Narrowing Down Possible Causes of Performance Anomaly in Web Applications

Satoshi Iwata*, Kenji Kono* †
*Keio University, † JST/CREST
Performance Anomalies in Web Applications

- Are unexpected, undesirable degradations in performance
- Lead to
  - Violations of service level agreements (SLAs)
  - Loss of potential customers
- Are difficult to be removed completely in test operations
  - Because they are caused by complex interactions between various runtime factors
    - E.g., workload, resource usage, and elapsed time after system startup
      - Increase of clients drives performance parameters inappropriate
      - As a service proceeds, memory leak caused by bugs accumulate little by little
Capturing Symptoms of Performance Anomalies

- Is required in real operation
  - Before the performance anomalies harm services severely

- Enables us to increase possibility of avoiding performance degradation
  - By proactively debugging the performance anomalies
  - By adding servers temporarily

Service status

- Normal
- Indicating symptoms
- Severely harmed

Key:
- Recover performance
Systematic Detection of Anomalies

- Allows the administrators to detect symptoms without any expertise
  - E.g., Detection system notifies them of anomalous deviations of response time

- But, the ad-hoc approaches are only for experts
  - Many thresholds must be determined to detect symptoms
    - E.g., Upper limit the average response time must not exceed
    - E.g., Period during which average response time must not continue to increase
Our Proposal

- Applying *control charts* to detect symptoms of performance anomalies
  - Control charts determine whether a manufacturing or business process is in a state of statistical control or not
    - By detecting deviations from the standard quality of products
    - They are suitable for detecting performance anomalies that are deviations from the standard performance

- Our method detects symptoms of anomalies that appear in *processing time*
  - Processing time is the time from request arrival at a front-end server to its departure from the server
  - Processing time can be measured unobtrusively
    - E.g., Apache web server can be configured to log each request’s URL and processing time
Control Charts

- They monitor *characteristic values* that characterize the quality of target
  - E.g., clincher size or company profitability
  - In our technique: processing time

- They raise an alarm if the measured values statistically deviate from standard values
  - To do this, they compare
    - A statistic of current characteristic values
      - E.g., average, median, or defective fraction
      - We chose median
    - Baselines calculated from the data measured in a controlled environment
Detail of Control Charts

- Base lines
  - Center line is the center of the measured statistics
  - UCL and LCL indicate 3σ above/below the center line

- Warnings
  - A plot outside the limits
  - Systematic patterns within the limits

![Graph showing control limits and warning](image-url)

- Median of clincher size [mm]
- Lot Number

Warning
Upper Control Limit (UCL)
Center Line
Lower Control Limit (LCL)

Monitoring clincher manufacturing process
Merits of Control Charts

- Administrators do not have to decide complicated parameters
  - If so, only experts can use them
  - All the parameter we have to decide is *group size* to calculate statistics
    - In control charts, the default value is “5”
    - We used the default value

- Output is simple
  - Output is “normal” or “anomaly”
  - Administrators do not have to make a decision
Effort 1: Characteristic Values

- We chose four statistics of processing time as characteristic values, instead of processing time itself
  - Average, maximum, median, and minimum
  - We prepare four control charts and plot:
    - Median of average processing time, median of maximum processing time, and so on

Merits

- We can concentrate on more harmful anomalies
  - Natural fluctuations in processing time itself raise less important warnings
- We can detect various anomalies
  - Different anomalies appear differently in processing time
Effort 2: Observing Unit

- We observe processing time of each *request type* separately
  - By investigating which servlet processes the request and what parameters are needed to process it by using URLs

- A *request type* can be naturally defined on the basis of its task
  - For RUBiS (an auction site prototype), there are 27 request types.
    - E.g., *SearchItemsInCategory*, *RegisterUser*, and *PutBid*

- Merits
  - Helps us detect early symptoms
    - Average processing times are different among request types
  - Helps us narrow down suspicious components
    - Different request types use different components
Summary of Our Method

- Observes four statistics as characteristic values
- Separately observes processing time of each request type

Request types

- Home
- Browse
- PutBid
- ... AboutMe

![Graphs showing response time for different request types.](image)
Typical Scenario

- Our approach can be used in both test and real operation phases
  - During the test phase, we calculate three *preliminary* baselines
  - During the real phase, we use the baselines calculated in the test phase
Case Studies

- We show some case studies in which we
  1. Detected symptoms of performance anomalies
     - With our method
  2. Narrowed down the possible causes
     - With the information acquired from our method
  3. Hunted for and debugged the root cause

- Experimental setup
  - Interval to calculate four statistics: 10 minutes
  - Web application: RUBiS [http://rubis.objectweb.org/]
    - An auction site prototype modeled after eBay.com
    - Implemented using JavaEE platform
  - Server software: Apache 2.2.11, JBoss 5.0.1, MySQL 5.1.34
  - Machine spec: CPU 3.00 GHz, memory 2 GB, OS Linux 2.6.27
Case 1: SearchItemsInCategory

- To calculate the baselines of control charts, we ran RUBiS with 200 clients

- SearchItemsInCategory-Min. control chart raised warnings
  - We could detect a symptom of performance anomaly
    - Minimum processing time increased only a few milli-seconds
Narrowing Down Possible Causes

- The cause of this performance anomaly would be characteristic to SearchItemsInCategory
  - Since it is the only request type that minimum value increases as time goes on

- We investigated RUBiS source code to determine suspicious components
  - The components used to process SearchItemsInCategory but not used to process SearchItemsInRegion are suspicious
    - SearchItemsByCategory Servlet
    - SB_SearchItemsByCategory Session Bean
    - The SQL sent by Query Session Bean
Hunting for the Root Cause

- We manually determined the root cause

1. We paid special attention to the SQL sent by Query Session Bean
   - It is the only component that sends complicated SQL

2. We investigated *items* table
   - The SQL only accesses *items* table

3. We speculated that growing *items* table caused the performance anomaly
   - RUBiS never delete any records in the table

4. We added a new index to the table
   - To limit the amount of records to be linearly searched
After Adding the Index

- We ran RUBiS with 200 clients again
- We could cure the performance anomaly
  - No warnings were raised
  - We could confirm that this change does not have negative side-effects
Case 2: Increasing Clients

- After calculating baselines, we increased clients

- Many warnings were raised by *-Max. control charts
  - When increased clients from 300 to 400
  - We could detect a symptom of performance anomaly
    - Average processing time increased only 91 milli-seconds

![PutBidAuth-Max. control chart](image1)
![BrowseCategories-Max. control chart](image2)
Narrowing Down Possible Causes

- We investigated source code to categorize request types into three tiers
  - To determine the server which includes the root cause
- We counted anomalous request types in each tier
- We concluded that there would be a problem in the JBoss application server
  - No warnings appear in the web tier

Observations with 400 clients

<table>
<thead>
<tr>
<th></th>
<th>web</th>
<th>application</th>
<th>database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request types that</td>
<td>0 / 5</td>
<td>2 / 4</td>
<td>17 / 18</td>
</tr>
<tr>
<td>raised warnings</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hunting for the Root Cause

- We manually determined the root cause

1. We investigated the logs generated by JBoss
   - It told us that the number of threads reached the maximum number specified by the `maxThreads` parameter

2. We raised `maxThreads` from the default (200) to 400
After Raising $maxThreads$

- We increased clients from 300 to 400 again
- We could cure the performance anomaly
  - No warnings were raised
  - We could confirm that this change does not have negative side-effects
Other Case Studies

- Many warnings were raised when increased clients from 200 to 300
  - Average processing time increased only 141 milli-seconds
  - We could narrow down possible causes to the web server
    - Because almost all request types raised warnings
  - We solved the problem by changing `KeepAliveTimeout` from default (5) to 1

- Many warnings were raised when increased clients from 400 to 500
  - Average processing time increased only 105 milli-seconds
  - We could narrow down possible causes to the web server
    - Because almost all request types raised warnings
  - We solved the problem by turning off `KeepAlive` feature
False Positive

- A RegisterUser-Min. control chart regarded normal behavior as a symptom of a performance anomaly

- RegisterUser has two main execution paths neither of which is erroneous
  1. A new user is successfully registered to the database
  2. The name of the new user has already been taken by someone else

- But we think control charts are still useful
  - They are not insignificant warnings
Related Work

- Some techniques entrust anomaly detection to administrators
  - They collect and present useful information
    - Processing time in each component [Aguilera et al. ’03]
    - Resource usage in each component [Chanda et al. ’07]
  - Control charts may be applied to detect anomalies within them

- Some techniques systematically detect anomalies within other metrics
  - We believe that these techniques are complementary to ours
    - Number of accesses to each page [Bodik et al. ’05]
    - Resource usage at the granularity of machines [Cohen et al. ’04]

- Control charts have been used to detect intrusions into web applications [Ye et al. ’02]
  - Event intensity is chosen as a characteristic value in control charts
Conclusion

- We applied control charts to detect symptoms of performance anomalies
  - They do not need complicated parameters
  - Their output is simple

- Our method
  - Observes four statistics of processing time as characteristic values
  - Separately observes processing time of each request type

- Our method could detect several performance anomalies caused by
  - Inadequate index in *items* table
  - Inappropriate performance parameters
    - *maxThreads*, *KeepAliveTimeout*, and *KeepAlive*
Narrowing Down Possible Causes

- We can decide which types of requests are becoming anomalous

- Then, we can even narrow down possible causes of the anomaly to some components
  - By analyzing the components that are used to process the anomalous request types

- Furthermore, we are guided by the following principles
  - Control charts that have long-lived warnings are important
  - We should pay special attention to the ones that raise warnings earlier
  - A root cause does not always harm all request types related to it