Scalable Run-Time Correlation Engine for Monitoring in a Cloud Computing Environment

Miao Wang
Performance Engineering Lab
University College Dublin
Ireland
miao.wang@ucdconnect.ie

Viliam Holub
Performance Engineering Lab
University College Dublin
Ireland
viliam.hulob@ucd.ie

Trevor Parsons
Performance Engineering Lab
University College Dublin
Ireland
trevor.parsons@ucd.ie

John Murphy
Performance Engineering Lab
University College Dublin
Ireland
j.murphy@ucd.ie

Patrick OSullivan
WPLC SVT
IBM Software Group
Dublin, Ireland
patosullivan@ie.ibm.com

Abstract

Monitoring the status of running applications is a real life requirement and important research area. In particular log analysis is often required to understand how the system is behaving during execution. For example it is common for system administrators to collect and view logs from different hardware and software components to gain an understanding into system behavior, especially during activities such as problem determination. A recent research project in this area, the Run Time Correlation Engine (RTCE), provides a framework for run-time correlation of distributed log files in a scalable manner for enterprise applications. The framework has been designed for enterprise applications consisting of distributed software components and is in use in real industry environments. The purpose of this paper is to explore how the RTCE can scale for cloud computing environments where providers of cloud services will require large architectures (e.g. data centers) to deploy such services.

1. Introduction

Cloud computing is increasingly used for its benefits. It can provide an environment for modeling large scalable enterprise systems. Moreover, cloud computing can give many benefits over existing systems. The benefits include allowing systems to easily scale with more machines, processing more parallel tasks, allowing users to run an OS inside of a web browser and simplifying client application running environment [7]. Many companies such as Google, Amazon, Microsoft and Sun are running cloud systems to provide their services on the Internet. For example, Google Docs [6] as a software service gives users the ability to view Word, Excel, and PowerPoint files; Amazon Elastic Compute Cloud [1] provides web-scale computing capacity for end users; Microsoft Live Mesh [10] offers data storage to share files and programs among different devices; Sun Cloud [11] gives the ability to users to run their own applications or even own operating system in the Cloud environment. By using scalable, multifunctional, and flexible cloud computing environments, companies can easily improve their quality of service for their clients.

However, services running inside the cloud can be complex. Thus it can be difficult for administrators to monitor and understand the behavior of the different components in the cloud system. Furthermore, since cloud architecture may contain hundreds or even thousands of component instances [9], monitoring, analyzing, and understanding the state of the entire cloud system can be particularly challenging.

This paper focuses on log correlation and analysis for a cloud environment consisting of a large number of software and hardware components. The aim is to provide a scalable log correlation, analysis, and symptom matching architecture that can perform real time correlation for large volumes of log data. The work builds on current research whereby the Run-Time Correlation Engine (RTCE) [8] has been developed for distributed enterprise applications. The RTCE has been built in conjunction with IBM and is currently used in their integrated testing environments. This paper will (a) show the benefits of RTCE in terms of the

time savings for end users from tests carried out across a number of test teams, within an industry testing environment (b) assess the performance characteristics of the current RTCE technology through a series of performance tests and (c) will propose and discuss three new architectures to improve the scalability of the RTCE for larger scalable cloud environments where the volume of information and number of distributed components is expected to be much greater than the current deployments.

More precisely Section 2 introduces the current work, its benefits (in terms of time savings) and limitations of this approach (from a performance perspective). Furthermore, section 2 gives results from a pilot rollout program within IBM where time savings attributed to using this technology were recorded through interaction with a number of different test teams. Performance tests are also discussed in this section. Section 3 proposes a number of solutions to overcome these limitations based on scalable architectures. Section 4 will discuss some related work. Section 5 presents our conclusions and Section 6 presents plans for future work.

2. Current Work

Industry enterprise applications are highly complex, especially for systems consisting of a large number of software components. Each software component (e.g. Oracle, WebSphere, DB2) will in general produce numerous logs files containing information on the system level events. Within the log files, the amount of data produced by typical enterprise applications can be extremely large. Administrators very often need some logging facilities (e.g. log analysis/management tools) to help analyze the log data. An issue with such log facilities is that very often they are application or vendor specific. This can be particularly problematic in heterogeneous environments. Furthermore because the format and details contained in log data from different software components can differ between different applications, understanding this data in a coherent manner can be difficult for administrators. The different layouts of event data can be problematic and time consuming for correlation and analysis of the data. In additional, using such correlated data to understand the overall behaviour of the system is often required by administrators to analyze and identify the actual problem when issues arise. The task of aggregating events from different log files can be challenging. Correlating and analyzing log data at run time can be particularly difficult and at present is largely not performed, since the log files are updated regularly with large volumes of information. Testing teams in general need to wait until applications stop running. This can be particularly problematic for timely root cause analysis, which requires correlated event data in run time.

The current RTCE implementation provides runtime correlation and symptom matching in the context of large enterprise applications. The RTCE solution provides for (a) automatic data collection, (b) data normalization into a common format, (c) runtime correlation and analysis of the data to give a coherent view of system behaviour at run-time and (d) a symptom matching mechanism that can identify known errors in the correlated data on the fly [8]. Figure 1 gives the current high level architecture of the RTCE and an example of how the system monitors and analyzes logs for each application on the network. Next we will explain the different RTCE components and how they interact to provide an overall solution. Monitoring Agents (MA) are deployed on the different hardware components. Monitoring Agents read log data from each application and route the events to the Event Correlation Engine (ECE) via TCP/IP connection. When ECE receives new events from MA instances, it will process these events and present the data on the web server (Tomcat). By interacting with the RTCE web server, users can get different views of the information from the RTCE core such as a single view of correlated data from multiple sources, log statistics, and reports on automated real time matching of known symptoms [8]. To present the information in a convenient way, the RTCE web server uses dynamic web interaction technologies, avoiding the need to force users to refresh entire web page to view new information.

2.1. Time Savings

The Run-Time Correlation Engine is a useful tool especially for testing teams that need to monitor and analyze their application logs. In the case of correlating different types of log files, testing teams using RTCE do not have to be familiar with the format of each log file since the RTCE converts logs to a single common format. After deploying and testing RTCE in industry systems, we have seen that RTCE can improve productivity by saving testers' time normally spent on log monitoring and analysis. See Table 1 for a list of time savings which are taken from an initial tool roll out program within IBM. As part of this program

Table 1. RTCE time saving

Application Type	Time Saved (per run analysis)	Defects
Social Business	23 hours	10
Team Collaboration	4 hours	0
Social Networking & Collaboration	10 hours	21
Chat Server	7 hours	1
Enterprise Email Server	12 hours	4

Table 2. RTCE stability test performance

Case	CPU (Q9400 2.66 GHz)	Memory Usage (8GB available)	Network I/O	Event Processing Rate	Duration
1	100% - 140%	2GB	9.0 Megabytes/sec	78,962 events/sec	3 hours
2	100% - 140%	2GB	8.8 Megabytes/sec	76,709 events/sec	3 hours
3	100% - 140%	2GB	9.3 Megabytes/sec	80,165 events/sec	4 hours

Table 3. RTCE stress test performance

Case	CPU (Q9400 2.66 GHz)	Memory Usage (8GB available)	Network I/O	Event Processing Rate	Duration
1	300% - 350%	512MB	N/A (Crashed)	N/A (Crashed)	5 mins
2	80% - 120%	1GB	6.3 Megabytes/sec	54,942 events/sec	1 hour
3	80% - 120%	4GB	8.7 Megabytes/sec	75,352 events/sec	1 hour
4	80% - 120%	4GB	9.2 Megabytes/sec	79,608 events/sec	1 hour

the RTCE was rolled out to 7 different test teams each responsible for testing a different enterprise application. For confidentiality purposes the real application names are not given above. The table shows the time saved, attributed to the tool, for analysis of log data from reliability test runs of the different applications. We carried out informal interviews with team leads and managers responsible for the testing of each application. The aim of the interviews was to establish the time saved using RTCE versus performing manual log analysis during a reliability test run of an enterprise application. The time savings varied significantly due to log sizes and the level of analysis carried out on different applications (e.g. some applications required more aggressive log analysis during testing). In fact in some cases full log analysis was not always carried out after a test run due to the time required for manual analysis of the logs. For example for the Social Business application full log analysis typically takes three (8 hour) days for a test engineer to complete. Analysis using RTCE in contrast takes 1 hour. In addition to saving time, the RTCE was also an effective tool for detection of defects in the different applications. Table 1 also lists the number of bugs detected and raised as a result of using RTCE during the initial roll out program. By using the RTCE, a testing team can quickly detect defects as the system runs or through offline post-run analysis. Another advantage of using RTCE is that it gives a global and correlated view of all events. This is useful when a particular error occurs and the error requires analysis of logs from multiple communicating components. For example, a web server throws an exception to a user saying the user account does not exist in the system. By way of example, viewing the correlated events in RTCE, the administrator can find that the actual system database is out of usage at that particular time due to too many requests queued in database connection pool, which causes the database to reject any incoming requests. Correlating the DB error messages with the web server information in a single correlated log allows quick identification of the issue. Further advantages of the RTCE include aggregating, correlating, and analyzing events in run time, matching error messages using symptom database, and filtering noise in log data.

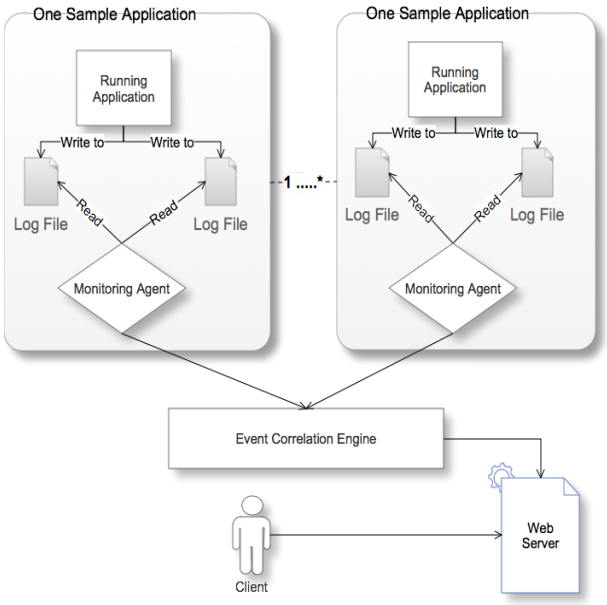

Figure 1. RTCE example.

2.2. Performance test

To assess the scalability of RTCE for future and current cloud computing environments, a series of performance tests have been carried out. First, we did some stability tests on RTCE for 3 hours duration. The result in Table 2 shows the values for CPU consumption, memory allocated, network I/O and the event processing rate for the RTCE system over the course of the test runs. The RTCE was deployed on a machine with quad-core Q9400 2.66GHz, 8GB RAM and 100Mbps Network card. During the stability test, we make use of a load generator which sends 80,000 - 82,000 events per second to the RTCE for a 3 hour period. The event processing rate is on average 78,612 events per second. We believe 2GB memory is a limiting factor and is not sufficient for RTCE to handle 80,000 events per second due to large symptom database contents stored in the memory. Based on this belief, we carried out a number of stress tests to verify if the problem is related to memory. During the stress tests, the load generator constantly sends 100,000 events per second to the RTCE. The results are shown in Table 3. In the first test, the CPU is heavily used due to the limited memory resource which will make CPU constantly swap data between memory and virtual memory (hard disk). After a few minutes, the RTCE crashed with the OutOfMemoryError exception. Comparing to the other stress tests, the only difference is that the first stress test is running RTCE with less memory. In fact it is four times lower and the CPU usage is nearly three times higher. The second stress test is also giving much lower throughput when the memory size is reduced by half. From this result, we believe the memory size has a significant effect on the RTCE performance. Furthermore with the Q9400 machine, we increased the memory size up to 4GB to perform the same stress test. In the beginning of the test, RTCE can process up to 86,000 events per second. But the event process rate reduces slowly. After one hour of the test run, RTCE can only process 77,000 events per second on average. By looking at the log file, we see by giving RTCE 4 GB memory, each time the garbage collector frees memory, it pauses the program for a longer period than with the 2GB memory case. Therefore, we believe the larger heap size than 2 GB may not necessarily guarantee better performance for RTCE as the long GC pause time slows down the event process rate (it is possible that garbage collection tuning and memory analysis may allow for more efficient garbage collection and thus a higher processing rate). Finally, by comparing the result of stress tests and stability tests, we see the 2GB memory is capable of running RTCE with a stable event processing rate.

2.3. Limitations of the current work

The event correlation engine has the responsibility to collect all the events sent by each MA. Obviously the system capabilities will heavily rely on the number of events sent to the event correlation engine by the MA instances. Under recent stability test, the event correlation engine can handle about 78,000 events per second on average shown in Table 2. In real life experience, this event processing rate is capable of handling about 192 medium size applications, which are running in IBM Dublin Software Verification and Testing team (SVT). This estimate is based on discussions with testing experts in IBM where they have estimated that 400 events per second is the maximum that RTCE needs to be able to handle per second for a single enterprise application. This is an estimate that formed part of our testing objectives. To validate this estimate we have gathered information on event volumes during recent testing activities. In a recent test, applications such as the one for Team Collaboration can generate 33.7 events per second for each log with 1050 users online. During the test, there are 26 logs generated by all such applications. Based on these measurements we are satisfied that our estimate is somewhat realistic. However different applications may produced vastly different amounts of log data. Thus log events produced per second may be application specific. The current RTCE implementation however scales well for all enterprise industrial systems where it has been applied to date. However Cloud computing environments may consist of hundreds or even thousands of applications. In this case, the current RTCE module may not be able to handle logs from all the applications. Therefore if people want to use the system for large systems like the server farms or data centers for Cloud, the scalability issue of RTCE has to be addressed.

2.4. Solution

To solve the RTCE scalability issue, we would like to have a long term solution. The aim is to design a new RTCE system that can easily scale in Cloud computing environments. The solution will only focus on the design of new architecture for RTCE instead of focusing on code tuning, which is normally considered as the short term solution. The reason of not considering code tuning will be discussed in the next section. For the long term solution, we would design a new architecture for RTCE to allow it scale in a cloud computing environment. The solution would be to make a distributed module of Event Correlation Engine (ECE) which will work with a limited number of MA instances at run time. Beyond this, there would be a master ECE to correlate all the distributed ECE instances.

3. Problem Analysis

There are several ways to solve the performance related issues. The first considered solution is to make the system run faster by either upgrading hardware or optimizing RTCE code. Upgrading hardware is costly and can only solve a short term performance requirement. Code tuning probably can provide better algorithms or data structures to make the code run faster. But there must be a new maximal throughput point. When considering the cloud environment, the size of the cloud system may steadily increase. Once the new maximal throughput point is reached, another more efficient algorithm or data structure would have to be investigated. For example, we have considered to use the well known Lempel-Ziv-Welch data compression algorithm [15] to reduce the size of the event data to be sent. By using data compression, we can increase the amount of information being sent on the network. But the algorithm still has a maximal compression ratio and will not always scale. Unless an algorithm can allow data compression with unlimited ratio, the data compression approach may not be allow the to RTCE scale well in a cloud computing environment. Therefore, the short term solutions are not ideal, and a long term solution must be investigated. The second considered solution is based on balancing the workload between each component of the system. In the case of RTCE, there are a few possible solutions that can be possibly investigated. For example parallel computing can make multiple processors process concurrent tasks simultaneously. Parallel computing requires programs written with independent tasks, so that when these tasks can be processed concurrently without conflicts. The current RTCE implementation can provide multiple threads to be processed on different processors and by doing so, RTCE can take advantage of parallel computing to maximize threads execution. But from the RTCE stability testing experiments, we see the RTCE only uses 30% of overall CPU resources. (120% out of 400%) This means the performance of RTCE code execution is not heavily relied on CPU resources. Thus by using parallel computing technologies, it may not be able to significantly increase the performance of RTCE and it may not solve the real problem of the scalability issue. Grid computing is another technology that can hep design a scalable computer system. Comparing to parallel computing, Grid is a lot more powerful. Grid technology can allow to share and access, including computers, storage systems, catalogs, networks, and various forms of sensors [5]. To support RTCE in Grid, it may be too expensive and may not be practical in industry environments as a lot of typical Grid functionalities will not be used by the RTCE. (e.g. share and access computers, catalogs, and sensors). Also maintaining instance configurations like policies, trust levels, and authorizations in Grid is not a simple task. Maintaining a Grid based RTCE may become more complicated than solving the original scalability problem for RTCE. Therefore from the cost and complexity perspectives, setting up Grid to support RTCE might waste a lot of money to build, and time to maintain. The next solution strategy will be focused on cluster technology which can link multiple RTCE instances together to improve performance. There are three possible solutions to be discussed in this paper. The first two are mainly based on Load Balancer technology and the third one is using a Master Event Correlation Engine with a number of distributed ECE instances.

3.1. Overview of approaches

In Section 2.1, the performance figure shows under stability testing the existing RTCE can process close to 10 megabytes per second which is using almost the maximal network card capacity. Inside of the RTCE, it only uses 30% of overall CPU resource and these resources are almost equally consumed by three RTCE components as shown in Table 4. By looking at the Figure 1, the bottleneck point on the Event Correlation Engine may be where the Monitoring Agents send information to the ECE via the network. Then by using multiple instances of Event Correlation Engine we may be able to solve the scalability problem. When more application instances are required for monitoring, new instances of Event Correlation Engine can be added to the RTCE to handle the extra application instances. As discussed previous, the cluster computing technology is chosen to support multiple RTCE deployment.

First, we consider to use one load balancer to setup a clustered RTCE, which should be able to provide more service capacity for event processing. Each time if the current service rate is lower than the arrival rate, we can simply add more RTCE instances to the load balancer to make the system easily scale. It will be similar to the module presented in the paper [3]. This solution is considered to be the simplest solution if the load balancer can allow RTCE to serve more MA instances when more applications need to be monitored. To evaluate this solution, some performance experiments must be carried out along with some bottleneck detection tests.

In the first solution, to allow RTCE scale there will be more instances of ECE connected to the load balancer as required. However continuing to add new instances of MA to the load balancer, may result in the load balancer becoming the bottleneck in the clustered system. Therefore a more advanced solution will be introduced which consists of using distributed load balancers. The idea is to avoid a situation where a single load balancer becomes the bottleneck. The evaluation of the second solution will be focused on how easily the structure of the system can scale when more applications need to be monitored. And how well the corre-

Table 4. RTCE CPU usage

Component	Functionality	CPU usage
File Log Processor	writing indexed log data onto the hard disk	34%
Context Factory	constructing new events for UI to consume	33%
Log Data Adapter	receiving data from MA instances	29%

lated event data can be presented to the users.

The third solution is proposed by the paper, which is the Distributed Event Correlation Engines with a Master Event Correlation Engine. Instead of using extra component such as load balancer between MA and ECE, the third solution will use the same interaction model between MA and ECE. But there will be extra functionality added in the existing ECE, which needs to be able to send high level statistics to another component via TCP/IP. The statistics data will then be consumed by the Master ECE to perform some high level analysis for the user. The solution will be evaluated based on how many applications a distributed ECE can potentially correlate after adding the extra functionality. The current RTCE can correlate approximately 200 medium size applications. In the third solution, each distributed ECE may have lower event processing rate than the existing ECE as each one has to send some high level data to the master ECE. But ideally the new performance of each distributed ECE instance will not be slightly lower than the existing ECE. Because the new functionality is designed to only send summary statistics data to the master ECE and the summary data may only be sent at low frequency. (e.g once per five seconds) Beyond the new distributed ECE, another evaluating point of this solution is how efficiently the master ECE can correlate the summary statistics from the distributed ECE instances. For example, evaluating the master ECE to work out how much CPU or Memory will increase when a new distributed ECE is connected.

3.2. Solution One

This solution is using a single load balancer to connect multiple ECE instances. Load balancer is a popular technique used in cluster systems. It allows users to interact with the entire clustered system while hiding the actual server instances behind the load balancer. Without knowing the actual server to interact with, users only need to interact with the load balancer but can still get the performance provided by multiple server instances. By using load balancer, RTCE can easily scale by simply configuring more ECE instances to the network. Each time when clustered ECE is running at full capacity while there is still more instances of MA need to be correlated, then new ECE can be easily added to the load balancer to serve the extra workloads. (See Figure 2) The clustered event correlation engines are joined together to serve all the instances of MA on the network as a single engine. For better performance, the load balancer should send events to an non-overloaded ECE. However, as shown in the picture, the interaction between web server and Event Correlation Engine is more complex, which is now one to many. One web server displays the outputs from each Event Correlation Engine. The reason of not using multiple web servers is that when load balancers send their events to non-overloaded Event Correlation Engine, the events from the same log will be randomly distributed among those engines. On the web server, the log content will be randomly presented to the user. It is going to be too hard for users to watch a particular log content by manually merging log content. Users have to manage to access all the web servers and combine the contents from each web server to get a good view of one particular log. Using a new single web server to manage each event and present these events in the right sequence can ease the log viewing for the user. However, web server has to do extra work to correlate all the processed events from each event correlation engine. This will require another research on finding the best way of organizing randomly received events. There is also other problems may occur in the new system. For example, in the load balancer if the rate of forwarding data is less than the rate of receiving data, then the load balancer itself might become a bottleneck. So far the service rate of load balancer still reminds unknown unless some real experiments can be done. Although, the load balancer technology is a common solution for optimizing web based workload [12] [18] [16], it may not be able to optimize RTCE workload due to the difference between RTCE and normal web interaction, which usually only requires little data to be sent from client to the server. For example, an HTTP server running on www.httpserver.com, a sample HTTP client request will be "GET /index.html HTTP 1.1 (newline) Host: www.httpserver.com". In this example, there is only about 100 bytes in the HTTP request body and the request probably will only be sent once in one second. Comparing to RTCE interaction, each MA can possibly send 400 events per second, the web interaction has much smaller data to be sent on the network. In the Run-Time Correlation Engine system, there will be massive volume of messages sent from MA instances to the Load Balancer. From our RTCE experiment, we can see the CPU and memory are not heavily used during the stability testing. But the network card

is nearly running at the full capacity. Then it is likely that the network card of the load balancer can also become the bottleneck of the system.

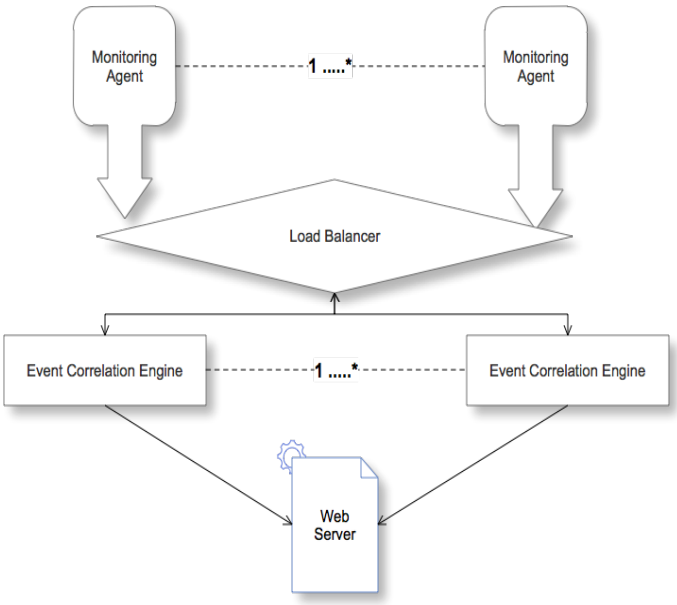

Figure 2. Load balancer based RTCE.

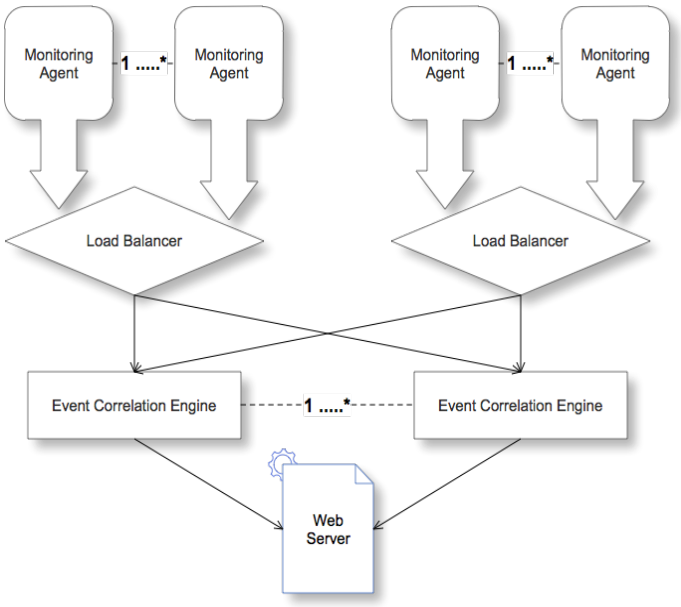

Figure 3. Multiple load balancers based RTCE.

3.3. Solution Two

Another extended approach based on the single load balancer technique is to deploy multiple load balancers, each of load balancer only handles a limited number of MA instances on the network. Then each of the load balancer can send their events to any non-overloaded Event Correlation Engine. (See Figure 3) In this module, none of load balancers and ECE instances would be the bottleneck of the system. However, the problem of randomly distributing log data to non-overload ECE still exist. The web server still has to be re-designed to be able to present random events in the correct order for users. Another disadvantage of this solution is that it will require more configurations on the load balancers, because each load balancer needs to be aware of all the available ECE on the network. Each time when a new ECE is added, all the load balancers need to be updated.

3.4. Solution Three

The third solution is based on the architecture of using a master ECE along with a flexible number of distributed ECE instances. In this module, the distributed ECE instances are distributed on the network and each ECE instance can work independently. One ECE instance only deals with a limited number of MA instances. Also there will be a master ECE to collect statistics from each distributed ECE instances and perform some high level analysis for the user. (See Figure 4) In this module, the existing ECE needs to have one extra functionality to become the distributed ECE, which needs to send the statistics data to the master ECE via TCP/IP. The distributed ECE may not be able to handle as many MA instances as the existing ECE. But as discussed previously, the performance drop may not be significant as we believe. There is no extra work required to support more MA instances comparing to using load balancers. Web server does not need to re-order any random events from ECE. It only needs to present the processed events on the web for the user. The master ECE will be slightly different to the existing ECE. The master ECE has the main functionalities of collecting statistics from each of the distributed ECE instances to provide a global view for the entire system. For example, providing figures for how many events the entire system is now processing, how many MA instances are deployed in total, and how busy each Event Correlation Engine is. For users that want to have a detailed view of each log, they can simply access the web server, which works on top of the Event Correlation Engine, which is correlating that particular log file. This solution can well solve the scalability issue for the existing Run-Time Correlation Engine with some small changes as we expect.

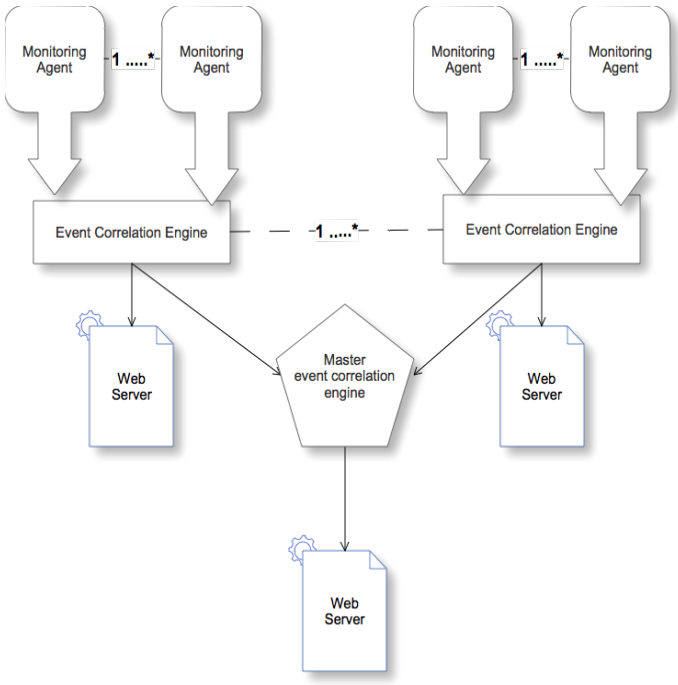

Figure 4. Distributed RTCE.

4. Related Work

Application logs contain essential information for administrators to monitor and identify issues occurring in the system. There are many software systems provide the ability to aggregate, correlate, and analyze events from log files.

Tenable Network Security Log Correlation Engine [13] is a commercial produce priced on per installation. It provides similar functionalities as RTCE provides such as aggregating, correlating, filtering, and tailing log events. Besides, the Tenable Log Correlation Engine is an ad-hoc application which is closely integrated with Tenable Security Center system [14]. Because it is closely integrated with the other product, it may or may not support other types of application logs.

SASE [17] is one of complex event processing engines over streams. The engine proposes a new event model and event language. The arriving events to the system will be transformed into new event model and processed using SSC (Sequence Scan Construction) technique. The event language is based on SQL style and extends the previous complex event languages for the purpose of monitoring application using RFID (Radio Frequency Identification) technology. In the paper, authors explained their proposed complex event language and how useful the language is. They also evaluated SASE in order to prove the effectiveness and scalability.

Cayuga [4] is another recently proposed system for complex event processing. It uses different event definition language over SASE. But the language used in Cayuga is also SQL style like. In the paper of Cayuga, it introduces a more complete system over SASE. Cayuga uses similar system architecture as RTCE. Cayuga has Event Receiver in the front-end to deserialize arriving data and assign timestamps and uses Priority Queue to organize event based on detection time. There is a Client Notifier can send event notifications to subscribing clients. The Cayuga Query Engine plays the core part of the system, it dequeues data from Priority Queue and executes state transitions generate outputs when final state is reached. Cayuga uses a shared heap between many system components to reduce overhead of creating, copying, and deleting objects. Moreover, in order to achieve better performance, it uses an optimized Garbage Collection to manage events in the heap.

The Tenable Log Correlation Engine is a commercial product and closely integrated with Tenable Security Center system. Therefore, its key algorithms and system architecture remain unknown and can not be compared to RTCE. For SASE and Cayuga, which were designed based on RFID technology, there is a couple of similarities to what RTCE provides. First of all, SASE and Cayuga are both able to aggregate and correlate data in the real time. And the new data produced by them can be useful for further consumption. SASE and Cayuga both transform arrive data into their SQL style event language, whereas, RTCE convert log data into timestamp based event object. Cayuga implemented their own shared heap and Garbage Collector. RTCE is currently using Java Virtual Machine to manage heap. There is a big difference between RTCE and SASE and Cayuga systems at the end point of the system. RTCE represents the processed data along with other functionalities such as highlighting, shadowing and tailing events on the web server for users to easily interact with. The AJAX web technology used in RTCE also makes the interaction extremely user friendly. From the scalability point of view, Cayuga uses shared heap for Event Receiver, Query Engine and Client Notifier to interact with internal event objects. This means Cayuga system must have all the components running in a single machine in order to share the same memory content. When there are more event sources needed to be managed, there must be more event receivers deployed on the same machine. Then when there are massive number of applications connecting to the Cayuga system, the hugh number of Event Receivers deployed on a single machine can easily become the bottleneck. Whereas, RTCE puts each MA on the remote machine and interact with RTCE core via network. Therefore, the MA instances will give the minimal performance impact to the RTCE system.

5. Conclusions

The single load balancer approach can enable multiple event correlation engines to support more MA instances. Although the load balancer technique works well in normal web client and server interaction, it may not be appropriate to be used in the Run-Time Correlation Engine due to the large amount of data sent from MA instances to the load balancer.

The multiple load balancer approach can solve the bottleneck issue of the first solution and the system can also take advantage of smart load distributing functionality provided by load balancers to maximize the usage of event correlation engines. However, there will be two new problems introduced. One is on the new distributed load balancer algorithm for handling heavyweight data rather than traditional web based lightweight data requests. Another one is to add extra correlating functionality to the web server to be able to correctly present log contents to the user. To solve these two problems will require more research efforts comparing to the third solution. As the result, the second solution does not seem to be an effective solution.

The third solution is presented as the distributed event correlation engine architecture. This solution may require the least efforts to implement in order to solve the scalability issue. Moreover, there is a new feature added for users to be able to have an overview of the entire cloud system. The key in this approach is to keep the existing RTCE code to require the least change. From our experience, complex solutions always bring new complicated problems to the existing system.

Comparing to the load balancer, the distributed event correlation engine has a drawback, which may not be able to maximize the usability of the distributed ECE instances. Load balancers can send data to non-overload engines to avoid machines to be overloaded. In the distributed correlation engine module, messages are permanently sent from MA instances to their linked event correlation engine only. Possibly some instances of ECE are only handling small amount of messages, while the other ones are overloaded by handling large amount of messages. This optimization issue can be solved by adding the federation feature [2] in the RTCE system. This will be discussed in the future work.

6. Future Work

The future work will be validating all the performance related assumptions in relation to each of the solutions. The first experiment ideally should be carried on the single load balancer solution to prove that single Load Balancer is not capable of handling too many log data when the number of MA instances is too big. Then the second experiment will be carried on the third solution, which is the distributed correlation engine approach. Ideally the experiment will be able to prove that the third solution is able to make the existing Run-Time Correlation Engine to be able to easily scale. And after getting the evident to prove the third approach is a solution, then we would like to build the distributed load balancers solution and compare the performance and possibly other aspects of these two solutions to exam the advantage and disadvantage of each approach. The third solution may not be perfect comparing to any other solutions. But by comparing different solutions, we should be able to get more useful ideas to improve the system. There may be new problems found during the experiments. By theoretically analyzing the problem, the solution can never be fully convinced. The real life experiments are the necessary evidence to prove the solution. The experiment is the first priority task to do.

There is another concern about the master event correlation engine. We are looking at how to use a federated way to provide the same functionality as the Master Event Correlation Engine can provide. In this future, the master engine should be removed from the architecture. Instead, each distributed event correlation engine should be able to communicate with each other to share their resources and balance the workload to avoid any machine to be overloaded. And from the usability point of view, no matter which event correlation engines web server the user is accessing, he or she can always get the global view of the system and viewing a particular log contents. The current engine should be able to transfer and process any required information from other engines for the user.

References

[1] Amazon. Amazon elastic compute cloud (amazon ec2), Oct. 2009. Available [online]: http://aws.amazon.com/ec2.

[2] J. Bakker and F. Pattenier. The layer network federation reference point-definition and implementation. In *Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA '99*, pages 125–127, 1999.

[3] S. Bouchenak, N. De Palma, D. Hagimont, and C. Taton. Autonomic management of clustered applications. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–11, Sept. 2006.

[4] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. White. Cayuga: a high-performance event processing engine. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1100–1102, New York, NY, USA, 2007. ACM.

[5] I. Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, 2002.

[6] Google. Google app engine, Oct. 2009. Available [online]: http://appengine.google.com.

[7] B. Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, 2008.

[8] V. Holub, T. Parsons, P. O'Sullivan, and J. Murphy. Runtime correlation engine for system monitoring and testing. In *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*, pages 43–44, New York, NY, USA, 2009. ACM.

[9] S. J.Carolan. Introduction to cloud computing, June 2009. Available [online]: http://www.scribd.com/doc/17274860/Introduction-to-Cloud-Computing-Architecture.

[10] Microsoft. Live mesh beta, Oct. 2009. Available [online]: http://www.mesh.com.

[11] S. Microsystems. Sun cloud computing, Oct. 2009. Available [online]: http://www.sun.com/solutions/cloudcomputing.

[12] X. Qin. Performance comparisons of load balancing algorithms for i/o-intensive workloads on clusters. *J. Netw. Comput. Appl.*, 31(1):32–46, 2008.

[13] Tenable. Tenable log correlation engine, Oct. 2009. Available [online]: http://www.tenablesecurity.com/products/lce.

[14] Tenable. Tenable security center, Oct. 2009. Available [online]: http://www.tenablesecurity.com/products/sc/.

[15] T. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, June 1984.

[16] J. L. Wolf and P. S. Yu. On balancing the load in a clustered web farm. *ACM Trans. Internet Technol.*, 1(2):231–261, 2001.

[17] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418, New York, NY, USA, 2006. ACM.

[18] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo. Workload-aware load balancing for clustered web servers. *Parallel and Distributed Systems, IEEE Transactions on*, 16(3):219–233, March 2005.