A Mechanism to Measure Quality-of-Service in a Federated Cloud Environment

Shoumen Bardhan
Hewlett-Packard Enterprise Services
4000 N Mingo Road
Tulsa, OK 74116
918.625.1833
shoumen.bardhan@hp.com

Dejan Milojicic
Hewlett-Packard Labs
1501 Page Mill Rd
Palo Alto, CA 94304
650.236.2906
dejan.milojicic@hp.com

ABSTRACT

In a federated Cloud environment, services may be composed of other services from different Clouds with different Cloud provider Quality-of-Service (QoS) guarantees. Providers running services on the multiple Clouds will be contractually obligated to meet or exceed the QoS which they have agreed to provide to their consumers. A key challenge for the service providers will be to demonstrate compliance to the agreed upon QoS. We present a basic mechanism to continuously measure QoS in a federated Cloud environment so that resources can be provisioned or de-provisioned dynamically to meet Service Level Agreements. We have validated our mechanism by constructing prototypes and the results demonstrate that it is possible to continuously measure QoS at the minute granularity and for various service configurations prevalent in the industry.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Measurement Techniques

General Terms
Measurement, Performance, Design, Standardization

Keywords
Quality-of-Service, Service Level Agreements, Cloud Federation

1. INTRODUCTION

A major obstacle in Cloud computing is performance unpredictability because providers are unable to foresee temporal variations in service demands and the geographical distribution of their consumers [1]. Furthermore, no single Cloud provider is able to establish infrastructure large enough to support the perception of unlimited computing resources of the Cloud computing paradigm. For example, Amazon EC2 customer has a limit of 20 Reserved Instances per Availability Zone that they can purchase each month [2].These limitations will necessitate Cloud providers to engage in agreements with other Cloud providers to complement their own capacity. Cloud federation facilitates dynamic expansion and contraction of application services across multiple Clouds to achieve QoS targets under variable workload and computing resources. Due to this dynamic nature of the Cloud federation, continuous monitoring on QoS attributes is necessary to enforce Service Level Agreements (SLAs). We propose a basic mechanism for representing and measuring QoS of services across Cloud federations, accounting for various configurations such as single-point-of-failures, redundant services and planned down-times.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit, September 21, 2012, San Jose, CA, USA. Copyright 2012 ACM 978-1-4503-1267-7…$10.00.

2. PROBLEM DESCRIPTION

The typical enterprise environment consists of thousands of systems per customer supporting around hundred services per client [3]. Each service, in turn, is supported by a diverse collection of systems consisting of web-servers, VMs, databases, application servers, storage, networking etc. SLAs are not only defined for each service but also for each of its sub-systems and their components. Service providers are contractually obligated to demonstrate compliance at each level of the service tree by an audit trail of calculation chain. This task of measuring and demonstrating SLA compliance becomes even more complex in a Cloud federation environment where service resources may be dynamically provisioned and de-provisioned within and across Cloud boundaries to handle sudden variations in service demands [1].

Existing techniques for measuring QoS usually employ centralized approaches to overall system monitoring and management. These centralized techniques are not an effective solution in a federation environment where live migration of virtual machines cross Cloud boundaries and challenges of auto-scaling and elasticity arise from unpredictable service demands [4].

To maximize cost-effectiveness and efficiency of composite systems it also becomes critical to allocate the optimal software and hardware configurations to ensure that QoS targets of services are achieved. This task of mapping services to resources becomes even more challenging in a Cloud federation model where expansion and resizing of provisioning capabilities are based on unforeseen spikes in workload demands in separate domains [4].

Enterprises with global operations will face difficulty in meeting QoS expectations for their entire user base because no single Cloud infrastructure provider has their data centers at all possible locations throughout the world. For example, Amazon has data centers in the US (e.g., one in the East Coast and another in the West Coast) and Europe. However, currently they expect their Cloud customers (i.e., SaaS providers) to express a preference about the location where they want their application services to be hosted [4]. As a result, Cloud providers would logically construct federated Cloud infrastructure by mixing private and public Clouds.

To meet aforementioned requirements, services need to be more SLA-aware to clearly identify the boundaries of SLA violations and responsibilities. Since the QoS attributes change constantly in a dynamic environment, measurement and monitoring of system performance is required for dynamic allocation of resources in the changing environment of Cloud federation.

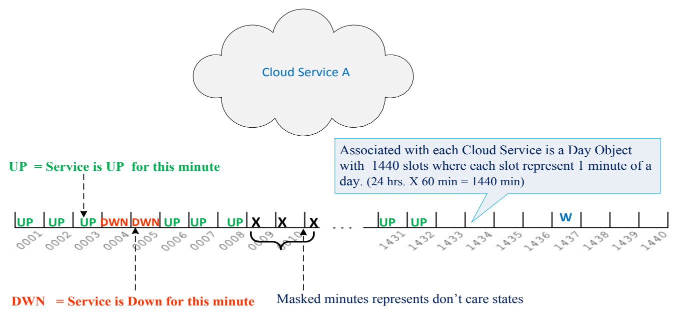
3. VISION

One of the biggest premises of Cloud computing is elastic scaling, which gives the users the perception of unlimited computing resources over the Internet. Cloud federation allows individual Cloud providers to engage in an agreement with other Cloud providers to enable elastic service composition which crosses Cloud boundaries [5]. Development of a basic mechanism to measure QoS is critical to complying with SLAs in a Cloud federation model where resource availability is uncertain. The most common QoS attributes which are part of SLA contracts are availability, response time and throughput of services. In a federated Cloud environment these metrics are constantly changing and they need to be monitored continuously to demonstrate compliance with the negotiated contracts.

We describe a basic mechanism for representing and measuring QoS of composed services across Cloud federations. We illustrate how this mechanism (Figures 1, 2, and 3) can measure 'up-to-the-minute' QoS for both individual services and composite services under various service configurations, such as single point of failure, redundant services (multiple servicer replicas) and services with planned downtime. Finally, we provide an algorithm (Figure 4) which uses this mechanism to optimize dynamic resource allocation within various service configurations.

4. REPRESENTATION

Figure 1 represents the basic mechanism for representing and measuring QoS of composed services across federations.


Cloud Service A
Associated with each Cloud Service is a Day Object with 1440 slots where each slot represent 1 minute of a day. (24 hrs. X 60 min = 1440 min)
UP = Service is UP for this minute
UP UP UP DWN DWN UP UP UP X X X . . . UP UP W
0001 0002 0003 0004 0005 0006 0007 0008 0009 0010 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440
DWN = Service is Down for this minute
Masked minutes represents don't care states

Availability % up to Nth minute = $\frac{\text{(Total number of UP minutes)}}{\text{(N - Don't Care minutes)}}$ * 100;

In this example, Availability % up to 10th minute = (6/8)*100 = 75%

If Calculated QoS >= Target QoS

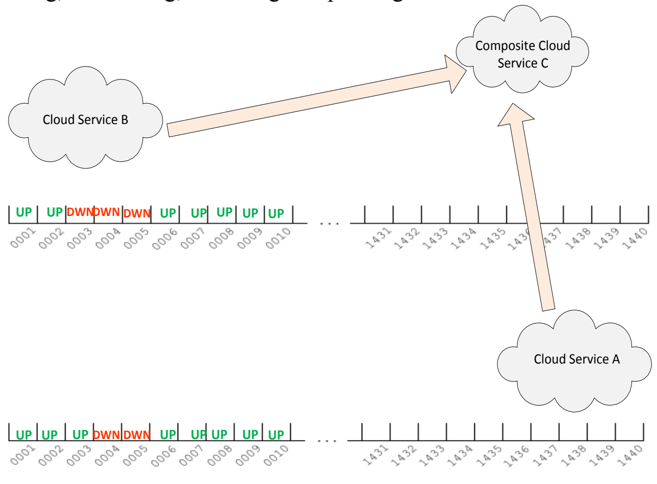
service has Met QoS up to the Nth minute

Figure 1. Basic Representation and Mechanism.

Associated with each service, is a Day object represented by an array consisting of 1440-minute objects representing each minute of a day (24 hours x 60 min = 1440). By default, each minute will be marked as 'UP' (or 1) minute. When a service disruption occurs during a particular minute the minute will be marked as a 'DWN' (or 0) minute. Additionally, a mask will be applied (indicated as X in the figure) to represent those minutes during which the user does not care whether the service is up or down. Figures 1 and 2 demonstrate the calculation of availability metrics up to the 10th minute. As long as the measured QoS is greater than or equal to the target QoS, the service would have met its expected target up to that minute. QoS of the entire Service tree and each of its components can be measured by recursively traversing the tree in a depth-first manner as explained in section 5. Also note how we assign a weight (W) vector to each minute to represent the component's relative contribution to the QoS of the composite service (parent) relative to other services for that minute. In Section 4.2 we explain how this weight vector mechanism can be used to calculate different QoS metrics under different service configurations such as composite service, single-points-of failures and services with redundancy built in.

4.1 Measuring Composite Cloud Services QoS

Figure 2 illustrates how the mechanism can be used to measure QoS attributes for composite services. In a federated environment, a composite service, for example an online publishing service, may be composed of other services such as editing, advertising, searching and printing services.



Availability of the Composite Service C up to Nth minute

= (INTERSECTION * of UP minutes / N) * 100;

= (7/10)*100 = 70%

* INTERSECTION operation represents the UP minutes which belong to both Service A and to Service B.

If Calculated QoS >= Target QoS

service has Met QoS up to the Nth minute.

Figure 2. Continuous measurement of composite services.

For the composite service to be available, all the constituent services need to be available at the same time. This mechanism uses the set operation INTERSECTION (as depicted in the figure) to determine those minutes for which the composite service is available. The array of 1440 minutes is stacked on top of each other as the services are composed in a bottoms-up approach. For services which are redundant (e.g. two printing services) the set operation UNION is used.

4.2 Measuring Service Configurations QoS

In a service tree, when a child node contains a single-point-of-failure (e.g. the load balancer in clustered database system) the parent service will incur an outage for every minute this single point of failure is down. However, if there is some redundancy built in (e.g. the database servers in the cluster environment), an outage of one or two servers may not impact the service. In essence, each child node is weighted differently with regard to how it impacts its parent node. The QoS impact calculations, therefore, need to differentiate between single-point-of-failures and those with redundancy built in. Additionally, a service may also be defined as 'impacted' only when n out of m servers providing the service are impacted. For example, when 3 out of the 5 database server goes down, then the system may incur serious latency issues, but when 2 out of those same 5 servers are down, the system may run without any problem.

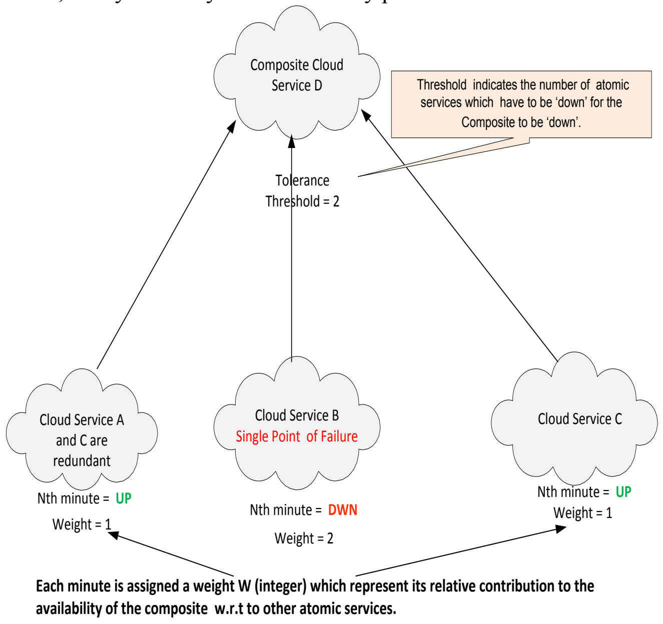

Composite Cloud Service D
Threshold indicates the number of atomic services which have to be 'down' for the Composite to be 'down'.
Tolerance Threshold = 2
Cloud Service A and C are redundant
Nth minute = UP
Weight = 1
Cloud Service B
Single Point of Failure
Nth minute = DWN
Weight = 2
Cloud Service C
Nth minute = UP
Weight = 1
Each minute is assigned a weight W (integer) which represent its relative contribution to the availability of the composite w.r.t to other atomic services.

Figure 3. Use of Weight Vectors.

To account for these various service configurations, each node is given a 'weight' which indicates its criticality to the service it provides to its parent relative to other children. The parent is given a threshold tolerance level. This threshold tolerance indicates the level at which the parent service will incur an outage. The parent service will incur an outage when the sum of the weights of the children (incurring an outage) is greater or equal to this threshold tolerance of the parent.

Single-point-of-failure: The weight of these components is assigned the same value as the tolerance threshold of its parent. In this figure, Cloud Service B is the single-point-of-failure and it will take the Cloud service D down when it goes down (weight = threshold tolerance of parent).

Redundant services: Cloud service A and C are redundant services, so both of them have to be down for its parents to be down. Hence we assign weight such that the sum of their weight equals the tolerance threshold of its parent. Similarly, n out of m server configurations can also be accounted using this weight vector mechanism.

Service Disruptions in a Cloud federation: When service disruptions occur in a federated environment, it is necessary to accurately identify the offending cloud partition so that the other providers contributing to the same service are not penalized unduly. Furthermore, these disruptions must also be classified according to cause of disruption. For example, when a disruption is caused by anomalies in a third-party application hosted by an infrastructure cloud provider, the provider of the cloud infrastructure should not bear its consequence. To accurately identify and classify these types of service disruptions we associate configurable business rules to each of the minute objects. This enables filtering out problem tickets at the minute granularity.

QoS vary by cloud-partitions, providers and clients: One key observation in a federation model is that the service providers have to serve much more diverse clients than traditional enterprise services usually do [9]. To optimize profit, providers will service multiple clients with differing QoS from the same service tree. Furthermore, QoS levels may change at each level of the service tree for the same client. For example, a database service may have to be contractually available 99% of the time, while each of the servers providing the database service may have to be available for 95% of the time, while the load balancer for the servers needs to be up 99% of the time. To represent and measure QoS accurately, these threshold QoS metrics are stored in the day object (Figure 1) for each client and computations are carried out based on the information available in the service disruption for each client at each node in the SLA service tree.

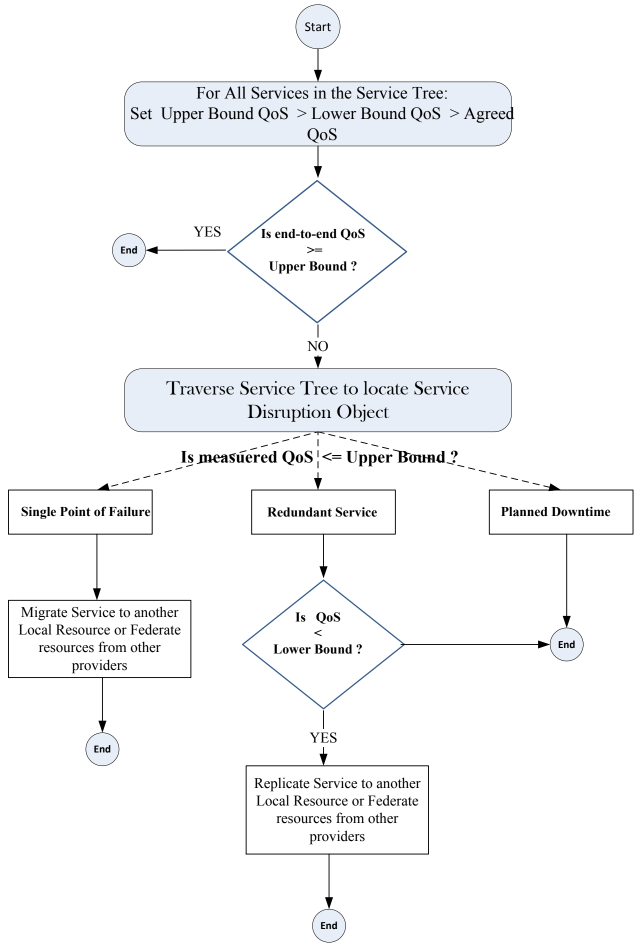
Duplicate tickets elimination in a Cloud federation: Service disruption tickets can be created at device level, application level or at the service level. When SLAs are defined broadly at all these levels, often a secondary problem tickets are created to reflect the scenario more accurately. For example, the fact that a web service is down and one of the 2 servers supporting the service is down needs two distinct tickets to reflect reality [8]. In a federated environment where composite services are constructed from distinct cloud partitions, a mechanism needs to be in place so that providers are not penalized twice for the same outage. To detect duplicate tickets for the same outages, we stack outage minute objects from the problem tickets to the day object tray at each node of the service tree. If there are multiple outage minute objects for the same minute slot, then a duplicate ticket has been detected and they are rejected and logged for further analysis and reporting.

4.3 SLA-Based Provisioning Algorithm

Figure 4 demonstrates an SLA based provisioning algorithm to allocate resources for various service configurations. For each service and its components, set an Upper Bound QoS and a Lower Bound QoS threshold both of which are greater than the agreed upon QoS for that specific Service.

The range between the Lower Bound QoS and agreed QoS reflects a "near-breach" situation and alerts the provider of an impending violation of QoS. It reflects a service quality which has not yet violated the contractual QoS but it is close to breaching it. When the service quality falls below this lower bound, it provides an opportunity to add more resources or migrate the service to another cloud before a complete violation of QoS occurs.

In practice, expected QoS targets can vary from one node to another within the same service-tree. Hence, accurate measurement of both the target QoS and the up-to-the-minute QoS are necessary for all nodes within the service tree. This mechanism of measuring QoS continuously can be used to locate those services which are 'near-breaches' and based on the criticality of the system (single-point-of-failure, redundant etc.), additional resources can be provisioned dynamically to prevent SLA violations. As an example, for single-point-of-failure, as soon as the measured 'up-to-the-minute' QoS falls below the upper bound, live migration to another local resource or to another Cloud in the federation is initiated. For services with redundancy built-in, this may occur only when the measured QoS falls below the lower bound.



Start
For All Services in the Service Tree:
Set Upper Bound QoS > Lower Bound QoS > Agreed QoS
Is end-to-end QoS >= Upper Bound ?
YES
End
NO
Traverse Service Tree to locate Service Disruption Object
Is measured QoS <= Upper Bound ?
Single Point of Failure
Redundant Service
Planned Downtime
Migrate Service to another Local Resource or Federate resources from other providers
End
Is QoS < Lower Bound ?
End
YES
Replicate Service to another Local Resource or Federate resources from other providers
End

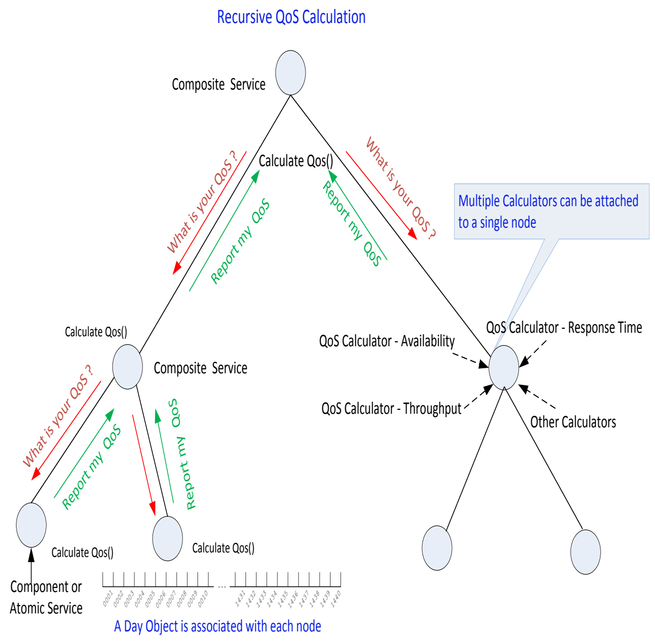
Figure 4. Algorithm for resource allocation, based on 'up-to-the-minute' QoS measurement of different service configurations.

Our assumption is that different cloud providers will either provide access to measure the service quality at each node or implement this technique at each node in their service tree. In order to speed-up processing, service quality at each node is pre-computed with the assumption that a service quality has met the contractual level unless a service disruption occurs. Only when a service disruption occurs, QoS computation is carried out using the representation described in Figure 1. In the software implementation, each minute is represented as an object which is capable of computing its own QoS value (e.g. availability percentage) based on several QoS attributes like down-time minutes, up-time minutes, caused-by field, scheduled/unscheduled outage etc.

'Up-to-the-minute' QoS measurement may not only prevent SLA violations but also optimize resource allocation by provisioning resources only when it is needed.

5. EXPERIMENTS

A prototype design was constructed and implemented to validate whether the mechanisms will hold up to various service configurations. The experiments were conducted on an HP server having the following configuration: 3.59 GHz with 4 GB of RAM running a standard Windows 2003 Server Standard Edition, Service Pack 2. The prototype environment was developed in C#, .NET 4.0 frameworks and the underlying database was SQL server 2008 in a load-balanced mode.



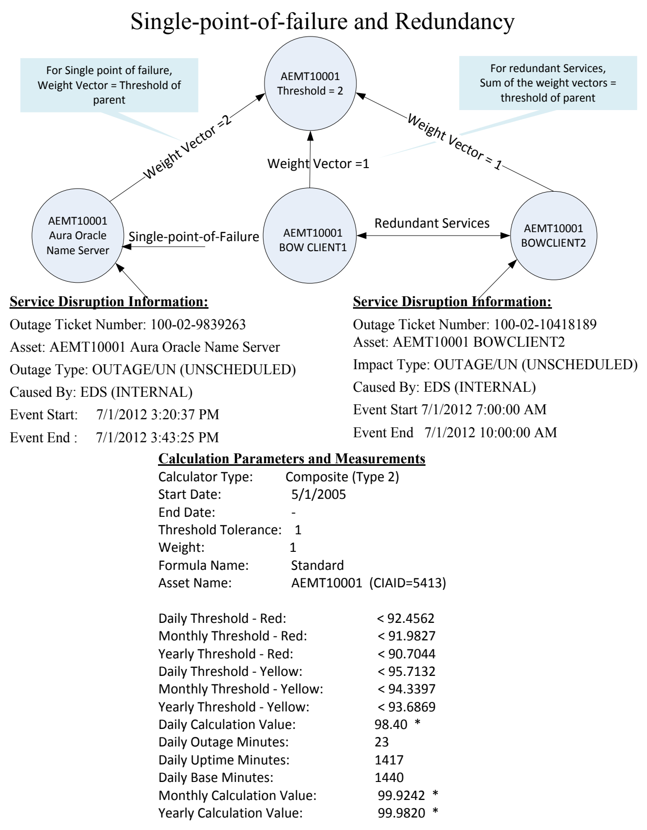
Recursive QoS Calculation
Composite Service
Calculate Qos()
What is your QoS ?
Report my QoS
Report my QoS
What is your QoS ?
Multiple Calculators can be attached to a single node
Calculate Qos()
Composite Service
QoS Calculator - Availability
QoS Calculator - Response Time
QoS Calculator - Throughput
Other Calculators
What is your QoS ?
Report my QoS
Report my QoS
Calculate Qos()
Calculate Qos()
Component or Atomic Service
A Day Object is associated with each node

Figure 5. Recursive QoS calculation up/down the service tree.

The experiment was modeled after a real-world SLA service tree for a customer of Hewlett-Packard. The service tree (see Figure 5) consisted of 848 nodes where each node represented a server or a service. The typical service tree has the physical components (servers, network, and router) towards the leaf of the tree structure while the root represented the composite service. Each of the intermediate nodes represented a sub-system or a service which support the composite service at the root.

Associated with each node is the day object which is represented by an array of 1440 minutes objects to calculate up-to-the-minute QoS at every node. Starting at the root of the tree, the composite service calculates its QoS in a recursive manner based on the QoS of its children. Multiple Calculator Objects can be attached to a single node to compute different types of QoS at each node. The experiment was set up with single-point-of-failure and redundant services at different levels. While the computation logic was constructed at the application layer, the cumulative computation (up-to-the-minute) was done using stored procedure at the persistence layer to speed-up calculation.

The software program was run for each day for the month of June. The total number of tasks calculated was 1635 per day, since some nodes had multiple calculators associated with it. The task took 1.58 minutes to complete on one CPU. The results reveal that the software representation was able to make accurate calculation of the QoS for every minute for availability metrics at each node of the service tree. There were a total of 13 service disruptions during the month of June and the software was able to compute the QoS in terms of availability percentage accurately.


Single-point-of-failure and Redundancy
For Single point of failure, Weight Vector = Threshold of parent
AEMT10001 Threshold = 2
For redundant Services, Sum of the weight vectors = threshold of parent
Weight Vector =2
Weight Vector =1
Weight Vector = 1
AEMT10001 Aura Oracle Name Server
Single-point-of-Failure
AEMT10001 BOW CLIENT1
Redundant Services
AEMT10001 BOWCLIENT2

Service Disruption Information:
Outage Ticket Number: 100-02-9839263
Asset: AEMT10001 Aura Oracle Name Server
Outage Type: OUTAGE/UN (UNSCHEDULED)
Caused By: EDS (INTERNAL)
Event Start: 7/1/2012 3:20:37 PM
Event End : 7/1/2012 3:43:25 PM

Service Disruption Information:
Outage Ticket Number: 100-02-10418189
Asset: AEMT10001 BOWCLIENT2
Impact Type: OUTAGE/UN (UNSCHEDULED)
Caused By: EDS (INTERNAL)
Event Start 7/1/2012 7:00:00 AM
Event End 7/1/2012 10:00:00 AM

Calculation Parameters and Measurements

Calculator Type:	Composite (Type 2)
Start Date:	5/1/2005
End Date:	-
Threshold Tolerance:	1
Weight:	1
Formula Name:	Standard
Asset Name:	AEMT10001 (CIAID=5413)

Daily Threshold - Red:	< 92.4562
Monthly Threshold - Red:	< 91.9827
Yearly Threshold - Red:	< 90.7044
Daily Threshold - Yellow:	< 95.7132
Monthly Threshold - Yellow:	< 94.3397
Yearly Threshold - Yellow:	< 93.6869
Daily Calculation Value:	98.40 *
Daily Outage Minutes:	23
Daily Uptime Minutes:	1417
Daily Base Minutes:	1440
Monthly Calculation Value:	99.9242 *
Yearly Calculation Value:	99.9820 *

Figure 6. Weight Vectors Representing Service Configuration.

Figure 6 reveals actual calculations of SLA availability metrics for a customer of HP. There were two outages on the same day, one on a single-point-of-failure (left node) and one on a redundant server (right node). Since, two nodes on the right are redundant both of them have to incur an outage for the parent to be down. This is achieved by assigning weights to each of them such that their sum of the weight vectors equals the threshold tolerance of the parent. In this case, even though one of the redundant components was down for 3 hours, the parent was up because the sum of their weight of the down minutes is still less than threshold of parent. Also note, how the node on the left is a single-point-of-failure and its weight vector equals that of parents. Hence, the outage of 23 minutes it incurred was cascaded up to its parents.

5.1 Scaling-up in a Single-Datacenter

Experimental Setup: The setup is modeled after one of HP's clients in the travel and hospitality industry, whose assets are located in the Tulsa datacenters. The goal of the experiment was to find out if the measurement mechanism can scale-up when the number of nodes increases within an SLA service tree. The size of the service tree was steadily increased from 100 to 1200 and QoS computational time was recorded for each day, month-to-date, and year-to-date. The calculations were carried out for each minute for each day for a period of one month. The number of service disruptions during the same period was 19.

Results: The results reveal that the software representation of the mechanism scaled-up well when the size of the service tree was steadily increased. Figure 7 depicts a graph where the X axis represents the varying number of nodes in the service tree and the Y-axis represents the time it took to calculate up-time availability for a period of one month for the entire service tree. The result of the experiments also demonstrated that the computational time depended on the number of service disruptions for the period of measurement. This dependency can be explained by noting that the impact of the service disruption at any node needed to be cascaded up the service tree to calculate its impact on the end-to-end QoS.

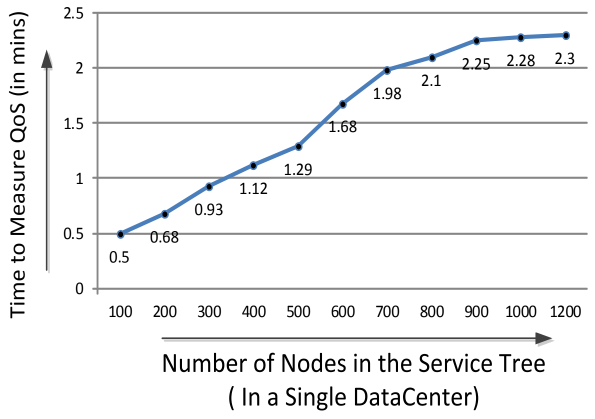


Figure 7. Measurements in a Single Datacenter.

5.2 Scaling-up in a Cloud Federation

Experimental Setup: Cloud partitions were simulated by carving out distinct computational environments within two geographically separate datacenters. The goal of the experiment is to find out if the measurement mechanism can also scale up in a cloud federation environment when the number of cloud partitions was increased. A service tree was modeled after one of HP's clients in the Transportation Industry whose assets are located in the Tulsa Datacenters. The size of the service tree was maintained fixed at 1200 nodes but they were equally distributed among each of the datacenter partitions. The number of partitions was varied from 1 to 8 and the QoS measurement software was run in each partition while sharing a single load-balanced database for storing results.

Results: The results of the simulation reveal that the software calculated the SLA performance metrics successfully in a federated environment. Figure 8 depicts a graph where the X axis represents the number of cloud partitions and the Y-axis represents the time it took to calculate the up-time availability for a period of one month for the entire service tree. As the number of cloud federations were increased the computational time increased linearly, which can be attributed to the fragmentation of service tree amongst different cloud partitions. The calculation results revealed that this measurement technique scaled up in a distributed environment as well. In the future, we intend to carry out this measurement technique in a real cloud federation where live migration of VM's and dynamic resource allocations can take place.

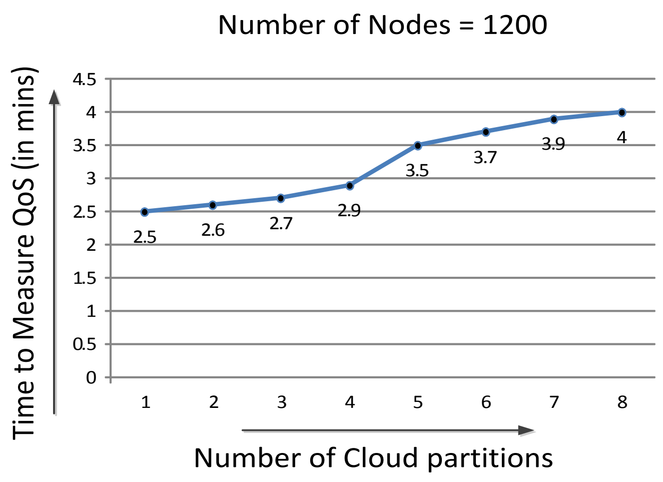


Figure 8. Measurements in a Cloud federation environment.

6. CONCLUSION AND FUTURE WORK

We presented a basic mechanism for representing and measuring QoS continuously for both individual service and composite services. The same mechanism can be used to represent and measure different QoS metrics such as availability, response time, throughput in a continuous manner in a Cloud federation. Development of this basic mechanism is critical to enabling elastic service composition in a Cloud federation model where resource availability is uncertain. We believe that the proposed measurement techniques will not only help in effective allocation of Cloud resources to satisfy QoS targets but also lower operational cost by enabling optimal resource pooling and surplus re-distribution in the highly dynamic federation model. Due to mounting concerns of trust and privacy, the task of measuring and demonstrating SLA compliance may be ultimately delegated to third-parties [7], who may adopt this clear and simple mechanism as a uniform standard of measurement.

In our future work, we shall examine how this continuous QoS measurement technique may prevent costly SLA violations by making services 'QoS-aware' so that autonomic resource management can occur in the changing environment of a Cloud federation.

7. REFERENCES

[1] Rajiv Ranjan, Rajkumar Buyya, Manish Parashar.2011. *Special section on autonomic Cloud computing: technologies, services, and applications*. Concurrency and Computation: Practice and Experience Special Issue: Special section on Autonomic cloud computing: technologies, services, and applications Volume 24, Issue 9, pp. 935–937, 25 June 2012

[2] *Amazon Elastic Compute Cloud (EC2),* http://aws.amazon.com/contact-us/reserved-instances-limit-request/

[3] Nicolas Bonvin, Thanasis G. Papaioannou and Karl Aberer. *Autonomic SLA-driven Provisioning for Cloud Applications.* 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. May 23-26, 2011, Newport Beach, CA, USA, pp. 1-5

[4] Rajkumar Buyya, Rajiv Ranjan, Rodrigo N. Calheiros. *InterCloud: Utility-Oriented federation of Cloud Computing Environments for Scaling of Application Services.* Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010, Busan, South Korea, May 21-23, 2010), LNCS, Springer, Germany, pp. 14-20

[5] Stuart CLAYMAN, Alex GALIS, Clovis CHAPMAN, Giovanni TOFFETTI, 2010. *Monitoring Service Clouds in the Future Internet.* Towards the Future Internet G. Tselentis et al. (Eds.)IOS Press, 2010,pp.121-122

[6] R. Buyya, D. Abramson, and S. Venugopal. *The Grid Economy.* Special Issue on Grid Computing, Proceedings of the IEEE, M. Parashar and C. Lee (eds.), 93(3), IEEE Press, March 2005, pp. 698-714

[7] Pankesh Patel, Ajith Ranabahu, Amit Sheth. *Service Level Agreement in Cloud Computing.* Cloud Workshops at OOPSLA09, 2009*,* pp. 2-4

[8] Shoumen Bardhan, Steve Stevens. *Service Level Agreement Automation.* Hewlett-Packard TechCon'11, March 2011, pp 2

[9] {ychi,hjmoon,hakan,tatemura}@sv.nec-labs.com. *SLA-Tree: A Framework for Efficiently Supporting SLA-based Decisions in Cloud Computing.* EDBT/ICDT '11 Proceedings of the 14th International Conference on Extending Database Technology Pages 129-140. 2011