Overload Pattern classification For Server Overload Detection

San Hlaing Myint*
*University of Computer Studies, Yangon, Myanmar.

ABSTRACT
In automated overload control system, it is difficult to accurately predict the condition of high performance server. Overload Control mechanism mostly depend on the accuracy of detection and prediction. Therefore, it is important to accurately determine whether a server is or is not in overload state. So we can prevent from overload saturation before it occurs. An important step towards overload prediction is the ability to detect it. In this paper, we described overload pattern classification by using Random Forest classifier and overload detection mechanism was proposed for server overload control system. In proposed mechanism, we make a synthetic dataset by using performance counter pattern and do experimentation on various classifiers.

Copyright © 2013 Institute of Advanced Engineering and Science.
All rights reserved.

1. INTRODUCTION
The One of the key factors in customer satisfaction is the application performance. If an application regularly takes too long to respond, the customer may become unsatisfied and he may eventually switch to another service provider. Our aim is to detect performance problems for Ultra-Large-Scale (ULS) system and to provide an overload prevention mechanism. Some existing approaches made their detection based on small amount of performance metric such as response time [Ref-10]. Proposed method is based on measuring a wide variety of performance counters such as ...MemoryAvailable Mbytes and ...Processor%processor Time.

Some conventional control approaches based their request discard decisions on hard thresholds and make admission decision [Ref-11]. Traditionally, server utilization or queue lengths have been the variables mostly used in admission control schemes. In this work, the response of performance counters was chosen as the control parameter since it indirectly affects system utilization and system overloading. Admission controller negatively affects the performance of some customers, therefore, our approach tried to avoid this problem. This approach reduces request drop rates and raises up server performance.

When server gets overloaded, the response time of hardware components become long and the response time of a service becomes long which affects the many users. When hardware components response so long, server becomes overload vice versa. One of the best solutions is to reduce request drop rate of admission control mechanism by predict the state of the server. So the admission control mechanism handles some requests with fair admission decision whenever the arriving traffic is too high and thereby maintains an acceptable load in the system.

In server overload control, an interesting problem is that the underling hardware should be scale up when request rate exceed the limitation of server. In real time situation, it is very difficult to scale up hardware resources before user notices a decrease in performance. Therefore, server overload situation is needed to predict accurately with complete set of performance counter. The acceptance decision of admission
control process mostly depends on accuracy of overload predictor. So it is very important to take a complete relevant performance metric for overload prediction.

In this paper, server overload detection mechanism is presented as a part of overload control system that we proposed. We present how random forest can improve server overload control. Our research is an ongoing research. We make a synthetic dataset by using performance counter patterns. Experimentation is performed based on many classifiers for evaluating the performance of the best classifier. Experimental results demonstrate that selected classifier improve the accuracy of perdition decision.

2. Related Work

The aim of existing research on overload prediction mostly related to admission control and resource scheduling issue for server overload prevention. An admission control mechanism for web servers using neural network (NN) was proposed in [2]. The control decision is based on the desired web server performance criteria: average response time, blocking probability and throughput of web server. A NN model was developed able to predict web server performance metrics based on the parameters of the Apache server, the core of the Linux system and arrival traffic. In [4], Server overload detection method is proposed by using statistical pattern recognition method. The classifier predicted server overload situation (underload, normal, and overload) on 14 performance counters that they assume to be significant for overload detection.

Ref [3] presented a dynamic session management based on reinforcement learning. A learning agent decides the acceptance or rejection of an arriving session by estimating the response time only for service request. In Ref [5], an efficient admission control algorithm, ACES, based on the server workload characteristics. The admission control algorithm ensures the bounded response time from a web server by periodical allocation of system resources according to the resource requirements of incoming tasks. By rejecting requests exceeding server capacity, the response performance of the server is well maintained even under high system utilization. The resource requirements of tasks are estimated based on their types. A double-queue structure is implemented to reduce the effects caused by estimation inaccuracy, and to exploit the spare capacity of the server, thus increasing the system throughput.

In our experience, most of existing research emphasized response time and other related factors of web service to predict system resource and server overload condition. In our consumption, service response time is directly related to hardware components of physical server. Therefore, it is impossible to lack estimation response time of hardware components. It is indispensible to know the response time value of hardware components to increase estimation accuracy of prediction method. In this work, performance counters are chosen as important variables of detection mechanism and the best classifier is defined on our own experimentation.

3. Server Overload Detection

In this section, we will explain how to detect the overload condition of physical server by using classification method. In this approach, the following stages are distinguish (i) Data generation, (ii) Data preparation, (iii) Designing classifier, and (iv) Evaluating classifier. The implementation of these stages will present in next section.

3.1. Data generation

The first step of proposed method is collection data from the server by using performance monitor (PM). PM produces performance counters which describe server states (State1, State2, State3) which is defined in proposed Server Overload Control approach. Here, State1, 2, 3 means; Low, Normal and High). Data set is generated on window server which allow to use Performance Monitor tool. In order to train classifier, synthetic data set is created. We avoided collecting data from real server because it can take long time to get enough data. Therefore, synthetic data set is created by using load generator such as CPU Busy which perform a stress test on the same server. But the specifications of production server must equal to real server. During stress test, the load will vary from one state to another. Two measurements are interested for training data set; (i) Performance counter pattern which is used to describe the server state, and (ii) Performance counters value which is used to decide whether a performance counter pattern should be defined as state1 or state2 or state3.

3.2. Data preparation

3.2.1. Feature selection

Performance counters are measurements of system state or activity. They can be included in the operating system or can be part of individual applications. Windows Performance Monitor requests the current value of performance counters at specified time intervals. Actually Performance Monitor tool can generate 1948 performance counter patterns, but some of these are not very unlikely to be of interest when
monitoring for overload. In [4] 36 counter patterns are selected which are assume to be significant for overload prediction. In our consumption, all performance counters related to physical servers and their processes, some may be not significant because of server behavior.

![Figure 1. Result of Feature selection](image)

We can defined which counters patterns are whether significant or not by examining their counter values. Firstly we calculate information gain of each feature by using information gain ranking filter. Here, we can divide features into two groups. First group is $\geq 1$ and second group is $\geq 0$. And then group 1 data set and the whole dataset which contain group 1 and group 2 are trained and tested. The result is shown in Figure 1. According to the figure; we can see group 1 is obviously effective on classifier. Therefore, we selected significant features which contain higher information gain value ($\geq 1$). Table I presented some of selected performance counter list. We can improve data set by reducing features from 1948 to 926 features.

3.2.2. Data transformation

To be able to predict server state correctly, it is important to transform it for use with a classifier. Performance Monitor tool generate data collector set which contain performance counter patterns and values. These are need to assigned to a target class (overload, normal and underload) based on their values. We built our data set in the form of (ID, Values). Each performance counter contain about 2000 records which are recorded during 1 minute. The values are used to define which pattern are meet with S1, S2 or S3.

<table>
<thead>
<tr>
<th>Main Categories</th>
<th>Counter Name</th>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Disk</td>
<td>Avg.Disk Queue Length</td>
<td>D1571</td>
<td>0.000714245</td>
</tr>
<tr>
<td></td>
<td>%Disk Read Time</td>
<td>D1572</td>
<td>0.396698</td>
</tr>
<tr>
<td></td>
<td>%Disk Write Time</td>
<td>D1574</td>
<td>2.16554666666666</td>
</tr>
<tr>
<td>Processor</td>
<td>%Interrupt Time</td>
<td>U1616</td>
<td>1.559920773</td>
</tr>
<tr>
<td></td>
<td>% Idle Time</td>
<td>U1619</td>
<td>98.27500867</td>
</tr>
<tr>
<td></td>
<td>%Processor Time</td>
<td>U1731</td>
<td>2.504948441</td>
</tr>
<tr>
<td>Memory</td>
<td>Page Faults/sec</td>
<td>M1746</td>
<td>1655.02885723717</td>
</tr>
<tr>
<td></td>
<td>Available Bytes</td>
<td>M1747</td>
<td>1367638016</td>
</tr>
<tr>
<td></td>
<td>Pages/sec</td>
<td>M1754</td>
<td>20.5277281622</td>
</tr>
<tr>
<td>Network Interface</td>
<td>Packets/sec</td>
<td>NT1833</td>
<td>93.93193628</td>
</tr>
<tr>
<td></td>
<td>Packets</td>
<td>NT1834</td>
<td>1.020999307</td>
</tr>
<tr>
<td></td>
<td>Received/sec</td>
<td>NT1835</td>
<td>1.020999307</td>
</tr>
<tr>
<td></td>
<td>Packets Sent/sec</td>
<td>NT1836</td>
<td>1000000000</td>
</tr>
</tbody>
</table>

Table 1. Selected Performance Counter List
3.3. Designing classifier

Although understanding the data distribution is very helpful for choosing the best classifier, it is very difficult to understand the different data distribution. For small dimensional data set, it can be easy to understand by plotting the data, but it is not simple for very large scale data set. In this work, in order to know which classifier is the most suitable one for our synthesis dataset many classifiers are tried heuristically with our dataset. Some classifiers are sensitive to very large or small data set. Since performance monitor recorded thousands of performance counters contained thousands of features.

3.4. Evaluating Classifier

We can compare random forest with others classifiers (see in Table 2) by doing ten repetitions of 10-fold cross-validation. The best classifier is selected based on eight variables presented in Table 2. We tried to choose the best classifier by testing 500 up to 5000 records and 150 to 1000 features. The result are presented in Figure (2). In our experiment, we used eight criteria to measure the performance of classifiers. These criteria are Correctly Classified Instances, Incorrectly Classified Instances, Kappa statistic, Mean absolute error, Root mean squared error, Relative absolute error, Root relative squared error and Classifier speed. In this experiment, all criteria are not different so far except C8. Classifier speed is very important for real time implementation. Random forest (RF) classifier are obviously significant in criteria 8. More number of records reduces the classification speed of RF. In other criteria, RF has a significant decrease. In Figure1 and Figure2, we present the experimental result based on number of features and records. According to figure, we can see clearly RF is slightly decreased on the number of features.

![Figure 2. Experimentation on number of records](image1)

![Figure 3. Experimentation on number of features](image2)
Inspection of Fig(2) and Fig(3) shows that Random Forest gave the lowest generalization error on C2, C4, C5, C6, C7 and C8. Although Naive Bayes and RBF Network gave the lowest error on C2, C4, C5, C6 and C7, there are quite differences on C8.

<table>
<thead>
<tr>
<th>Table 2. Experimental Result on Features 926/record-1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
</tr>
<tr>
<td>--------------------------------</td>
</tr>
<tr>
<td>Instances</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
</tr>
<tr>
<td>Kappa statistic</td>
</tr>
<tr>
<td>Mean absolute error</td>
</tr>
<tr>
<td>Root mean squared error</td>
</tr>
<tr>
<td>Relative absolute error</td>
</tr>
<tr>
<td>Root relative squared error</td>
</tr>
<tr>
<td>Classifier speed</td>
</tr>
</tbody>
</table>

4. Experimental Result on Selected Classifier

Random Forest is an attractive classifier due to their high execution speed. Random Forest is well suited for microarray data. It can give excellent performance even when most predictive variables are noise and can be used when the number of variables is much larger than the number of observations, and returns measures of variable importance [5]. A measure of the importance of the predictor variables, and a measure of the internal structure of the data (the proximity of different data points to one another) [7].

In this experimentation, we tried to measure the accuracy of Random Forest classifier by testing various number of decision trees in order to obtain stable results. The number of decision trees in each forest was varied up to 500 and the optimal number required to minimize test error was obtained. The number of random features selected at each node was chosen using Breiman’s heuristic log2 F +1 where F is the total number of features available. This has been shown to yield near optimal results [12].

Figure 4. OOB error rate
The outcome of these experiments on individual datasets is shown graphically in Figure 4. This shows the graph line of lowest test error attained at 500 decision trees. Also it shows the average test error taken over all datasets. The selected classifier is tested on various numbers of features to measure the accuracy and processing speed of classifier. Figure 5 shows the result of experiment selected classifier on features.

5. CONCLUSION

In this paper, we have practically evaluated performance of various classifiers on a synthetically generated dataset. We presented how random forest classifier is match with our synthesis dataset and how can be useful for server overload detection. In prediction method, performance counter values are effective to improve the accuracy of detection mechanism. According to experimental results, the accuracy of selected classifier are . In future word, we will combine random forest with overload prevention method such as admission control and implement in our overload control mechanism.

ACKNOWLEDGEMENTS

I would like to express my heart-felt thanks to University of Computer Studies, Yangon and Professor Dr.Ni Lar Thein, Rector of University of Computer Studies, Yangon, for overall supporting for this research. Next I would like to thanks my Co-Supervisor Dr.Aung Htein Maw and Dr. Su Thawda Win for their good guidance. I would like also thanks to my teachers for their support during Ph.D coursework.

REFERENCES


**BIOGRAPHY OF AUTHOR**

San Hlaing Myint received his Bc.Tech, Bc.Tech (Hons) and Mc.Tech degrees in university of computer studies, Mandalay, Myanmar, in 2005 and 2008, respectively. He is currently working toward the PhD degree in university of computer studies, Yangon, and Myanmar. His current research focuses on overload control for web server. He is a tutor in Computer Technology at University of Computer Studies, Lashio, Myanmar.