Evaluating the QOS for E-Commerce Architecture Using Proposed WQM Algorithm

Riktesh Srivastava
Information Systems, Skyline University College, University City of Sharjah, Sharjah, UAE

Abstract
Impulsive demands posed by unexpected flows of requests from users on Internet have made it exceedingly challenging for E-Commerce to offer anticipated Quality of Service (QoS). In order to provide stable service, it is important for electronic commerce architecture to recognize the QoS (Quality of Service), such as, the response time for requests, delay in accessing the services offered by servers implemented in E-Commerce architecture, and the number of requests waiting in the queue and thereby identifying the strategies to service the requests promptly. The correct and swift QoS can prominently aid to build and operate electronic commerce architecture more proficiently.

In this paper, design and implementation of a QoS-enabled WQM (Web Queue Model) Algorithm is proposed, which uses the queuing models to prioritize the requests from different classes of users. The proposed algorithm takes the requests from clients, and uses three different set of algorithms simultaneously, namely, Classifier Algorithm, Priority Queue Algorithm and Final Queue Algorithm, to improve the QoS for requests. The comprehensive performance of the WQM Algorithm is depicted using various Queuing models and the results are shown accordingly. The queue models which are implemented using the simulation study include the comparative study of the WQM models using M/M/n/K and G/G/n/K models.

1. INTRODUCTION
The Quality of Service (QoS) for any electronic commerce applications plays an essential part in appealing and holding consumers. The workload experienced by these applications tends to vary in a very dynamic way. The complication of the Electronic Commerce architecture shared with the huge short-terms deviations of the workload calls for QoS study for EC configuration. This paper introduces Web Queue Model (WQM), to evaluate the work load of the n-Tier EC architecture implemented in Hybrid Cloud to dynamically monitor and tune so that preferred QoS levels are accomplished. The complete WQM algorithm implemented using Java programming is actually combination of Classifier Algorithm, Priority Queue Algorithm and Final Queue Algorithm. These algorithms execute in concurrency for every requests being received and defines the class of queue. By doing so, different classes of request are generated, which forms the base of WQM algorithm.

The study of web request classification is quite different from text classification, as mentioned in [1]-[7]. These differentiations are:

1) Text classifications are typically performed on structured corpora with well-controlled authoring styles [8], whereas web request classification does not have such property.
2) Secondly, web requests are semi-structured documents, which may have embedded information for depiction; such markup is typically not required for web request classification algorithms.
3) Finally, web requests always exist within a hypertext, whereas, this property is not present in typical text classification problems.
The Classifier algorithm thus designed is based on classification of customers’ requests rather than the text classification of customer text. The classification algorithm designed followed the functional point analysis technique. Priority Queue Algorithm is combination of providing the web requests to the appropriate queue and then allocating the Data Center for each class of queue. Final Queue Algorithm is implementation of processing the web requests using either M/M/n/K or G/G/n/K algorithm. Final Queue Algorithm founds its base on the study performed by [9]-[12].

2. ARCHITECTURE FOR WQM IMPLEMENTATION

The complete architecture of the Electronic Commerce architecture for implementing the proposed WQM algorithm includes 4 layers:

1) Web Request Management Server (WRMS)
2) Three Data Centers (DC) with 5 Virtual Machines (VM)
3) Application Servers (AS)
4) Database Servers (DS)

The network diagram demonstrated below intricate the structure for any Electronic Commerce architecture for Hybrid cloud. The point worth noting here is that Web Request Management Server and Data Centers are implemented in the private cloud whereas Application Servers and Data Base Servers are implemented in the public cloud. The Web Request Management Server is the Web Server, implemented using Apache Tomcat, whereas, Data center includes VMware.

![Figure 1: E-Commerce architecture for WQM Algorithm](image)

2.1. Classifier Algorithm

Classifier Algorithm operates at WRMS level with the aim to identify the request type and then classify them into three classes, namely, CDC1, CDC2, and CDC3. Each of these classes has to be processed by any one of the Data Centers with 5 Virtual Machines each. The classification of the request is based on grain size. In WQM algorithm, the three grain sizes are:

1) Fine Grain: If the request to be processed takes less than 5 seconds, based on the request type.
2) Medium Grain: If the request to be processed takes between 5 to 10 seconds, based on the request type.
3) Coarse Grain: If the request to be processed takes more than 10 seconds, based on the request type.

2.2 Priority Queue Algorithm

The Priority Queue Algorithm is designed using Weighted Round Robin (WRR) Technique. In WRR queue algorithm, the requests are categorized into different classes (CDC1, CDC2, CDC3) and then each queue are assigned to the specific queue. The algorithm also addresses the problem of starvation of each request to the final queue algorithm to be processed. The priority queue algorithm is mentioned in Figure 2 mentioned below:
2.3 Final Queue Algorithm

The main experimentation for the proposed WQM algorithm is the final queue algorithm. After the queue is formed to be processed, all the requests must be organized in the final queue. The aim of the final queue algorithm is that each request must be given significance to be processed. Amongst the various queue models available, the analytical study was the comparative study between M/M/n/K and G/G/n/K queue models. The section illustrates the pseudo code of both the queue.

2.3.1 Pseudo code for M/M/n/K Queue Model

```
begin
//Computation of Arrival Sequence
using M/M/n/K Queue Model
read b, SUM=0
read μ, Q(L)
for i=1 to N in step of 1 do
    x(i) = RAND U((i))
    y(i) = -1/b(log(1-x(i)))
    Write y(i)
end for
for i=1 to N in step of 1 do
    SUM=SUM+y(i)
    Write SUM <Arrival Sequence>
end for
//Computation of Departure Sequence
using M/M/n/K Queue Model
for i=1 to N in step of 1 do
    x(i) = RAND U((i))
    y(i) = -1/b(log(1-x(i)))
    Write y(i)
end for
for i=1 to N in step of 1 do
    SUM=SUM+y(i)
    Write SUM <Departure Sequence>
end for
//Computation of Response time for M/M/n/K Queue Model
for i = 1 to 10 in step of 1 do
    Read a(i)
    Write a(i)
    for i = 1 to 10 in step of 1 do
        Read d(i)
        Write d(i)
    end for
    Q=0
    for i=1 to 10 in step of 1 do
        Write(y(i))
    end for
end for
```

2.3.2 Pseudo code for G/G/n/K Queue Model

```
begin
//Generate Gaussian Series
read n, λ, μ, b, a
read SUM=0
read str
for i=1 to n in step of one do
    sum=0
    for j=1 to n in step of one do
        P(j)=RAND U(j)
        sum=sum+P(j)
    end for
    y(i) = 1/12*sum
    Write(y(i))
    end for
end for
Y[GAUSSIAN]
//Generate Equiprobable Series
for i=1 to 10000 in step of one do
    Generate RAND U(i)
    Write((U(i))
end for
Y[EQUIPROBABLE]
//Generate Negative-Exponential Series
for i=1 to 10000 in step of one do
    x(i) = RAND U((i))
    y(i) = -1/b log(1-x(i))
    Write(y(i))
end for
Y[NEG-EXPONENTIAL]
//Generate Modified-Geometric Series
b=1/λ
a=1-b
delta=b/a
for i=1 to n in step of one do
    x(i)=RAND U((i))
    y(i)=(1-exp(delta*x(i)))/b
    Write(y(i))
end for
```

Copyright © 2013 ISICO
count \( q(i) \)
\[ Q=Q+q(i) \]
Avg \( Q(L) = \frac{Q}{N} \)
Write \( Q(L) \)
end for
\[ R(t) = \frac{1}{\mu Q(L)} \]
Write \( R(t) \)
end

with the condition
// Arr(i) < Dpr(i); \ i \in (1,N)
str= Y[GAUSSIAN] U Y[EQUIPROBABLE]
U Y[NEG-EXPONENTIAL] U Y[MOD-GEOMETRIC] U Y[BERNAULLI] \( \{D(i,\ i\in0,n)\} \)
for i=1 to n in step of one do
Dp(i)=0
Dp(i)=Dp(i) + D(i)
end for
Theta=SUM of Dp(N)
for k=1 to N in step of one do
Dpp(k)=[Dp(k) * N *\( \mu \)]/Theta -1.0
Write Dpp(k)
end for
// Computation of Response time for G/G/n/K Queue Model
Read Arrival Instances Arr(i) having General Distribution
Read Departure Instances Dpp(i) having General Distribution
for i=1 to n in step of one do
q(i)=[No of arrival Arr(j)<D(j)] - *(i-1)
SUM=SUM+q(i)
end for
for i=1 to n in step of one do
Rt(i)=SUM/n
Write Rt(i)
end for
end

3. TECHNICAL SPECIFICATIONS OF THE ELECTRONIC COMMERCE SETUP

Web Request Management Server and three data centers forms the base of the complete experimental setup. For the experiment, Apache Tomcat 7 is used for Web Request Management Server. The challenging job for the study was to setup three Data Centers for the information flow. The specification for each of the Data Centers is:

<table>
<thead>
<tr>
<th>Data Center 1</th>
<th>Data Center 2</th>
<th>Data Center 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: 0</td>
<td>ID: 1</td>
<td>ID: 2</td>
</tr>
<tr>
<td>Number of Processors: 17</td>
<td>Number of Processors: 12</td>
<td>Number of Processors: 10</td>
</tr>
<tr>
<td>VM Policy: TIME_SHARED</td>
<td>VM Policy: TIME_SHARED</td>
<td>VM Policy: TIME_SHARED</td>
</tr>
</tbody>
</table>

The public cloud (Data Center 1) was used to set up Application and Data Base Server and was based in US. The other two private clouds (Data Center 2 and Data Center 3) are based in Sharjah within the University premises. There are 5 VM used in two Data Centers 2 and 3, which handles 300 requests per minute and buffer size is 1000 Bytes for each request.
3.1 Load Balancing Policy for the Data Center 2 and Data Center 3

Data Center 2 followed Round Robin technique, wherein, the controller allocates the requests to a list of VMs on a turning basis. The first request is assigned to a VM- selected randomly from the group and then the controller disperses the successive requests in a circular order. Once the VM is consigned the request, the VM is moved to the end of the list. In this technique; there is a better allocation concept known as Weighted Round Robin Allocation in which controller assigns a weight to each VM so that if one VM is capable of handling twice as much load as the other, the powerful server gets a weight of 2. In such cases, the controller assigns two requests to the powerful VM for each request assigned to a weaker one. The major issue in this allocation is that it does not consider the advanced load balancing requirements such as processing times for each individual requests. Thus the choice is to use RRLB technique for Data Center 2.

Data center 3 uses Throttled Load Balancer (TLB) allocation policy. The TLB preserves a record of the state of each virtual machine (busy/ideal). If a request arrived regarding the allocation of virtual machine, the TLB sends the ID of ideal virtual machine to the data center controller and data center controller allocates the ideal virtual machine.

4. RESULTS AND ANALYSIS

Table 1 demonstrates the experiment outcomes for each of the Data centers w.r.t the number of requests arriving per minute. The table is illustrative in nature and is self-explanatory.

<table>
<thead>
<tr>
<th>Number of Requests per minute</th>
<th>M/M/n/K Queuing Model</th>
<th>G/G/n/K Queuing Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Processing time at DC1</td>
<td>Processing time at DC2</td>
</tr>
<tr>
<td>250</td>
<td>299.143</td>
<td>299.614</td>
</tr>
<tr>
<td>300</td>
<td>301.316</td>
<td>298.424</td>
</tr>
<tr>
<td>350</td>
<td>299.391</td>
<td>300.918</td>
</tr>
<tr>
<td>400</td>
<td>299.366</td>
<td>300.509</td>
</tr>
<tr>
<td>450</td>
<td>301.408</td>
<td>299.85</td>
</tr>
<tr>
<td>500</td>
<td>299.962</td>
<td>301.545</td>
</tr>
<tr>
<td>550</td>
<td>303.333</td>
<td>299.937</td>
</tr>
<tr>
<td>600</td>
<td>307.562</td>
<td>300.281</td>
</tr>
</tbody>
</table>

The analysis of the results is displayed in the following three charts of Figure 3.
5. CONCLUSION

Based on the outcomes cited in section 4 of the research paper, for DC1 and DC2, the response time is practically the same for the lower number of requests received at the web request management server. However, G/G/n/K model was found to be more appropriate for greater quantity of requests arriving at the architecture. For DC3, both the models were found correspondingly apposite for the number of requests arriving. As a commendation of WQM algorithm, the employment of WQM algorithm using G/G/n/K was found to be more seemly under the given conditions.

REFERENCES


BIBLIOGRAPHY OF AUTHOR

Dr Riktesh Srivastava is presently working as Assistant Professor, Information Systems in Skyline University College, Sharjah, UAE. Dr Srivastava has published more than 50 papers in various International Journals of Computer Science and Information System and is on editorial board of more than 10 International Journals of repute. His research areas include Distributed Computing, Cloud Computing and Mobile programming.