

This table holds language-dependent information related to a catalog entry

Name	Type	Description
CATENTRY_ID	BIGINT NOT NULL	The internal reference number that indicates to which catalog entry this language-specific information relates.
LANGUAGE_ID	INTEGER NOT NULL	The identifier of the language. For a list of language components, see the LANGUAGE table.
NAME	CHAR (128)	The language-dependent name of this catalog entry.
SHORTDESCRIPTION	VARCHAR (254)	A short description of this catalog entry.
LONGDESCRIPTION	CLOB (1000000)	A long description of this catalog entry.
THUMBNAIL	VARCHAR (254)	The thumbnail image path of this catalog entry.
AUXDESCRIPTION1	VARCHAR (4000)	Additional description for this catalog entry.
FULLIMAGE	VARCHAR (254)	The full image path of this catalog entry.
AUXDESCRIPTION2	VARCHAR (4000)	Additional description for this catalog entry.
XMLDETAIL	CLOB (1000000)	Reserved for IBM internal use.

AVAILABLE	INTEGER NOT NULL	Indicates the length of time to availability of this catalog entry. Do not use this column to indicate inventory levels.
PUBLISHED	INTEGER NOT NULL	Indicates whether this catalog entry should be displayed for the language indicated by LANGUAGE_ID: 0 Catalog entry should not be displayed 1 catalog entry should be displayed
AVAILABILITYDATE	TIMESTAMP	Date this catalog entry becomes available.
KEYWORD	VARCHAR (254)	A keyword used for searching.
OPTCOUNTER	SMALLINT	The optimistic concurrency control counter for the table.
UP_NAME	CHAR (128)	The equivalent value of the NAME column in upper case characters. This column is used only for DB2 (LUW) database-types to enhance performance of text-based searches issued from Management Center.

ProductDesc Table

A.10.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`productdesc`;

CREATE TABLE  `estore`.`productdesc` (

  `PRODUCT_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,

  `LANGUAGE_ID` varchar(45) NOT NULL,

  `NAME` varchar(128) NOT NULL,

  `SHORTDESCRIPTION` varchar(254) NOT NULL,

  `LONGDESCRIPTION` varchar(400) NOT NULL,

  `THUMBNAIL` varchar(254) NOT NULL,

  `FULLIMAGE` varchar(254) NOT NULL,

  `KEYWORD` varchar(254) NOT NULL,

  PRIMARY KEY (`PRODUCT_ID`),

  CONSTRAINT `FK_productdesc_1` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product`
(`PRODUCT_ID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

APPENDIX B

B.1. Abbreviation

Abbreviation	Definition
JVM	A Java virtual machine (JVM) is an abstract computing machine that enables a computer to run a Java program
DBMS	A database-management system (DBMS) is a computer-software application that interacts with end-users, other applications, and the database itself to capture and analyze data. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases.
CPU	is the abbreviation for central processing unit. Sometimes referred to simply as the central processor, but more commonly called processor, the CPU is the brains of the computer where most calculations take place.
AI	Artificial intelligence (AI, also machine intelligence, MI) is intelligent behavior by machines, rather than the natural intelligence (NI) of humans and other animals.
FCFS	First-come, first-served (FCFS), this principle states that customers are served one at a time and that the customer that has been waiting the longest is served first.
ANN	An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.
CPA	Cognitive Performance Application
JDBC	Java Database Connectivity (JDBC) is an application-programming interface (API) for the programming language Java, which defines how a client may access a database. It is Java based data access technology and used for Java database connectivity. It is part of the Java Standard Edition platform, from Oracle Corporation.

SQL	a standard language for storing, manipulating and retrieving data in databases
UML	The Unified Modeling Language (*UML*) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system

CURRICULUM VITA

Name: Shade El-Hadik

Address:

- 1618 Grayson Lakes Blvd.

  Katy, TX. 77494

Education:

- B.S. Computer Science
  American University in Cairo
  1999-2003
- M.S. Computer Engineering and Computer Science
  University of Louisville
  2004-2004

Work Experience:

- Accenture
  - IBM Watson eCommerce Architect 2017-Present
- IBM
  - IBM eCommerce Architect 2015-2017
- Accenture
  - IBM eCommerce Architect 2012-2015
- Academy Outdoors & Sports
  - IBM eCommerce Architect 2011-2012
- IBM
  - IT Architect 2007-2011