Irish Centre for Cloud Computing and Commerce (IC4)

**CloudPASS:**

**Quality of Service Calculation**

**Author: Lei Xu**

**Claus Pahl**

**Date: 16th January 2014**

**Status: Version 0.1**

The research leading to these results has received funding from *Enterprise* Ireland and *IDA.*

 IDA Ireland Logo

Table of Contents

1 Introduction 4

2 Related Work 4

2.1 PM/VM Resource Metrics Layer 5

2.2 Service Related Metrics 5

2.3 Business Related Metrics 5

3 Framework Overview 6

3.1 Cloud Trust Label 7

3.1.1 Trust Label 7

3.2 Trust Calculation Engine 8

3.2.1 Data Storage 8

3.2.2 QoS Mapping & Trust Analysis 10

3.2.3 Performance Prediction 14

4 Elasticity Engine 15

5 Implementation 18

6 Quality Analysis in Federated Clouds 19

6.1 Requirements for distributed quality analysis 19

6.2 Challenges for distributed quality analysis 20

6.3 Synchronization solution overview 20

6.4 Fault tolerance and conflict resolution 21

7 Conclusion 21

8 Reference 22

Table of Figures

Figure 1. Cloud Monitoring Metrics. 4

Figure 2. Framework Overview. 6

Figure 3. QoS Mapping and Quality Analysis. 10

Figure 4. The impact of uncertainty in cloud elasticity. 15

Figure 5. Elasticity Engine. 16

Figure 6. Quality analysis in a Federated Cloud. 19

# Introduction

The goal of CloudPASS is to overcome the barriers by building trust in cloud services through a provider-independent evidence-backed trustmark service. It collects state information on cloud services, and dynamically calculates trustmark of the services. This approach offering both assurance and accountability promises to be more effective than those solutions building trust based on legal agreements and manual audits.

The report is structured as follows. Section 2 will review related work in SLA mapping. Section 3 will give an overview of the CloudPASS framework, including the cloud trust label, the monitoring system, the trust calculation engine, and the approach used to predicate the resources/performance of cloud systems required to support given trustmark. Section 3 will introduce the elasticity engine that dynamically adjusts resource provision based on workload. Section 4 will illustrate how the framework is implemented. Section 6 will introduce our solution to tackle the challenges faced by quality analysis in federated clouds. Finally, Section 7 draws a conclusion and discusses the future work.

# Related Work

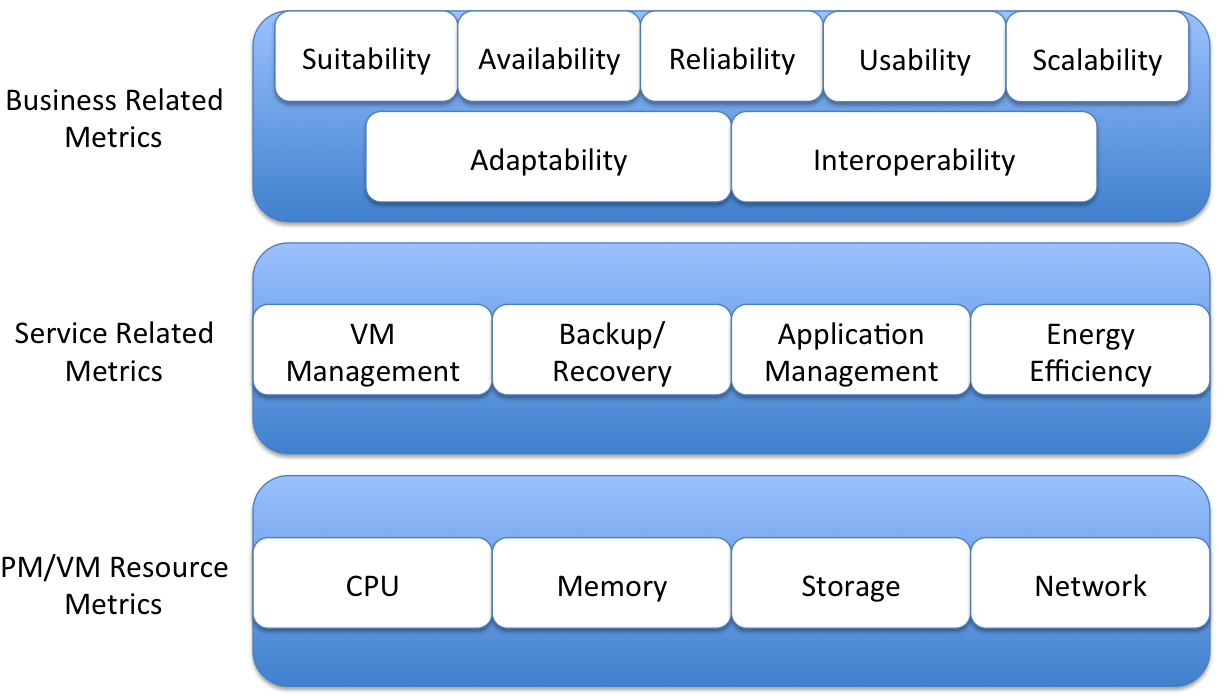


Figure 1. Cloud Monitoring Metrics.

Most cloud monitoring metrics that we can find can be categorized into the layered architecture illustrated graphically in Figure 1.

## PM/VM Resource Metrics Layer

Monitoring metrics of this layer are used to track the physical and virtual hardware resources consumed to provide cloud services. They can be obtained directly from monitoring systems. These metrics mainly focus on the following resources:

1. CPU [1]: CPU utilization, number of CPU
2. Memory [1]: memory size, memory utilization
3. Storage [1][2]: storage capacity, storage utilization, response time, throughput, average read/write speed, random input/output speed, sequential input/output speed
4. Network [2]: round trip time, response time, package loss, bandwidth, through, network utilization, latency

## Service Related Metrics

Service related metrics are applied to track resources consumed by the cloud services, includes the services delivered by IaaS, PaaS and SaaS. Most metrics of this layer are obtained by mapping low-level monitoring records. The metrics categorized into this layer are listed as follows.

1. VM management [2]: migration time, retention of log
2. Backup/Recovery [2]: backup interval, backup type, time to recovery, backup media, backup archive, and retention of log
3. Application management [3]: service response time (average response time, maximum response time, and response time failure), through and efficiency, accuracy, transparency, elasticity, stability
4. Energy efficiency [4]: data center infrastructure efficiency, and power usage efficiency

## Business Related Metrics

The metrics categorized into this layer are related to business goals. All of them cannot be received from monitoring records directly. To obtain these metrics, we have to map low-level monitoring records to high-level business specifications. The business related metrics considered in clouds systems include:

1. Suitability (related to number of non-essential features provided by service, and number of non-essential features required by the customer)
2. Availability (related to service consumption time, service down time)
3. Reliability (related to probability of violation and the promised mean time to failure)
4. Scalability (related to the number of cloud services consumed by a user, and the number of resources assigned to each of the services)
5. Adaptability (related to the time taken to adapt to changes or upgrading the service to a higher level)
6. Interoperability (related to number of platforms offered by the provider number of platforms required by users for interoperability)

# Framework Overview

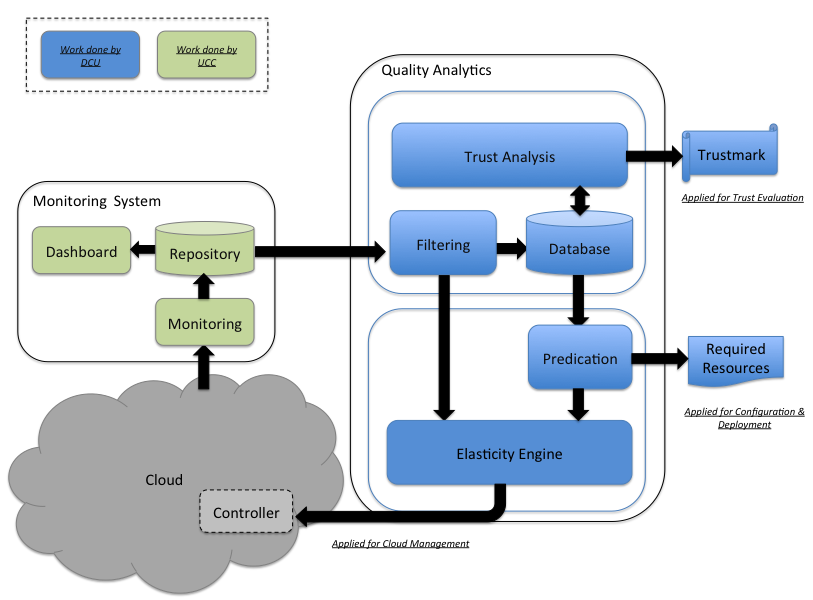


Figure 2. Framework Overview.

The CloudPass framework consists of its monitoring system, the trust calculation engine, and the elasticity engine, which is depicted graphically in Figure 1. The monitoring system is responsible for collecting low-level monitoring records directly from the cloud system, in which services are consumed. The collection is based trust labels specified by system administrators. Collected records are stored in the repository of the monitoring system, which can be displayed on the dashboard. Periodically, the Filtering – a sub-component of the trust calculation engine receives monitoring records from the repository. The raw records are then correlated and aggregated, and stored in the database. Another sub-component named Quality Analysis also receives trust labels associated with SLA from the system administrators. Based on the input, Quality Analysis consumes processed records stored in the database and performs trust calculation. The output of the calculation will be stored in the database to form the “learning set”. Once enough history records on trustmark calculation have been gathered, the Prediction component could predict the resources required by support given services to build trust. In addition, the output of the Prediction will form the knowledge of the elasticity engine, which dynamically scales resource offering to reduce the cost of service and avoids SLA violations.

## Cloud Trust Label

In this section, we will introduce our cloud trust label that specify the parameters considering in trust calculation, and how the three sub-components of the trust calculation engine cooperate with each other to perform QoS mapping, quality analysis and resource prediction.

### Trust Label

To develop a method of calculating trustworthiness, we deploy the Delphi process that involves 37 participants with different background. They are cloud service providers, cloud service users, legal experts or privacy experts. The output of this research sets up the cloud trust label used in this framework. The trust label includes the following the parameters:

* Data security\*\*
* Certification\*
* Service levels\*
  + Service uptime\*\*
  + Internal network uptime\*\*
  + External network uptime \*\*
  + Availability of dynamic load balancing\*\*
  + Cloud storage service availability\*\*
  + Primary DNS availability \*\*
  + Sever reboot time★
* Variation of terms\*
* Data portability (onboard & offboard)\*
* Backup of data\*
* Data location\*
* Ownership (data, meta data, service customization)\*
* Sharing of data (commercial & legal)\*
* Insurance levels\*
* Audit approvals\*
* Customer service level\*
  + Emergency support response time★
  + General support response time★
  + Engineering support available time\*
  + Physical security available time\*

\*: These parameters are collected based on the trust label definition. They are offered by the monitoring system from the service/system configuration.

\*\*: These parameters can be calculated based on low-level monitoring records. The formulae of mapping are illustrated in section 2.2.2 .

★: These parameters are offered by the monitoring system, which can be obtained directly from its records.

## Trust Calculation Engine

The trust calculation engine collects and aggregates monitoring records. The processed records are stored in its database. Based on the data, it can map low-level monitoring parameters to high-level QoS, calculate truskmark, and predicate required resources. The functionalities of the trust calculation engine are explained as follows.

### Data Storage

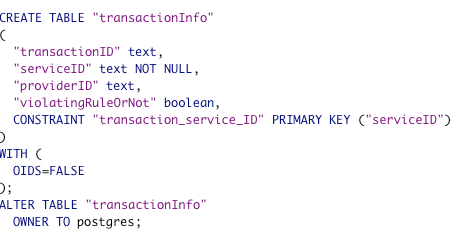
The trust calculation engine is supposed to receive service level and infrastructure level records from the monitoring system. The service level data will be applied for QoS mapping and quality analysis, whilst the infrastructure level records will be consumed for the prediction of resource consumption. Records offered by the monitoring system should be aggregated based on a 5-minute time interval.

In this work, we assume each service instance is associated with its *transactionID* and *serviceID*. The *transactionID* uniquely identifies a service transaction; the *serviceID* uniquely identifies a service been involved in a transaction in the entire environment -- the *serviceIDs* of a given service are different when it invoked in different transactions.

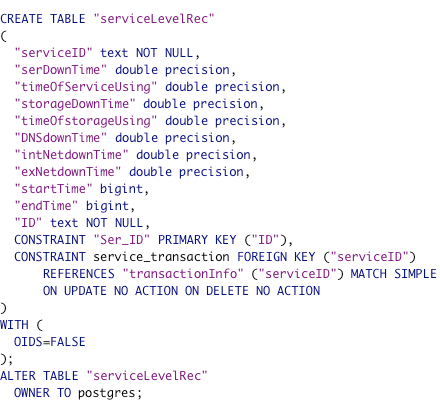
We decide to deploy a rational database in our solution. This is because the time intervals considered in trust calculation are different to that of those service level records offered by the monitoring system. Therefore, we have to perform aggregation. Meanwhile, to predicate the resources required to support given QoS, we have to retrieve infrastructure level records, and then process them to form service invocation characteristics – a range of values on system performance. Although, NoSQL solutions are famous for their scalability and performance, rational database systems and SQL are suitable for our operations since they are good at data enquiry and analysis.

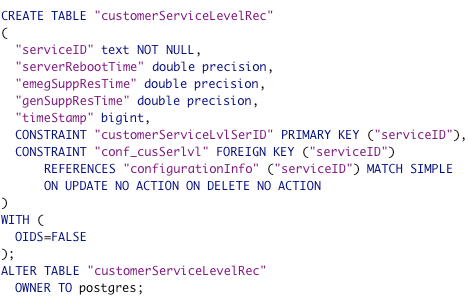
Based on the above, the monitoring data are stored in 5 tables in the database. The records of table “transactionInfo”, “serviceLevelRec” and “customerServiceLevelRec” are the states of services collected by the monitoring system. They will be applied for service level calculation. Meanwhile, the records of table “configurationInfo” are the configuration of the services consumed by a given customer, which will be considered in trust label evaluation. In addition, the data stored in table “infraLevelRec” are the infrastructure level information offered by the monitoring system that indicate the resource required to support certain service level. The data will be applied in resource prediction.

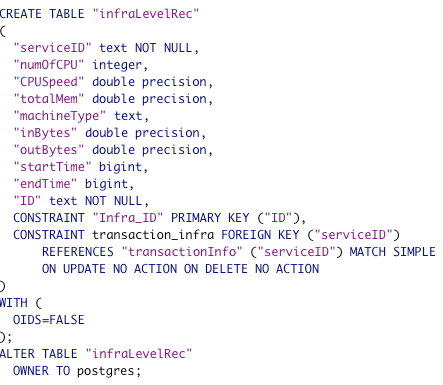
The SQL to create the tables are given as follows.











### QoS Mapping & Trust Analysis

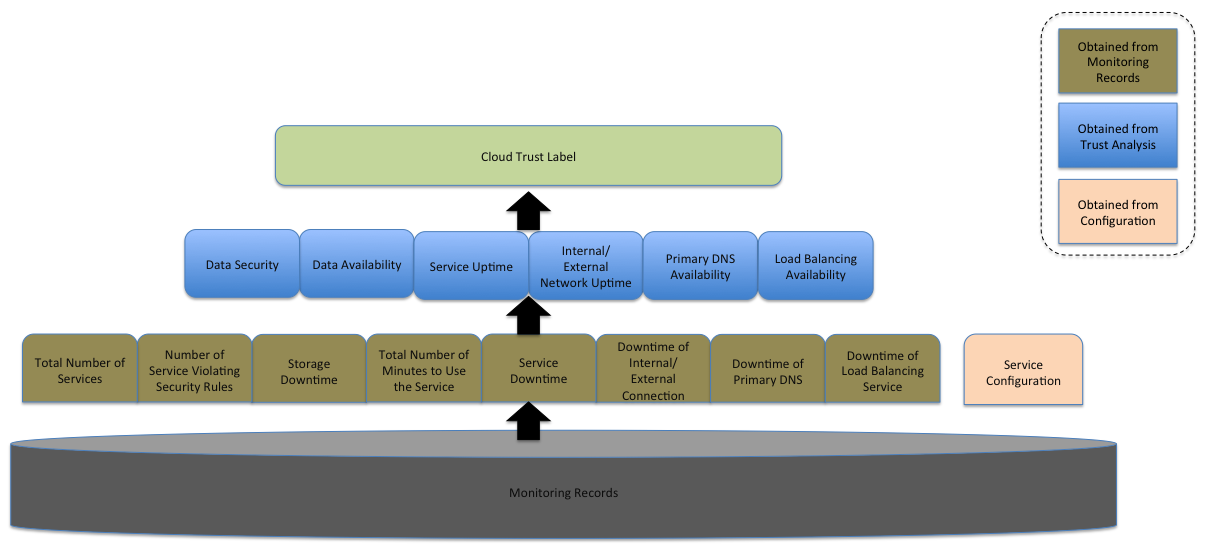


Figure 3. QoS Mapping and Quality Analysis.

The trust calculation workflow is depicted graphically in Figure 2. It will map monitoring records to QoS information, and then calculate the trust mark. Initially, processed monitoring records stored in the database are mapped to QoS terms based on a given time interval. The mapping process can be defined as follows.

1. Service Uptime
2. Definitions
3. “Total Number of Minutes to Use the Service” is the total accumulated minutes during a given time interval for which the cloud service is used.
4. “Service Down Time” is the total accumulated minutes during which the service is marked as unavailable. If a customer’s initiated attempts to use the service fail (5XX server Errors) then the 1-minute time interval is marked as unavailable. Failures caused by the network at the site used by the customer to connect the service are not included.
5. “Service Uptime” for a given service is calculated by subtracting from the Service Down Time divided by Total Number of Minutes to Use the Service. Service Uptime is reflected by the following formula:
6. Storage Availability
7. Definitions
8. “Storage Down Time” is the total accumulated minutes during which the cloud storage is marked as unavailable. If a customer’s initiated attempts to use the data complete unsuccessfully then the 1-minute time interval is marked as unavailable. Failures caused by the network at the site used by the customer to connect the service are not included.
9. “Total Number of Minutes to Use the Service” is the total accumulated minutes during a given time interval for which the cloud service is used.
10. “Storage Availability” for data stored in the cloud is calculated by subtracting from the Storage Down Time divided by Total Number of Minutes to Use the Service. Storage Service Availability is reflected by the following formula:
11. Data Security
12. Definitions
    * + 1. “Total Number of Services” is the total number of services constituting a transaction.
        2. “Number of Services Violating Security Rules” is the number of services that violate security rules in a transaction during the lifetime of the transaction. Data Security is reflected by the following formula:
13. Internal Network Uptime
14. Definitions
15. “Down Time of Internal Connection” is the total accumulated minutes during which the cloud service has no internal connectivity.
16. “Total Number of Minutes to Use the Service” is the total accumulated minutes during a given time interval for which the cloud service is used.
17. “Internal Network Uptime” is calculated by subtracting from the Down Time of Internal Connection divided by Total Number of Minutes to Use the Service. Internal Network Uptime is reflected by the following formula:
18. External Network Uptime
19. Definitions
20. “Down Time of External Connection” is the total accumulated minutes during which the cloud service has no external connectivity.
21. “Total Number of Minutes to Use the Service” is the total accumulated minutes during a given time interval for which the cloud service is used.
22. “External Network Uptime” is calculated by subtracting from the Down Time of External Connection divided by Total Number of Minutes to Use the Service. External Network Uptime is reflected by the following formula:
23. Primary DNS Availability
24. Definitions
25. “Down Time of Primary DNS” is the total accumulated minutes during which the DNS service is marked as unavailable.
26. “Total Number of Minutes to Use the Service” is the total accumulated minutes during a given time interval for which the cloud service is used.
27. “Primary DNS Availability” is calculated by subtracting from the Down Time of Primary DNS divided by Total Number of Minutes to Use the Service. Primary DNS Availability is reflected by the following formula:
28. Load Balancing Availability
29. Definitions
30. “Down Time of Load Balancing Service” is the total accumulated minutes during which the load balancing service is marked as unavailable.
31. “Total Number of Minutes to Use the Service” is the total accumulated minutes during a given time interval for which the cloud service is used.
32. “Load Balancing Availability” is calculated by subtracting from the Down Time of Load Balancing Service divided by Total Number of Minutes to Use the Service. Load balancing availability is reflected by the following formula:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Performance (Can I measure?) | Policy (Is there a policy?) | Preference (Can I modify?) |
| Data Security | Y | Y | N |
| Certification | Y | Y | N |
| Service Levels | Y | Y | N |
| Variation of Terms | Y | Y | Y |
| Data Portability (Onboard) | Y | Y | Y |
| Data Portability (Offboard) | Y | N | N |
| Backup of Data | Y | Y | Y |
| Data Location | Y | Y | Y |
| Ownership (Data) | N/A | Y | Y |
| Ownership (Meta Data) | N/A | Y | Y |
| Ownership (Service Customization) | N/A | Y | Y |
| Ownership (Application Customization) | N/A | Y | N |
| Sharing of Data (Commercial) | N | Y | Y |
| Sharing of Data (Legal) | Y | Y | N |
| Insurance Level | Y | Y | Y |
| Audit Approvals | Y | Y | Y |
| Customer Service Level | Y | Y | Y |

Table 1. Service Configuration Requirements.

The configuration records of all services that are stored in the database should meet the requirements specified in Table 1, which is set up based on our trust label. Meanwhile, the output of QoS mapping based on the equations should also satisfy the service level requirements specified in the trust label. The service level of a service is defined by a number of parameters, such as service uptime, internal and external network uptime. For simplicity sake, we could define SLA in multiple classes in this framework, including Gold, Silver, Bronze and Fail. Each of them contains the service level related to our cloud trust label, but the detailed requirements are varied in different classes. The setting of each class is given in Table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Gold** | **Silver** | **Bronze** | **Fail** |
| Service Uptime (%) | 100 | >99.99 | >99.9 | ≤99.9 |
| Storage Service Availability (%) | 100 | >99.99 | >99.9 | ≤99.9 |
| Data Security (%) | 100 | >99.99 | >99.9 | ≤99.9 |
| Internal network uptime (%) | 100 | >99.99 | >99.9 | ≤99.9 |
| External network uptime (%) | 100 | >99.99 | >99.9 | ≤99.9 |
| Primary DNS availability (%) | 100 | >99.99 | >99.9 | ≤99.9 |
| Load balancing availability (%) | 100 | >99.99 | >99.9 | ≤99.9 |

Table 2. SLA Class Definition.

When a customer signs a contract with the provider, the required SLA class should be clearly identified. Based on the SLA class definition, the trust calculation engine will evaluate QoS offered to the customer periodically.

As we discussed above, monitoring records are collected every 5 minutes. However, the time interval considered in quality analysis can be different. Therefore, the trust engine has to aggregate the data stored in the database. The aggregation is performed based on the ‘*startTime*’ and ‘*endTime*’ of both service-level and infrastructure-level usage records. Then, it will compare the received values with the service configuration requirements and specification of those SLA classes. If all the values meet the requirements of SLA class, then SLA mark of the service can be identified; if only part of the parameters satisfies the requirements, then the SLA mark will be downgraded to the lower class. For example, if internal network uptime of a service is 99.92% whereas its configuration and other QoS meet the requirements for the Gold class, then the QoS of the service will be marked as Silver. If this SLA mark is satisfied with what identified in the contact or better than that, then the provider will be marked as “trusted”. Otherwise, the provider will be considered as “untrusted”. Note that if the SLA class of a service is Fail, then its provider will automatically marked as “untrusted”.

### Performance Prediction

We assume most services have relatively fixed service invocation pattern (SIP), and a SIP is a group of service invocation characteristics (SIC) whose values are a range. Under a given SIP, service QoS keeps steady and thus the trust mark of services are stable. Based on above, we can have an invocation Pattern-QoS Matrix for services to indicate how SIP is correlated with QoS. As mentioned above, we define a number of QoS class, and apply them in the matrix (so we can consider multiple parameters instead of response time only). Furthermore, we specify a table that indicates how QoS classes of services are related to trust mark for users. This enable the loose couple between trust mark, QoS parameters and user requirements*.*

We need to collect enough pattern-QoS-service information into the database of the trust calculation engine. After that, once a user submits a service request, we can find out the required QoS level from the table based on the user profile and the service type. Then, if there is QoS information matched in the database, we can find the related SIP from the matrix, and then identify the resources required to support the service offering. Otherwise, we perform SIP prediction based on collaborative filtering.

# Elasticity Engine

Cloud elasticity is the degree to which a cloud-based application is able to adjust to workload dynamics by provisioning and deprovisioning resources on the fly, such that at each point in time the available resources, which are obviously costly, match the current demand as closely as possible, see Figure 3. Auto scaling is the state-of-the-practice solution for realizing cloud elasticity. It is particularly beneficial for applications whose usage is difficult or impossible to predict. When load increases, more resources are allocated to accommodate the need. The result is a good user experience, regardless of load. When resources are not needed, they are scaled back. This means good resource utilization and provisioning, which minimizes the cost of the application. It is difficult to provision resources with extremely high precision, but even a relatively simple auto-scaling process can lead to improved resource utilization without compromising the user experience.

However, cloud-based applications are deployed in an uncertain and dynamic environment and, as a result, the resource provisioning is prone to uncertainty. This uncertainty in the context of Figure 3 can be interpreted as the change in the area embedded between resource supply and demand. This areas certainly has a direct relation with cost of ownership as well as user experience. Unfortunately, although cloud elasticity solutions are abundant in the research community, current solutions do not handle the uncertainty and as a result, the only available solutions for elasticity that are adopted in industry is rule-based auto-scaling mechanisms, such as auto-scaling in Amazon EC2, Microsoft Azure and Google Compute engine platforms.



Figure 4. The impact of uncertainty in cloud elasticity.

The elasticity engine in this work is responsible for automatically adjusting resources on the fly. It is intended to make full use of resources and reduce the cost of service offering. This solution component uses fuzzy logic to reason about elasticity under uncertainties. As depicted graphically in Figure 3, the elasticity engine in our framework is a type-2 fuzzy logic system. It consumes the processed monitoring records to track states of services while its elasticity control rules are specified based on the prediction results from the trust calculation engine. Resource allocator will consume the output of the elasticity engine. The controller runs on behalf of cloud-based software and drives actuators to acquire/release nodes based on service status and environmental conditions.



Figure 5. Elasticity Engine.

When an auto-scaling solution is adopted, stakeholders of the system must define how you want to scale in response to changing conditions. For instance, let us consider a cloud application that currently runs on two instances. The system administrator may want to launch two additional instances when the load on the running instances reaches 80 percent, and then she want to terminate the additional instances when the load goes down to 20 percent. The auto-scaler can be configured to automatically scale up and then scale down based on specifying these conditions. However, there are some critical challenges regarding this approach:

* Challenge 1. *Parameters’ value prediction ahead of time*. The process of acquiring and releasing resources is not instant. First, the auto-scaling controller needs to invoke the cloud platform to initiate the acquisition process. During acquisition process, which may take on average 10 minutes, the cloud application is vulnerable to workload increase and as a result provide user dissatisfaction.
* *Challenge 2. Qualitative specification of thresholds.* The specification of the rules requires careful setting out of the lower and upper thresholds. This requires deep knowledge about the behavior of the system over time. Therefore, the overall accuracy of the policies remains subjective, which makes the resource provisioning prone to uncertainty.
* *Challenge 3. Robust control of uncertainty*. The monitoring data contain noise because no sensors or measurement technique are free of noise. This results in the oscillations for resource allocations if not handled properly.

The existing academic solutions on auto-scaling are abundant. However, none of them except rule-based approaches found their way to wide adoption by commercial cloud platforms. We associate this with the ignorance of uncertainty in their solutions and obviously, they become risky when it comes to commercial adoptions for risk-averse application areas. Our solution for auto-scaling (we call it RobusT2Scale) has three distinguishing benefits. Firstly, it enables qualitative rule specification through a well-defined methodology (cf. challenge 2). It can also handle measurement noises and robustly adjust resources (cf. challenge 3). Additionally, conflicting scaling policies can be handled within our approach. This can prepare tradeoffs to decide an appropriate action based on a proactive estimation of workload (cf. challenge 1). To the best of our knowledge, this is the first work based on type-2 fuzzy controllers for the problem of dynamic resource provisioning in the cloud. In other words, our solution tackles the problem of dynamic allocation of resources for cloud-based applications facing unpredictable workloads to decrease cost of ownership without violating SLAs. Our solution is a hybrid elasticity controller to adjust the required resources when the application is running. The notable novelty of our approach is to enable qualitative specification of elasticity rules. A secondary benefit is that the elasticity controller can handle conflicting rules. The proposed controller is also robust against noisy measurements.

RobusT2Scale is independent from both the underlying cloud platform and specific cloud-based application, and allows cloud service providers (e.g., SaaS application owners) to easily integrate it with their own application. RobusT2Scale can be deployed either in the cloud, preferably in a separate node from the application, or on premise. Therefore, not only cloud service providers can be the potential users of RobusT2Scale but also cloud platform providers (e.g., commercial or open platforms) and application providers (e.g., mobile applications (e.g., Instagram, Snapchat)) can adopt and use this auto-scaling solution. This solution has the potential to provide third party scaling services for other cloud services similar to the one that is already offered by RightScale Company.

The design of RobusT2Scale includes only few parameters. The prediction techniques can be used without offline training because it takes advantage of online incremental learning. This reduces the upfront efforts required for configuring RobusT2Scale for building elastic applications. This along with the fact that RobusT2Scale requires few design parameters increases the chance for the adoption of this approach. Note that RobusT2Scale is evaluated on one commercial cloud platform (i.e., Microsoft Windows Azure) and one open source platform (i.e., OpenStack) with real world workload patterns to make sure that the auto-scaler works properly in real-world settings.

# Implementation

The programming language used in the work is Java whilst the database system we deployed is PostgreSQL 9.2. Meanwhile, the communication between the monitoring system and trust calculation engine, the trust engine and the elasticity engine, and between different sub-components of the trust calculation engine is based on RESTful web services. The Jersey – RESTful web services in Java, is used for the communication.

The trust calculation engine consists of the following java packages:

***cloudpass.datamodel***

This package includes the data objects consumed by the trust calculation engine. It mainly consists of:

*TransactionInfo.java* -- store the information related to the service transaction consumed by a given user

*ServiceConf.java* -- store service configuration information requested by the trust label

*ServiceLevelRec.java* -- store the customer service level specified in the trust label

*InfraLevelRec.java* – store the infrastructure level performance data that record the infrastructure resources consumed to support the service transaction

*ViolationRec.java* – store the records of the services that violate the data security requirements

*LabelStatus*.java – store the service quality evaluation results

***cloudpass.server***

This package only includes the *Main.java*, which is entrance of the code. It receives the records from the monitoring system while processes the collected data and waits for requests from data consumers.

***cloudpass.SLAclass***

This package includes *Gold.java*, *Silver.java*, *Bronze.java* and *Fail.java*, which specify the definition of the SLA classes: Gold, Silver, Bronze and Fail respectively.

***cloudpass.storage***

This package includes *StorageManager.java* that is the storage interface for service configurations, and service-level, infrastructure-level and transaction-level records. The interface is implemented by *PostgreSQLStorageManager.java* and *MemoryStorageManager.java*. The former one stores records into and retrieve data from the PostgreSQL database. The aggregation of monitoring records based on the time interval is performed with SQL operations. The latter one process data in the memory that is only used for test purposes.

# Quality Analysis in Federated Clouds

In our current solution, we assume all services can be supervised by a single monitoring system. However, it is not always the case in practice. Ideally, services constituting a transaction may be hosted in multiple sites in a federated cloud. All the cloud sites offering monitoring of resources, however, there are many different and incompatible monitoring systems in current use and this causes integration problems. One solution to tackle this challenge is to deploy the trust calculation engine in every cloud site hosting the service we will evaluate, and enable these quality analysis components cooperate with each other to obtain a transaction level overview on the services constituting the service transaction. Data consumers can obtain the transaction level quality analysis from an arbitrary site constituting a federated cloud.

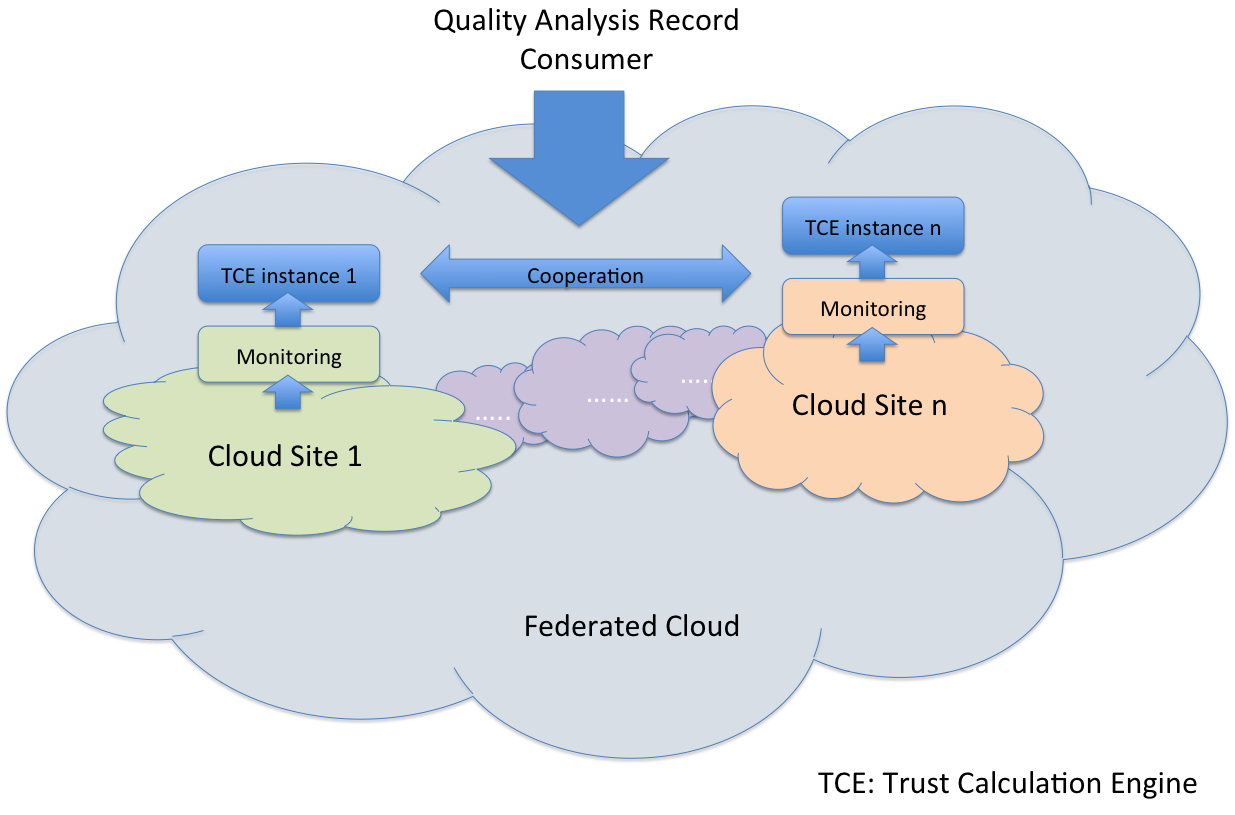


Figure 6. Quality analysis in a Federated Cloud.

## ****Requirements for distributed quality analysis****

Our quality analysis is performed at regular intervals for service consumed during pervious time interval. The improved trust calculation engine deployed on multiple sites of a federated cloud should fulfill the following general requirements:

1. Quality analysis results must be collected from all sites and then combined into a global one for a given service transaction.
2. No analysis results can be left unaccounted. Unaccounted records (due to network delays or instance failures) from a previous period must be accounted in a following period.
3. Each analysis result must be accounted in exactly one time interval.
4. Analysis results for a given time interval for a given data consumer must be identical for any request without regard of the cloud site handling the request. This means that consumption request on the quality analysis for a given time interval must be idempotent at all times.

## ****Challenges for distributed quality analysis****

**Quality analysis results are generated continuously due to constant service offering. Maintaining the results on all trust calculation engine instances current and consistent in near real time is challenging because results need to be propagated across geographically distributed instances. If all updates are propagated to all instances in near real time, as in approaches such as two-phase commit, the delay and network congestion experiences at instances processing requests can be significant. Asynchronous updates are preferred as long as they gradually reach all trust calculation engine instances. Moreover, when updates are propagated asynchronously, each instance can wait to collect a large amount of records and compress them in order to save network bandwidth and reduce update frequency.**

**The update propagation frequency depends on the time interval to perform quality analysis. Short intervals increase the possibility of accounting all results within the time interval next to the one within which the results are generated, and results in frequent propagation. Contrarily, long intervals means updates can be less frequent during most time of those long intervals until almost reaching the end of the intervals.**

**One possible solution for propagating updates is to collect analysis results from each instance only when a consumption request on quality analysis is received. However, this requires fetching large amounts of data from other instances before serving the request that delays the response. This problem may be aggravated due to saturated instances serving multiple different requests, or slow network or processing speed.**

**Another solution is to solve the problem is to use eventual consistency. The use of eventual consistency, coupled with asynchronous communication, facilitates the design of highly scalable, failure tolerant distributed systems. These systems produce low latency since any instance can serve a request immediately. This mechanism ensures consistency that promises the internal state of all instances will eventually be consistent sometime in the future. However, eventual consistency only guarantees that subsequent operations will obtain either the same value as in previous ones or more recent values. To apply the eventual consistency in our work and ensure all requirements listed in section 6.1 are satisfied, we need to enhance the mechanism.**

## ****Synchronization solution overview****

**Our solution to tackle distributed quality analysis builds on eventual consistency but provides the extra support requested to satisfy the requirements for quality management in a federated cloud. Therefore, we can not only manage analysis results correctly but also build a highly scalable and fault tolerant system. Our solution for synchronizing records across cloud sites can be viewed as a stronger form of monotonic read, i.e., eventual consistency with an added requirement for protecting that any successive retrieve of a given data set always returns the same result or a more recent one.**

**In our system mode, a federated cloud is composed of multiple sites that are geographically distributed and communicate over the Internet. Meanwhile, the monitoring systems of these cloud sites are not** compatible. The enhanced trust calculation engine is composed of several autonomous instances hosted in each of the cloud sites. These instances are able to receive monitoring records from the monitoring system of the sites. Instances communicate among each other through message exchange. A customer can enquire an arbitrary instance to obtain quality analysis results on given services.

**Our solution consists of two steps:**

***Analysis results ordering***

**In our work, trust calculation engine instances periodically synchronize analysis results among each other using eventual consistency. As we discussed in section 6.2, eventual consistency along cannot meet all the requirements given in section 6.1. Therefore, to tackle the issues, we first order the records exchanged between instances. At each instance, records are sorted by using Lamport’s logical clocks [5] and delivered based on FIFO order.**

***Results synchronization***

**The synchronization part is composed of two tasks: analysis result management and analysis interval management. Analysis result management involves listening local monitoring events, performing quality analysis, forwarding local results to other instances, and storing records from other instances based on FIFO order. Analysis interval management is intended to maintain the consistency of quality analysis information requested by data consumers. It involves tracking and exchanging information on the records that are accounted in every time interval, and identifying the records that are not accounted in their appropriate interval due to network latencies.**

## ****Fault tolerance and conflict resolution****

It is not uncommon to see errors occurred in a distributed system, such as our distributed quality analysis solution. Therefore, our work investigates how faults will affect the synchronization and consistency of data, and how we can handle the faults. Our solution focuses on the following failures, which can be observed in our work.

1. The request from consumer application to the trust calculation engine is lost.
2. An instance of the trust calculation engine crashes after receiving the consumption request.
3. The reply from the trust calculation engine to the consumer application is lost.
4. The consumer application crashes after sending the request.

# Conclusion

In this report, we have introduced our cloud trust label and the CloudPass framework. Our framework is intended to evaluate QoS of services, and offer assurance and accountability in a single cloud. It processes raw monitoring records and stores the aggregated data in its database. Then the processed data are consumed periodically to calculate the quality of services. The output of is not only used to evaluate the trust level of service providers, but also to form the learning set to predicate the resources required to support given SLA, and to specify fuzzy rules for our elasticity engine. Based on the prediction results, the framework is able generate messages for cloud controllers to dynamically scale resource offering. Furthermore, this report outlines our solution to enhance the framework and make it suitable for quality analysis in federated clouds. In addition, the framework to perform trust calculation in single clouds in implemented in Java, and whose components are loose coupled – communicate with each other via RESTful web services. Therefore, all components can be easily updated or replaced in future without affecting others.

# Reference

[1] Automatic, multi-grained elasticity-provisioning for the Cloud, online available at: <http://www.celarcloud.eu/wp-content/uploads/2013/11/Cloud-Monitoring-Tool-V1.pdf>

[2] Frey, S., L¨uthje, C., Reich, C., Key performance indicators for cloud computing SLAs, The Fifth International Conference on Emerging Network Intelligence, EMERGING 2013

[3] S. K. Garg, S. Versteeg, and R. Buyya. 2013. A framework for ranking of cloud computing services, Future Generation Computer Systems. 29, 4 (June 2013)

[4] Green data centers monthly newsletter, Feb 2010

[5] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Commun*. ACM, vol 21, no. 7, pp 558-565, July 1978