International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 2 Issue: 2 225 – 230

Threshold and Time Based Overload Detection Approach For Proxy Based Self-Tuned Overload Control for Web Servers

Charulata J. Patil

Asst. Professor, College of Engineering,Pune
cjp.comp@coep.ac.in

***Abstract*—** In today's world, almost all of the E-commerce websites or applications are available over the Internet. Such websites are often faced with incoming load of requests that exceeds their capacity, i.e. they are subjected to overload. Most existing web servers show severe drop in throughput and thus degraded performance at high overload.
The reference paper [1] suggests a Proxy based overload control mechanism, which uses the drop in throughput relative to arrival rate as an indicator of overload. On overload detection, a self-clocked admission control is activated, which admits a new request only when a successful reply is observed to be leaving the server system. Thus, the mechanism is self-tuned, and requires no knowledge of the system hardware. Such overload detection approach is time based and hence has a few limitations. This paper suggests a new overload detection approach that is Threshold and time based overload detection approach and comments about its advantages over the Time based overload detection approach.
The paper also involves analysing and comparing the results of Tomcat server without overload control, Proxy Based Self Tuned Overload Control (PBSTOC) with Time based overload detection [1] and PBSTOC with Threshold and Time based overload detection.

Keywords—*Web application; performance enhancement; Overload detection; Overload control.*

*****

I. INTRODUCTION

The Proxy-based Self-tuned Overload Control is an approach where the bottleneck resource is not known in advance and that can be applied for multi-tier applications. The paper [1] claims that an absolute indicator of a system in overload is when its throughput (rate of successful completion of requests) is lower than its request arrival rate. If the completion rate drops below arrival rate, it is a clear indicator, which the server cannot process the requests at the rate they are arriving, and is hence, overloaded. This is the ***Time based overload detection approach***. On overload detection, a self-clocked admission control is activated, which admits a new request only when a successful reply is observed to be leaving the server system. Thus, the mechanism is self-tuned, and requires no knowledge of the system hardware. The newly suggested ***Threshold and time based overload detection approach*** assigns weight to each incoming request and when the total inLoad of all the current incoming requests reaches the threshold value, indicates overload. Similarly, on successful completion of the request, reduces the request weight from the inLoad. The approach is combined with the time based overload detection explained above and together these approaches detect overload.

II. BASIC DESIGN

A. Main modules

Based on the previously stated requirements the basic overload handling system should have 2 main modules -

- Overload Detector: To detect the overload situation
  - Load calculation : To calculate the total current load on the system
  - Throughput calculation : To calculate the total current throughput on the system
  - Overload detection : When the throughput of the system drops drastically as compared to the load of the system.
- Overload Controller: To control admission of every new request into the system.

Since the mechanism works without the knowledge of which server is the bottleneck server, the requests made to the system as well as the replies should be measured by an external entity. Thus, a proxy which sits at the front end of the server system can be used to implement the modules and mechanisms mentioned above. For this implementation, a Java based proxy called Muffin is used as an external entity which could act as an observer for the incoming requests and outgoing replies. All the four modules discussed above should be implemented in this proxy. This proxy is set up in Reverse proxy mode.

.

B. Implementation architecture

1) Web application: This is the actual web application implemented and running on the web server. It is a 2 tier web application that uses the database server.
2) Web proxy with Self-tuned overload control: This is the proxy applied for the web server. The self-tuned overload control code will be embedded in the web proxy. This proxy is set in reverse proxy mode.
3) Web clients: These are the end users of the web applications that fire the requests to the web server. These are responsible for generating the traffic to the web server.

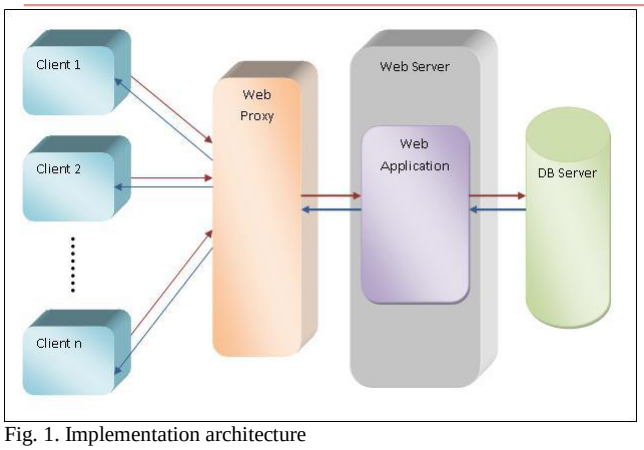


Fig. 1. Implementation architecture

III. DETAILED DESIGN

The following sections focus on the approach of Overload Detection i.e. Interval based load detection approach for PBSTOC suggested in the reference paper, its limitations and the improved approach i.e. overload threshold and interval based load detection approach for PBSTOC and its benefits over the earlier.

A. Interval based load detection approach for PBSTOC (Proposed in reference [1])

1) Modules

Load Calculation Module (LCM)
Steps:

- Each arriving request is routed through this module.
- It calculates the load at the end of each monitoring interval and sends an exponential average to the Overload Detection Module

Load Calculation Module (LCM)
Steps:

- Each successfully exiting request is observed by this module, which calculates the throughput at the end of the monitoring interval.
- It then sends an exponential average of the throughput to the Overload Detection Module.
- After an overload notification is received from the overload detection module, for every successful reply received from the server, it sends a notification to the overload control module, throughout the overloaded interval

Overload Detection Module (ODM)
Steps:

- At the end of its monitoring interval, this module calculates the average load and throughput, using the values received from the LCM and TCM at the end of their respective monitoring intervals.
- A Threshold value is used in order to flag overload in the system. The ratio of throughput to load as computed by this module is compared with this threshold. If the ratio is below the set value of Threshold, the system flags overload. (Note that Threshold is the only parameter in the overload control mechanism that needs to be manually configured).
- On detecting overload, a notification is sent by this module to the TCM as well as the overload control module. If the ratio is above the threshold, the overload status is reset to 'OFF'.

Overload Control Module (OCM)
Steps:

- On overload, all the requests which otherwise go through the LC module directly, are re-directed through this module.
- On arrival of a new request, the request is simply queued into its buffer, waiting for a notification from the TC module.
- Upon receiving a successful reply notification, the module checks for the waiting but not timed out request at the head of its buffer queue; it dequeues such a request and forwards it to the LC module.
- If overload flag has not been set, this module does nothing and every new request passes directly to the LC module.

2) Limitations of the approach

- Overload scenario is detected only after the monitoring time interval is passed.
- Every request is assumed to have the same load on the server i.e. the processing time; resources used etc. for all types of requests are assumed to be same.
- The load on the server is detected only on the basis of number of requests sent to or from the server and not on the basis of the actual load on the server.

B. Overload threshold and interval based load detection approach for PBSTOC (New approach discussed in this paper)

1) Modules

Translation filter
Steps:

- Pass all the incoming request through translation filter if it satisfies the Pass rule. If not, display 'Bad request'.
- If yes, Pass the request through Traffic monitor
- Check the URL of the incoming response and pass it through the filter.

Traffic monitor (Thread)
Steps:

- For each incoming request, check whether system is overloaded by Overload checker
- If not overloaded, pass the request directly to Load Calculator
- If overloaded, pass the request to Overload controller

Load calculator (Thread)
Steps:

- Find the type of the incoming request.
- Calculate the inLoad for the incoming request by inLoad = reqWt
- Update Overload detector inLoad.
- inReqNo++
- Update the currServerLoad as :inReqNo*reqWt
- Go to Overload detector step 3
- Call Call Handler

Call Handler
Steps:

- Check if the system uses any other proxy server.
- If yes, let the request pass to the actual server through that proxy.
- If not, direct call to the server and get the response back from the server.
- Call Throughput Calculator.

Throughput calculator (Thread)
Steps:

- Find the type of the incoming reply.
- InReqNo- -
- Calculate outLoad for current reply
- Update outLoad in Overload detector.
- Call Overload detector step 4, 5
- Update currServerLoad as currServerLoad = currSetrverLoad - outLoad.

Overload detector (Thread)
Variables:

- isOverload
- inLoad
- outLoad
- currServerLoad : This is the current load being handled by the server.
- overloadThreshold : Should be optimum. Any value above this will overload the server
- currLoadThroughputRatio : Current inLoad:outLoad ratio

Steps:

- Running as a separate thread. Sleeping for unit amount of time.(configurable)
- Calculates inLoad Vs outLoad as currLoadThroughputRatio for this amount of time. Find whether this is greater than the loadThroughputRatioThreshold value (configurable). If yes, set isOverload.
- If !isOverload and currServerLoad > overloadThreshold
  If yes, set isOverload (This is the key part. It helps adjusting the overloadThreshold to the optimum value.
- If this check is done in 'overload checker', the new request will never be able to enter into the server and overloadThreshold will never change)
- if isOverload and currServerLoad > overloadThreshold and currLoadThroughputRatio < loadThroughputRatioThreshold, update the overloadThreshold.
- If the currLoadThroughputRatio > the loadThroughputRatioThreshold (configurable) and !isOverload, set isOverload
- isOverload and currLoad < overloadThreshold, reset isOverload.
- isOverload and currLoadThroughputRatio < loadThroughputRatioThreshold (configurable) reset isOverload.

Overload Controller (Thread)
Steps:

- Queue all the incoming requests.
- Whenever reply comes,
  - Go to Throughput calculator
  - If queue is empty and go to Overload detector step 6
  - Allow the n queued requests that have their wt <= outLoad
  - Take these requests to Load Calculator

2) A typical request flow

- All the incoming requests are passed through the Translation filter of the muffin proxy.
- If the request URL satisfies filter condition, it passes through Traffic monitor
- A Traffic monitor is responsible for all the request and reply flow to and from the server.
- Inside Traffic monitor
  - For every request, isOverload condition is checked
  - If overload, all the requests are passed to the Overload controller where they are queued
  - If not, the requests are directly sent to the server via Load calculator
  - All the replies coming from the server are passed to Throughput calculator.
  - Overload detector continuously keeps check on all the incoming requests and outgoing replies. Depending on the Load:Throughput ratio or overloadThreshold, it detects the overload scenario.

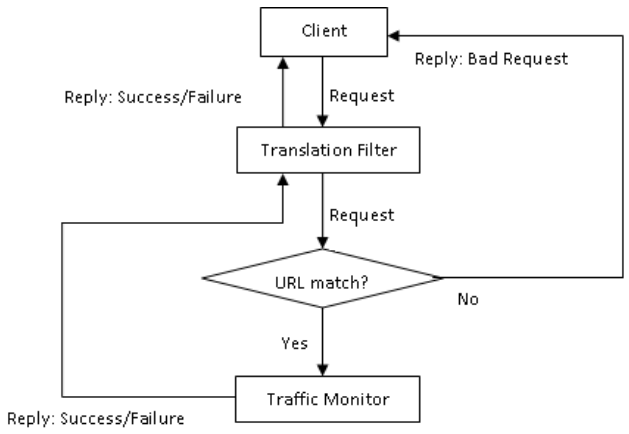


Fig. 2. Request Flow outside Traffic Monitor

 ISSN: 2321-8169
225 – 230

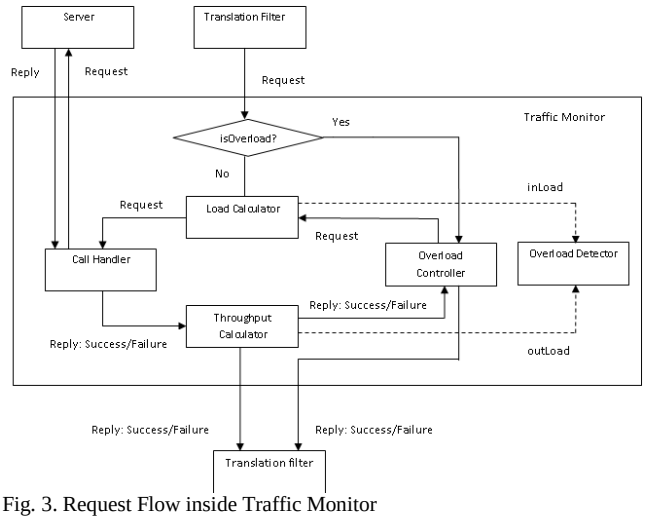


Fig. 3. Request Flow inside Traffic Monitor

3) Advantages of the approach

- Automatic Set/Reset of Overload status depending on Load:Throughput ratio
- Overload Threshold
- Setting the optimum Overload Threshold dynamically.
- Separate weight assigned to each type of request.
- The inLoad as well as outLoad of the request depends on the assigned weight for that type of request.
- The inLoad of all the requests sent to the server are used to find the total load on the server.
- Detection of overload scenario as soon as the new request is arrived based on the Overload Threshold and request weight.

IV. EXPERIMENT CONFIGURATION

Common configuration used for all experiments

- Tomcat without overload detection
  - Socket timeout : 90ms
  - Max Threads : 250
- PBSTOC with Time based overload detection
  - LoadThroughputRatioThreshold : 1.2
  - Socket timeout : 90 ms

  Static requests
  - LoadThroughputRatio calculator monitor interval : 1000ms

  Dynamic requests
  - LoadThroughputRatio calculator monitor interval : 500ms
- PBSTOC with Threshold and time based overload detection
  - LoadThroughputRatioThreshold : 1.2
  - Socket timeout : 90 ms

Static requests
- LoadThroughputRatio calculator monitor interval : 1000ms
- Overload threshold : 500

Dynamic requests
- LoadThroughputRatio calculator monitor interval : 500ms
- Overload threshold : 200

V. EXPERIMENTS AND RESULTS

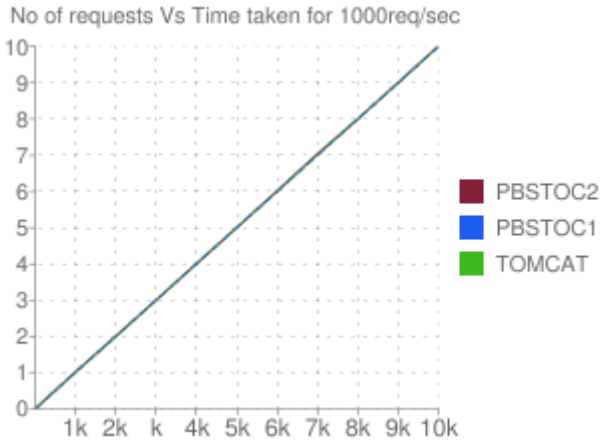
For each experiment, the results are compared for web application implemented

- Without PBSTOC i.e. requests directly going to the web server e.g. Tomcat
- With PBSTOC and time based overload detection approach suggested in reference [1] i.e. PBSTOC1.
- With PBSTOC and Newly proposed threshold and time based overload detection approach i.e. PBSTOC2.

A. Multiple static requests of single type with constant request rate and variable number of requests

Configuration
- Request arrival rate : 1000 requests/sec
- Request type : static
- url wt : 1

Results :



Observation
- Linear increase in time as the number of requests increased for Tomcat, PBSTOC with Time based overload detection as well as PBSTOC with Threshold and Time based overload detection
- Results almost identical for Tomcat, PBSTOC with Time based overload detection as well as PBSTOC with Threshold and Time based overload detection

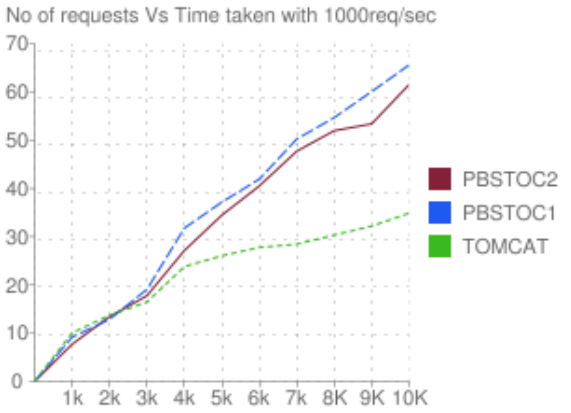
B. Multiple dynamic requests of same type with constant request rate and variable number of requests

Configuration
- Request arrival rate : 1000 requests/sec
- Request type : dynamic
- url wt : 1

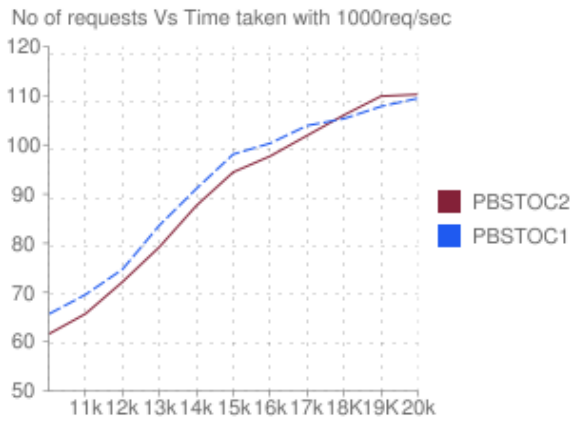
Results

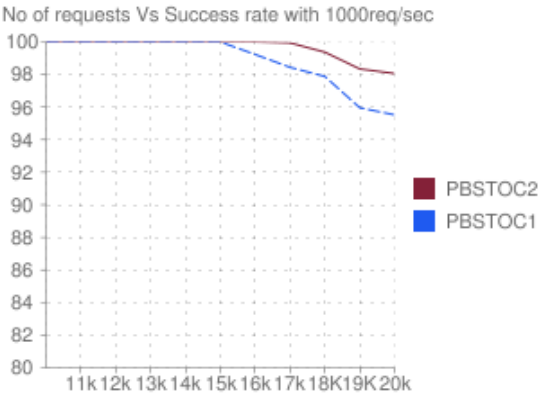
 ISSN: 2321-8169
225 – 230

1000-10000 requests :



11000-20000 requests :





Observation

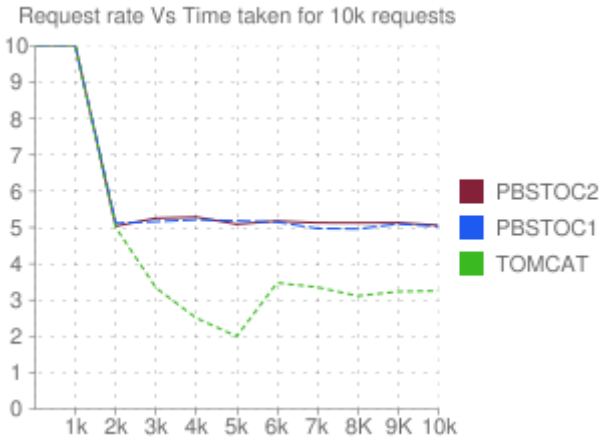
- Linear increase in time as the number of requests increased for Tomcat, PBSTOC with Time based overload detection as well as PBSTOC with Threshold and Time based overload detection
- Tomcat shows severe drop in percentage of successfully completed requests after 4000 requests.
- Performance of PBSTOC with Threshold and Time based overload detection is slightly better than PBSTOC with Time based overload detection.
- Performance of PBSTOC with Threshold and Time based overload detection > with Time based overload detection > Tomcat
- For BSTOC with Time based overload detection and PBSTOC with Threshold and Time based overload detection, the percentage of successful requests drops below 100% when the response time reaches the socket timeout(configurable value)
- The drop in the percentage of successful requests is very sharp in case of Tomcat server without PBSTOC and very less for PBSTOC with Time based overload detection and even lesser for PBSTOC with Threshold and Time based overload detection

C. Multiple static requests of same type with variable request rate and constant number of requests

Configuration

- Number of requests : 10000
- Request type : static
- url wt : 1

Results :



Observation

- For Tomcat server without PBSTOC, though the request rate rises above 5000 req/sec, constant connection rate of 3000conn/sec is maintained by Tomcat web server.
- For both the PBSTOC systems, though the request rate rises above 2000 req/sec, constant connection rate 1950conn/sec is maintained.

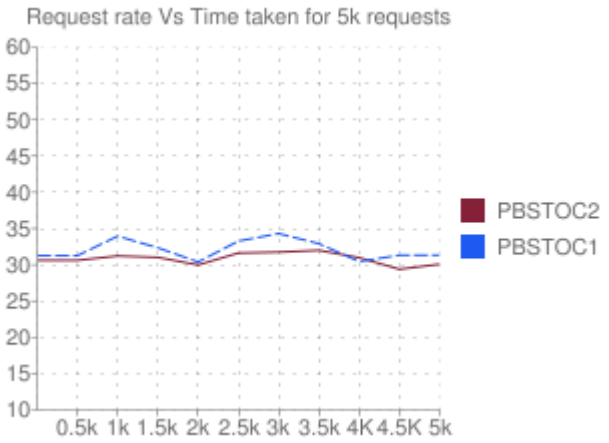
 ISSN: 2321-8169
225 – 230

D. Multiple dynamic requests of single type with variable request rate and constant number of requests

Configuration

- Number of requests : 5000
- Request type : dynamic
- url wt : 1

Results :



Observation

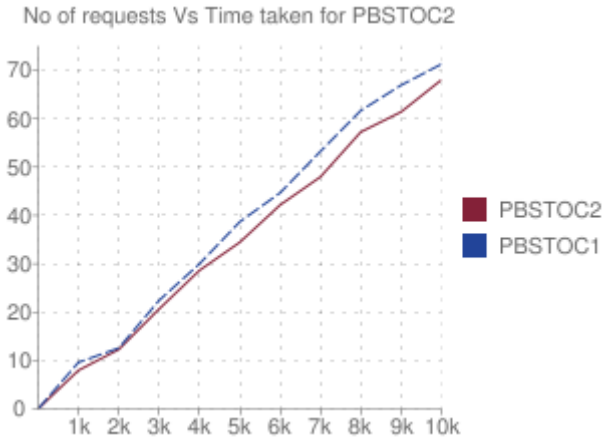
- For Tomcat server without overload control, the success rate starts dropping from 500 req/sec.
- For both the PBSTOC systems, though the request rate rises above 500 req/sec, constant connection rate around 160conn/sec is maintained.

E. Multiple dynamic requests of different type with constant request rates

Configuration

- Request arrival rate : 1000 requests/sec
- Request type : dynamic
- url fired : /friendsnet/home?lid=3, url wt : 1
- url fired : /friendsnet/view/friends?lid=3, url wt : 1.15

Results :



Observation

- Performance of PBSTOC with Threshold and Time based Overload detection > PBSTOC with Time based Overload detection
- The difference in time taken becomes more and more when the weight difference in the URLs that are fired is more.

VI. Conclusion

- For static content, Tomcat server without overload detection gives better performance compared to PBSTOC.
- For dynamic content, Performance of PBSTOC with Threshold and Time based overload detection > PBSTOC with Time based overload detection > Tomcat without overload detection.

VII. References

[1] Rukma P. Verlekar , Varsha Apte, A Proxy-Based Self-tuned Overload Control for Multi-tiered Server Systems. HiPC 2007, LNCS 4873, 285-296, 2007.