Quality-of-Service Data Warehouse for the Selection of Cloud Service: A Recent Trend

Ahmad Karawash[1, 2], Hamid Mcheick[1] and Mohamed Dbouk[2]
{ahmad.karawash1, hamid_mcheick}@uqac.ca, mdbouk@ul.edu.lb

1: Department of Computer Science, University of Quebec at Chicoutimi (UQAC), 555 Boulevard de l'Université Chicoutimi, G7H2B1, Canada.
2: Ecole Doctorale des Sciences et de Technologie, Lebanese University, Rafic-Hariri Campus, Hadath-Beirut, Lebanon.

Abstract Cloud computing presents an efficient managerial, on-demand and scalable way to integrate computational resources. However, existing Cloud is increasingly transforming the information technology landscape, and organisations and businesses are exhibiting strong interest in Software-as-a-Service. This enables application service providers to lease data centre capabilities for deploying applications depending on Quality of Service (*QoS*) requirements. However, it still remains a challenging task to provide *QoS* assured services to serve customers with best quality, while also guaranteeing the maximisation of the business objectives to service provider and infrastructure provider within certain constraints. In order to address these issues, this chapter proposes building a Data Warehouse of *QoS* to achieve better service matching and enhance dynamic service composition. The proposed *QoS* Data Warehouse model supports the following: ensures a deep analysis of the service's interior structure and properties through online database analysis; facilitates reasoning about complex service weakness points; supports visual representation of analysis results; and introduces a new *QoS* factor for study.

Keywords: Cloud service, Data Warehouse, Quality of Service.

1. Introduction

Cloud computing is a model for allowing expedient, on-demand network access to a shared collection of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly released with minimal management effort or service provider interaction. Cloud computing promotes availability and is composed of three service models. These services in industry are respectively referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Clouds aim to power the next generation data centres by exposing them as a network of virtual services (hardware, database, user-interface, application logic) so that users are able to access and deploy applications from anywhere in the world on demand at competitive costs depending on users' QoS requirements [1].

Cloud computing presents an efficient managerial, on-demand and scalable way to integrate computational resources. However, existing Cloud architecture lacks the layer of middle-ware to enable dynamic service composition. Service composition provides a current technology for developing complex applications from existing service components. Prediction of the *QoS* of Composite Services makes it possible to determine whether the composition meets the non-functional requirements [2]. Previous researches have focused on service composition and integration in terms of services, orchestration and choreography.

As SaaS gains greater acceptance, user cloud expectations start moving from best-effort service to guaranteed service. Hence, it is foreseen the development of QoS as a dominant consideration for cloud service adaptation. QoS has many facets which depend on the aspect that is crucial for the user. Application specific performance includes, for example, response time or throughput, application security varying from data integration and consistency to privacy and service availability, which are some of the QoS considerations that clouds need to address. Such qualities are of interest to service providers and service consumers alike. They are of interest to service providers when implementing multiple service levels and priority-based admission mechanisms. The agreement between the customer and the service provider is referred to as the Service Level Agreement (SLA). An SLA describes agreed service functionality, cost and qualities [3]. This work proposes building a Data Warehouse of QoS to manage the matching between customer and service provider. The obtained Data Warehouse gives a better analysis level, reasoning and decision-taking before selecting a cloud service.

This chapter is organised as follows. Section 2 describes some previous methods of service's selection. Section 3 discusses the service selection structure. In section 4, Quality of Service Data Warehouse model components are introduced, and section 5 highlights the model's benefits. The model simulation and the results are shown in section 6. As conclusion, the main ideas of this chapter are summarised and future perspectives considered.

2. Background

QoS has received much interest in cloud service research because of the rapid increase of the number of services and the approximate equal qualities of the discovered services. Several research activities focused on how to benefit from the *QoS* in the service selection process. Some of these studies sought to extend the Universal Description, Discovery and Integration (*UDDI*) Registry to support service consumers by comprehensible *QoS* information. First, it is relevant to mention the service selection algorithms used by the *QoS broker* for sequential composite flow models with only one *QoS* constraint (i.e. *Throughput*). There are two main approaches we can use to select the optimal services for each component of a business process. The first approach is the combinatorial approach [4], modelling the problem as a Multiple Choice Knapsack Problem (*MCKP*). In order to solve the *MCKP*, three methods are proposed: exhaustive search, dynamic programming and a minimal algorithm for *MCKP* and performance study method. The second approach is the graph approach, modeling the problem as the constrained shortest path problem in the graph theory. The proposed methods to solve the shortest path algorithm are: Constrained Bellman-Ford (*CBF*),

Constrained Shortest Path (*CSP*) and Breadth-First-Search (*BFS*). Also, there are a number of other research studies that dealt with the service selection problem. Keskes et al. proposed a model of automatic selection of the best service provider, which is based on mixing context and *QoS* ontology for a given set of parameters of *QoS* [5]. In 2010, Raj and Saipraba proposed a service selection model that selects the best service based on *QoS* constraints [6]. Squicciarini et al. (2011), furthermore, studied the privacy implication caused by the exchange of a large amount of sensitive data required by optimised strategies for service selection [7]. Garg et al. proposed the SMICloud framework for comparing and ranking cloud services by defining a set of attributes for the comparison of mainly IaaS cloud offerings [8], while Hussain et al. proposed a multi-criteria decision-making methodology for the selection of cloud services [9]. To rank services, they matched the user requirements against each service offering for each criterion. Wang et al. proposed a cloud model for the selection of Web services [10]. This model relies on computing what the authors called *QoS* uncertainty and identifies the most appropriate Web services using mixed integer programming. In 2012, Anita Mohebi proposed a vector-based ranking model to enhance the discovery process of services [11]. Rehman et al. proposed a cloud service selection framework that relies on *QoS* history [12]. A heuristic service selection method, called "Bee Algorithm", was proposed by Karry et al., which helped to optimise the discovery and selection of a service that meets customer requirements [13]. In this paper, we adopt the Service Oriented Architecture to build a Data Warehouse of quality of services. It enables application of an advanced level of analysis and optimisation in discovering cloud services.

3. Cloud service selection structure

Cloud computing can be defined as a model for enabling convenient, on-demand network access to a shared pool of resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. A cloud environment is characterised by system level, Cloud Broker level and user middle-ware level.

The user Middle-ware level includes the software frameworks such as Web 2.0 Interfaces and provides the programming environments and composition tools that ease the creation, deployment and execution of applications in Clouds.

The system level is composed of thousands of servers, each with its own service terms management systems, operating platforms and security levels. These servers are transparently managed by the higher level virtualisation [14] services and toolkits that allow sharing their capacity among virtual instances of servers.

The Cloud Broker level implements the platform level services that provide runtime environment enabling Cloud computing capabilities to build cloud services. The Cloud Service Broker performs several management operations to deliver personalised services to consumers. These operations are: security and policy management, access and identity management, SLA management, provision and integration management. The security and policy manager is responsible for managing different kinds of policies such as authorisation policies and QoS-aware selection policies of service providers. The access and identity manager is responsible for the accessing services and respect the identity rules of services. The SLA Manager directs the concession process between a consumer and a selected

SaaS provider in order to reach an agreement as to the service terms and conditions. The provision and integration manager is responsible for implementing different policies for the selection of suitable SaaS providers, based on the consumer's QoS requirements and the SaaS providers' QoS offerings. The back-end database stores sustain information about service policies, consumer profiles, SLAs, Registry and dynamic QoS information. Cloud broker layer works to identify the most appropriate cloud resource and maps the requirements of application to customer profile. Its job can also be dynamic by automatically routing data, applications and infrastructure needs based on some *QoS* criteria like availability, reliability, latency, price, etc. On the Broker side, service properties are stored as a combination of functional and non-functional properties. The functional properties relate to the external behaviour of a service such as: service inputs and outputs, service type and the information required for connecting to the service. However, the non-functional properties are summarised by the *QoS*.

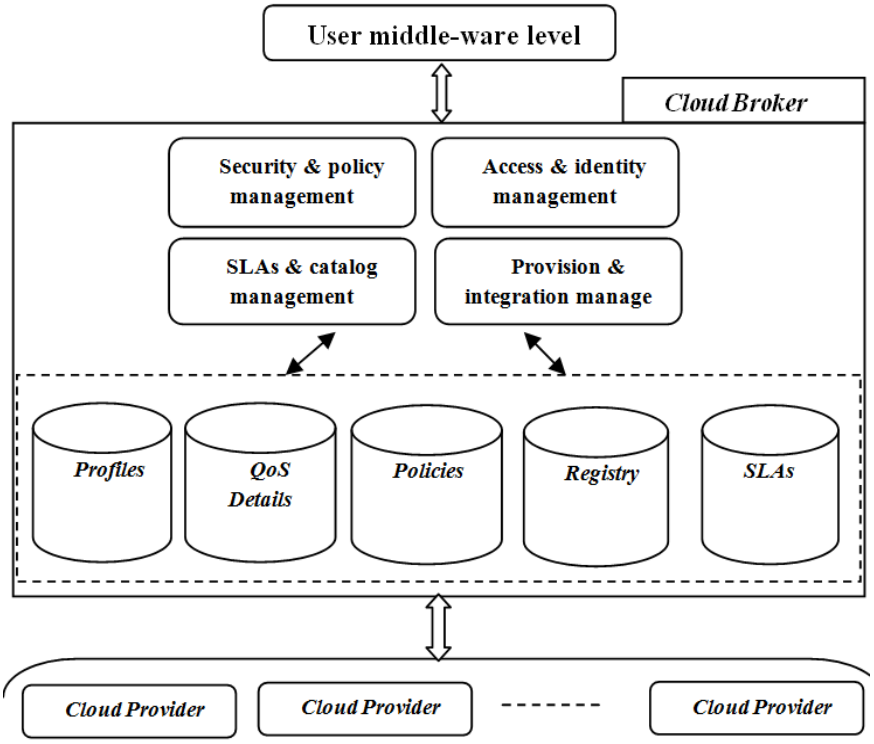

Figure 1: Main Layers of cloud service infrastructure.

By dynamically provisioning resources, Cloud broker enables cloud computing infrastructure to meet arbitrary varying resource and service requirements of cloud customer applications. However, there are still imperfections regarding service matching based on available services and customer profile requirements. The services selection problem is identified by an inaccurate *QoS* dependency and the utility of the imprecise domain of results suggested by *QoS* broker. As in [19], services are ranked into many levels such as Poor, fair, Good,

Excellent or Bronze, Platinum, Silver and Gold, based on Web Service Relevancy Function (WsRF), which is measured based on the weighted mean value of the QoS parameters.

The *QoS* broker orchestrates resources at the end-points, coordinating resource management across layer boundaries. Based on the available technology, Service consumer is still incapable of a real analysis of the *QoS* based on the internal structure of complex service. Today's service selection solutions do not focus on *QoS* support from the service requester view point, but they depend on service provider interpretation. Indeed, the current form of service selection is provider driven [15]. A consumer may interact with a composite service without knowing much about the qualities of the services that underlie it [16].

In order to improve the selection of a complex service, we propose to analyse the *QoS* of every sub-service, which shares in the composition of that service, using a *QoS* Data Warehouse (*QoSDW*).

4. QoSDW model

Nowadays, the cloud is full of a large number of cloud services. Some of these services are similar in goal and quality. Therefore, it is difficult to select best service depending on the traditional *QoS* methods. In order to improve the service selection process, we propose a *QoS* Data Warehouse (*QoSDW)* model. The *QoSDW* model (described in Figure 2) supports a better analysis of services before taking a selection decision. The *QoSDW* model extracts details about services stored in the service provider, and gives the service's consumer the ability to discover the hidden facts about the properties of these services.

4.1 QoSDW components

This section describes a model for the selection of a cloud service that can fulfil the service consumer request. In addition to the main cloud framework elements discussed in the previous section, the proposed QoSDW model adds a group of other components such as: *QoSDW Parser, Schema Manager, Graph Manager, QoSDW Analyser, QoSDW Cube, Analysis Interface, Service Tree Manager and Report Manager*.

QoSDW Parser: *QoSDW Parser* is simply a *service business process* parser. Based on the parsers outputs and the *QoS* at service provider, *QoSDW schema and QoSDW graph* are extracted and transported into the cloud broker to be stored in a specific database. Regarding the database tables, each row entry collects details about service activities. It provides information about the current state name, current state properties (as *My Role*, *Partner Role*), *PartnerLink*, name of the operation being invoked, condition of a looping structural activity, current state number and next possible state numbers.

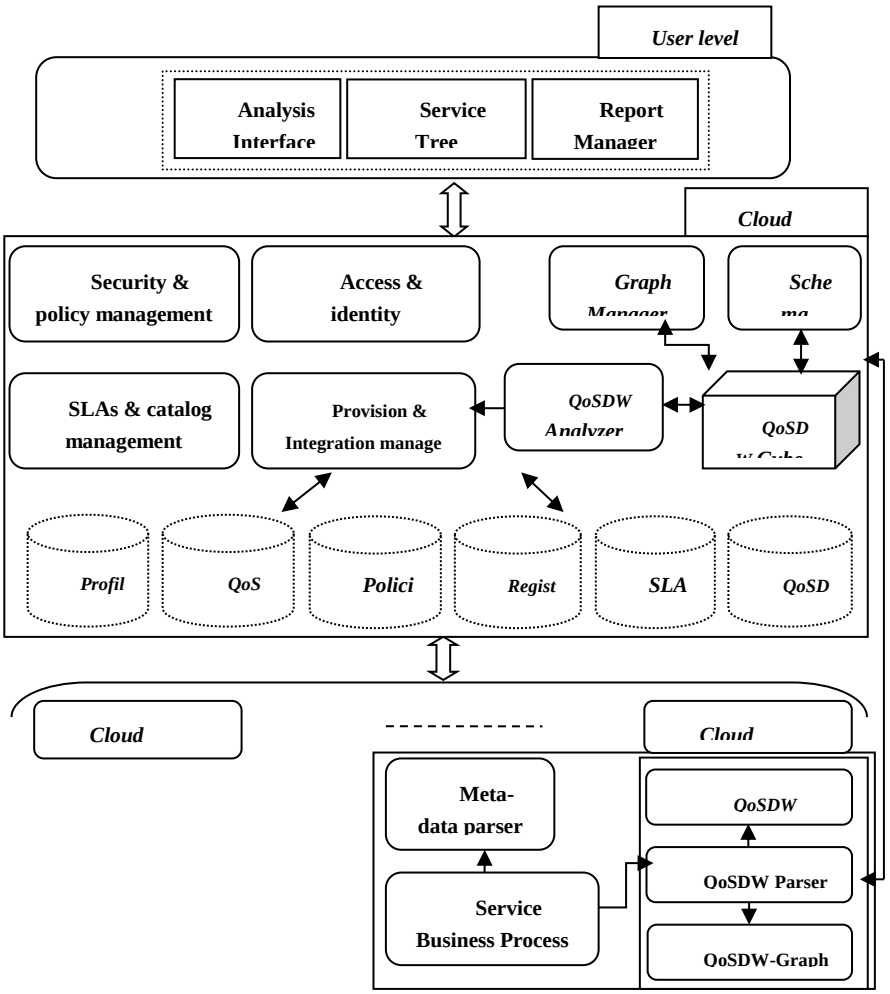

Figure 2: QoSDW model components.

Schema Manager is responsible for managing the *QoSDW* schemas. The *QoSDW Schema* is a star schema which is composed of a set of organised tables, and which has a main fact table and set dimensional tables. *QoSDW Schema* consists of 22 dimensional tables as follows: *Quality, Availability, ResponseTime, Documentation, BestPractice, Throughput, Latency, Successability, Reliability, Compliance, property, ServiceType, ServiceName, ExpiryDate, CreationDate, ServiceFlow, Loop, Sequence, AndSplit, XorSplit, AndJoin and XorJoin table.*

Graph Manager ensures transforming the output of parsing the service business into a directed acyclic graph. Also, it converts the obtained graph into a service tree. For example, figure 3 shows how *Steam Boat* service process diagram is transformed into a service tree. The service tree inserts a semantic layer into the service selection process.

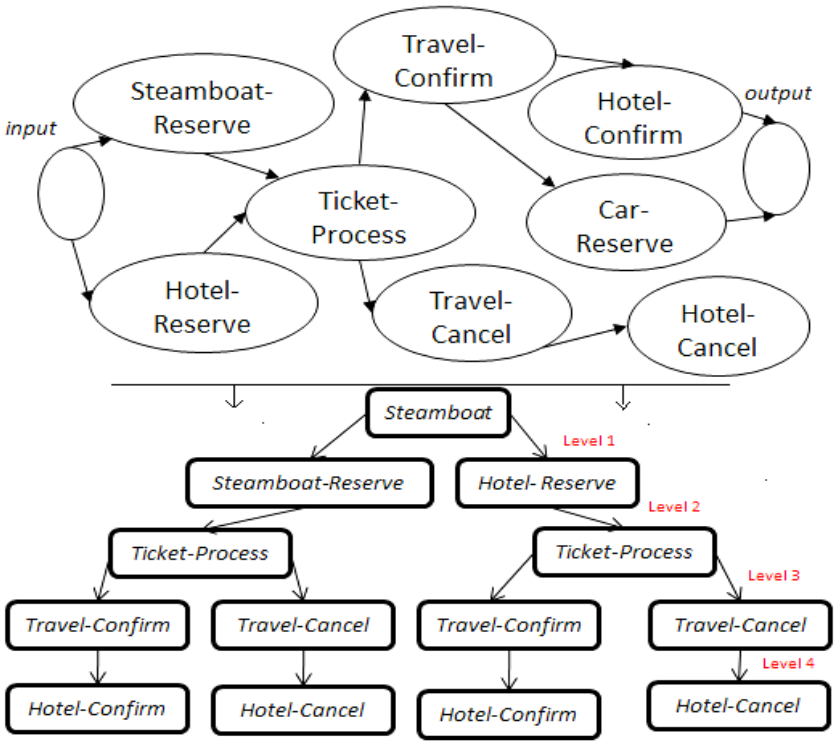

Figure 3: Transforming SteamBoat service business flow into a tree of sub-services.

QoSDW Cube is a Data Warehouse of quality and structure of both a service and its sub-services. It is accessed as a cloud service and supports users by details about the quality and flow of service through a special Analyser. It maps the idea of the multidimensional data model to service selection model, through which it gives the service's user the ability to apply a multidimensional query on the discovered set of services.

QoSDW Analyser works like an analysis tool. It monitors *QoS* changes and prepares analytical reports about *QoS* information stored in the *QoSDW Cube*. It gives the service consumer the right to query the *QoSDW Cube* through its interface.

Analysis Interface is a user interface application utilised to select cloud services (*SaaS*). It consists of a statistical form which allows a user to deal easily with large statistical data, through slice, dice, Drill Down/Up and Roll-up the statistical results. It communicates with the *QoSDW* Analyser and allows users to connect to the *QoS* data warehouse, at the cloud broker, and apply queries. When a service is selected, the *Selection Interface* connects the user to the required service via the SOAP/HTTP protocol.

Service Tree Manager supports a visual representation of the service's tree. It communicates with the *Graph Manager* indirectly through the *QoSDW Analyser*. Based on the service graph, the Analyser supplies the user by the service tree.

***Report Manager*:** Sometimes the service's consumer needs ready reports that support their analysis. *Report Manager* allows requesting two types of reports: the primary report gives analysis results about the quality of first level sub-services, and the advanced report supports a deep service tree analysis to detect a weak quality subservice (or fatal sub-service). Both reports are requested from the *QoSDW Analyser*.

4.2 Formal definitions of *QoSDW* model

The main objective of a *QoSDW* model is to provide efficient analytical reporting on the quality of service. In order to qualify a service, the *QoSDW* depends on analysing the quality of its sub-services. *QoSDW* depends on the service business process to specify the structure of subservices.

Definition 1: A *service business process* is a tuple $K = (A, E, C, L)$ where:
A is a set of activities,
E is a set of events,
C is a set of conditions and L is a set of control links.

Let f: $A \rightarrow B$ be a function that assigns activities to types, where activities are extracted from the set of activity *A*= {*sequence, flow, pick, switch, while, scope, invoke, receive, reply, wait, assign, empty, throw, compensate, exit*}. Let *I* be a set of service information, where *I* = {*service name, service type, service creation date, service expiry date*).

Let *g: P⟶I* be a function which assigns service information to properties.

QoSDW utilises an On-line Analytical Processing (*OLAP*) approach and performs analysis in conjunction with the operational database on a constant basis. The basic concept of *OLAP* model is to map the initial database schema to a multidimensional model. The *QoSDW schema* is structured as star (or snowflake) schemas.

Definition 2: A *QoSDW* schema is a tuple $S = (Q, P, B)$ where:
Q is a set of *QoS*, such that Q = {*Response time, Availability, Throughput, Successability, Reliability, Compliance, Best Practice, Latency, Documentation*}.

- P is a set of service properties, such that *P*= {*ServiceType, ServiceName, ExpiryDate, CreationDate*}.
- *B* is a set of activity type, where *B*= {*Loop, Sequence, AndSplit, XorSplit, AndJoin, XorJoin*}.
- Let *h* be a function which assigns the values of *QoS* to elements of set Q.

The *QoSDW* graph adds a type of semantic knowledge when analysing the quality of sub-services and covers indirectly the hidden service business process vague.

Definition 3: A *QoSDW graph* is a tuple $G = (Ni, Nf, N, F)$, where:

Ni is the node of the input, *Nf* is the node of output, *N* is the set of names of sub-services and *F* is the set of service integration models. *F* = {*Sequence, ANDSplit, XORSplit, loop, ANDJoin, XORJoin*}.

Let m: $B \rightarrow F$ be a function that maps service activities to integration models.

The operations which are applied in the analysis phase of the *QoSDW* model are summarised by: Composition, Pairing, Projection and Restriction.

Composition takes as input two functions f and g, such that $range\ (f) \subset def\ (g)$, and returns a function $g \circ f$: $def\ (f) \rightarrow range\ (g)$, defined by: $(g \circ f)\ (x) = g\ (f\ (x))$ for all x in $def\ (f)$.

Pairing takes as input two functions f and g, such that $def\ (f) = def\ (g)$, and returns a function $f \wedge g$: $def\ (f) \rightarrow range\ (f)\ X\ range\ (g)$, defined by:
$(f \wedge g)\ (x) = < f(x), g(x) >$, for all x in $def\ (f)$.

Projection is the usual projection function over a Cartesian product. Take function f: $X \rightarrow Y$ and g: $X \rightarrow Z$ with common domain X, and let πy and πz denote the projection functions over $Y\ x\ Z$:
$f = \pi y \circ (f \wedge g)\ and\ g = \pi z \circ (f \wedge g)$.

Restriction takes as argument a function f: $X \rightarrow Y$ and a set D, such that $D \subset X$, and returns a function f/D: $D \rightarrow Y$, defined by: $(f/D)\ (x) = f\ (x)$, for all x in D.

4.3 Building QoSDW Schema

The base of *QoSDW* schema is a finite labelled diagram whose nodes and arrows satisfy the following conditions: there is only one root, at least one path from the root to every other node and all arrow labels are distinct. Our goal from the obtained *QoSDW* schema is to have an organised store of service qualities, properties and structure in which multidimensional queries can be applied.

The proposed *QoSDW Schema* consists of the following tables:
Fact table: *Fact (service_id*, URI_type);*
Table of dimension Quality: *Quality (Quality_id*, Quality_value, foreign_ service_id);*
Tables of dimension Quality attributes:
Availability: *Availability (avail_id*, avail_value, foreign_Quality_id);*
Response time: *ResponseTime (response _id*, response_time_value, foreign_Quality_id);*
Documentation: *Documentation (Doc _id*, Documentation _value, foreign_Quality_id);*
BestPractice: *BestPractice (practice_id*, practice _value, foreign_Quality_id);*
Throughput: *Throughput (throughput_id*, throughput_value, foreign_Quality_id);*
Latency: *Latency (Latency_id*, Latency _value, foreign_Quality_id);*
Successability: *Successability (Successability_id*, Successability _value, foreign_Quality_id);*
Reliability: *Reliability (Reliability_id*, Reliability_value, foreign_Quality_id);*
Compliance: *Compliance (Compliance_id*, Compliance_value, foreign_Quality_id);*
Table of dimension property: *property (property_id*, property_value, foreign_ service_id);*
Tables of dimension property attribute:
Type: *ServiceType (ser_type_id*, type_value, foreign_property_id) /value: service or sub-service*

Name: *ServiceName (ser_name_id*, ser_value, foreign_property_id);*
ExpiryDate: *ExpiryDate (ExpiryDate_id*, ExpiryDate _value, foreign_property_id);*
CreationDate: *CreationDate (CreationDate_id*, CreationDate_value, foreign_property_id);*
Table of dimension flow: *ServiceFlow (flow_id*, service_flow_value, foreign_ service_id);*
Tables of dimensional flow attribute:
Loop: *Loop (loop_id*, input_service, output_service, service_stage, foreign_ flow_id) / stages: start node, normal node or end node.*
Sequence: Sequence (sequence_id, input_service, output_service, service_stage, foreign_ flow_id);*
AndSplit: *AndSplit (AndSplit_id*, input_service, output_service, service_stage, foreign_ flow_id);*
XorSplit: *XorSplit (XorSplit_id*, input_service, output_service, service_stage, foreign_ flow_id);*
AndJoin: *AndJoin (AndJoin_id*, input_service, output_service, service_stage, foreign_ flow_id);*
XorJoin: *XorJoin (XorJoin_id*, input_service, output_service, service_stage, foreign_ flow_id);*

4.4 Service selection based on *QoSDW*

Based on the *QoSDW* schema, the *QoS* Data Warehouse is built. Similar to the traditional discovery method, the service consumer requests a service and the service registry replies by a set of related service. If the *QoS* is not helpful to select the best service, the service consumer requests an OLAP analysis report about the quality of the discovered set of services. The *QoSDW* model consists of a special *QoSDW Analyser* which supports two types of reports about *QoS*. The first type is a preliminary report which provides information about the quality of first level sub-services. Figure 4 shows a visual representation given by the *QoSDW Analyser* about *QoS* of sub-services.

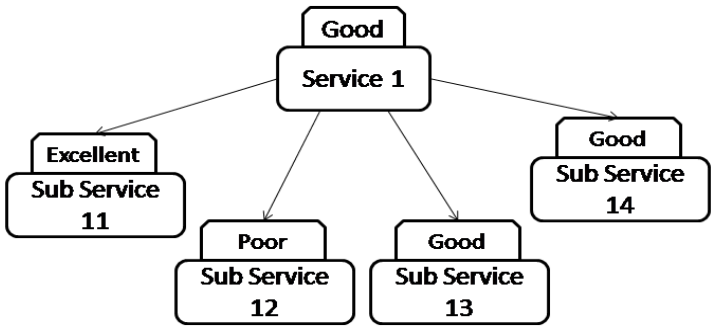

Figure 4: Visual representation of the initial report.

Sometimes the results of the initial report are not beneficial in designing a new composite service of better quality. Thus, the advanced *QoS* report is demanded by the service designer. As regards building the required report, the *QoSDW* Analyser applies some queries on Data warehouse, which results in a service's tree (figure 3 in section 4.1). Then, the Analyser utilises a tree search algorithm to detect fatal sub-services (as shown in **Algorithm 1** below).

Algorithm 1: Detection of infected services
Input: A tree graph,

A set of start nodes,
Boolean procedure *undercritical (n),* that tests if the *QoS* of a tree node '*n* 'is under critical values.
Frontier: = {<*s*>: s is a start node};
Fatalist: = {<*r*>: r is a sub-service of weak *QoS*};
Filter (x): a procedure that removes the node duplications from array list *x.*
While *frontier* is not empty:
Select and **remove** path *<n0; ...; nk>* from *the frontier*;
If *undercritical (nk)*
Add node *nk* to the *FatalList*
For every neighbor *n* of *nk*
Add <n0; ...; nk> to *frontier*;
End while
Filter (FatalList)
Output: Return the filtered set of *FatalList*

The fatal service is a weak quality sub-service (its *QoS* is below the critical values), which causes weakness in the quality of the parent service. The existence of fatal sub service is sufficient for the service consumers not to select the parent service, because they pay their money for utilising an infected service. Thus, the *QoSDW* models added a new quality attribute in the selection process – the number of fatal sub-services. Indeed, if there is a group of discovered services of equal *QoS* level, the service which has the least number of infected sub-services must be selected. In terms of infected services detection, the service designer is capable of rebuilding improved versions of these services, free of fatal sub-services.

Also, if the *QoSDW Analyser* reports are not helpful in selecting the best service, service consumers can apply their own queries on the Data Warehouse as described in the next section.

5. *QoSDW* model benefits

Because the *QoS* of sub-services are now accessible through *OLAP* queries, some hidden facts, about *QoS*, can be discovered. In the previous approaches, the discovered services are only qualified with no information about its internal sub services. Based on the *QoSDW* model, the weak sub-services which lead to bad parent service qualities can be studied and treated, in each part of the complex service, before the selection process. Compared with the traditional selection process, *QoSDW* is more advanced. Both service consumers and service providers can benefit from the *QoSDW* model. Service consumers are capable of applying a deep analysis concerning the service component before selection, using *QoSDW* Analyser reports and *OLAP* queries. The *QoSDW* is also beneficial for cloud service companies, because service designers are capable of analysing the fatal sub-services that cause a weak service and redesigning a similar service with better quality.

In order to show the advantages of the *QoSDW model* from the queries prospective, we present an *OLAP* example, which is simulated as graph and algebraic queries. Consider a schema *S*, an *OLAP* Query over *S* is a triple *Q = (x, y, z)*, satisfying the following conditions:

- *x* and *y* are path expressions such that the source (*x*) = source (*y*) = *root object.*
- *z* is an operation over the target of *y.*

The expression *x* will be referred to as the classifier of *Q* and the expression *v* as the measure of *Q*.

Figure 5 shows the *QoSDW* schema as an acyclic graph, such that the root is the object of an application, while the remaining nodes model the attributes of the object. Through queries, some functions (such as *av*, *rt* and *dc*) are used when invoking object. Concerning the online *QoS* analysis through *QoSDW*, *OLAP* queries are prepared using paths starting at the root object (Fact).

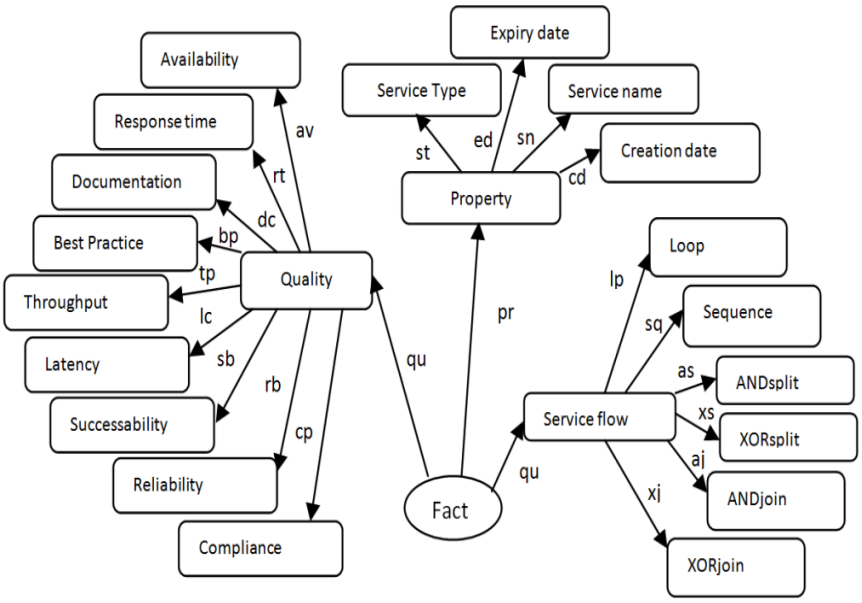

Figure 5: The proposed QoSDW schema.

Through OLAP, service consumers can apply an advanced query such as:

- *Q1:* Ask for sub-services which utilise *XORjoin* integration when invoking other services and their *Response Time* greater than 80 (*ms*) sorted by name of service.

  Let us divide the query *Q1* (following figures 7 and 8)*:*
  - Ask for sub-services: ***pr*** *o* ***st.value*** == *'**sub-service**'*
  - Which utilises *XORjoin* integration when invoking other services: ***qu*** *o* ***xj***
  - Their *Response Time* greater than 80 (ms): ***qu*** *o* ***rt. value*** > ***80***
  - Sorted by name of service: ***(pr*** *o* ***sn)^ (pr*** *o* ***st.value*** *==**'service**')*
- *Q1 = < **(pr** o **sn)**^ **(pr** o **st.value**==**'service'**), **((pr** o **st.value**== **'sub-service'**) ^ **(qu** o **xj)** ^ **(qu** o **rt. value** > **80))**, **sum**>*

Considering the steamboat service (section 3), the answer of the query *Q1* is: *Ticket_Process*. For more details about the algebraic base of *OLAP* refer to [17].

6. Simulation and results

In order to facilitate understanding the model, we discuss in this part an example of service selection and simulation of selecting service based on the fatal service property.

6.1 Service selection example based on *QoS*

QoS consists of a group of properties, but for the purpose of simplicity, our example examines just two of these properties (the service *cost* and service *response* time). For complex services, six basic integration models (Figure 6) are considered and they are compatible with the business process of a service.

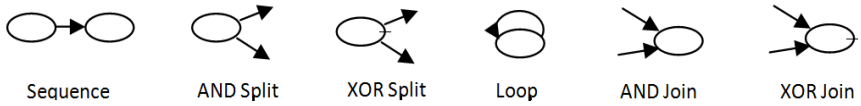

Figure 6: Basic integration models for complex cloud services.

Suppose a client is looking for a service to make a steamboat travel reservation (Figure 7). First, s/he needs to make a steamboat and hotel reservation and then apply for a visa ticket. If the visa is approved, he can buy the steamboat ticket and confirm the hotel reservation, otherwise, he will have to cancel both reservations. Also, if the visa is approved, he needs to make a car reservation. To complete its job, this complex service (*Steamboat Service*) invokes other services such as: *Steamboat-Reserve, Hotel-Reserve, Ticket-Process, Travel-Confirm, Hotel-Confirm, Car-Reserve, Travel-Cancel and Hotel-Cancel.*

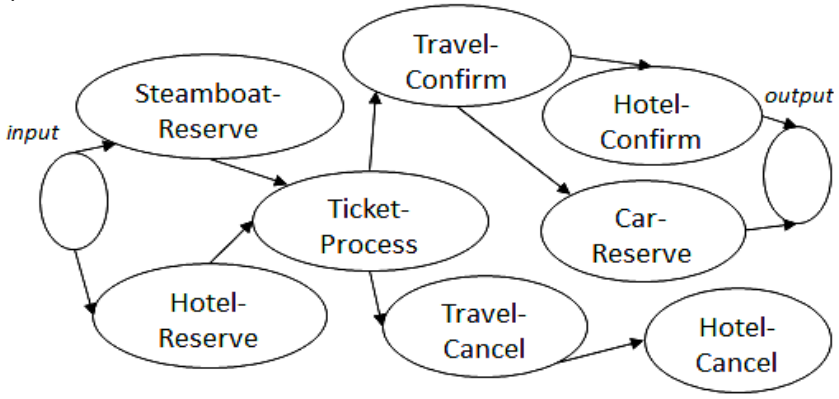

Figure 7: Example of interior composition of the Steamboat.

A cloud service designer wants to compose a service that serves all types of online travel reservation. One of the used services in this composition is *Steamboat* service, which serves an online boat reservation. However, several cloud providers support such type of boat service (Table 1). In this case, the service consumer depends on the *QoS* (or *QoSBroker*) to select the best service.

Table 1: QoS (cost & response time) of three services

Service Name	Cost ($)	Response Time (ms)	Class

Steamboat	0.088	106	Good
Steamboat-Travel	0.12	130	Good
Manage-Steamboat	0.18	183	Poor

Based on the service properties mentioned in Table 1, the service consumer will choose the Steamboat service because it is evaluated as best service (Class: *Good*) by *QoSBroker*. But how was the *QoS* calculated?

The *QoS* calculations are based on Cardoso's *QoS* formulas [18]. This provides insights into computational details about the estimation of some *QoS* in the service selection processes such as: response time and cost.

Response Time (T): Task response time refers to the time taken by a request to be processed by a task. For sequential tasks, two tasks $\boldsymbol{ti}$ and $\boldsymbol{tj}$, which are in sequence, may be reduced to a single task $\boldsymbol{tnew}$, so that: $\boldsymbol{T(tnew) = T(ti) + T(tj)}$. In a parallel system, multiple tasks $(\boldsymbol{ti, tj, \dots, tn})$ are reduced to their maximum according to the formula: $\boldsymbol{T(tnew) = Max_{i\varepsilon(0,1,\dots,n)}\{T(ti)\}}$.

Cost (C): Task cost is taken to be cost incurred by the service provider when a task is executed. For sequential tasks $\boldsymbol{ti}$ and$\boldsymbol{tj}$, the new task is calculated according to the following formula: $\boldsymbol{C(tnew) = C(ti) + C(tj)}$. While in parallel tasks $\boldsymbol{ti}$ and$\boldsymbol{tj}$, the cost is obtained using this formula $\boldsymbol{C(tnew) = \sum_{1\le i\le n} C(ti)}$.

Table 2: QoS (cost & response time) of Steamboat sub-services

Service Name	Cost ($)	Response	Class
Steamboat-Reserve	0.021	90	Poor
Hotel- Reserve	0.019	112	Good
Ticket-Process	0.015	125	Good
Travel-Confirm	0.009	122	Excellent
Hotel-Confirm	0.007	99	Good
Car-Reserve	0.012	84	Good
Travel-Cancel	0.003	114	Excellent
Hotel-Cancel	0.002	119	Excellent

The evaluation of a service depends on its entire structure and the quality of the other sub-services invoked to compose such service. In our example, the properties of the Steamboat service are based on the properties of its sub-services such as *Steamboat-Reserve and others* (as shown in Table 2).

6.2 *QoSDW* simulation

This section explains the results of the *QoS* data warehouse simulation based on the proposed *QoSDW* model. To implement this simulation, we use *SQL server 2012, Eclipse indigo, Apache Tomcat Server (v.7), Microsoft visual studio 2012 and windows Azure*.

Figure 8: Difference in *QoS* values among three cloud services (*SteamBoat*, *TravelViaSteamboat* and *ManageSteamboat*).

This simulation discusses a steam boat service selection, which was described in section 5.1 above, based on the fatal service property that resulted from the application of *QoSDW* model.

The service consumer requests a steam boat service and the *cloud broker* reply by a list of three discovered services: *SteamBoat*, *TravelViaSteamboat* and *ManageSteamboat* (see table 1 in section 5.1). In the traditional service selection process, the *QoSBroker* calculates the *QoS* of the discovered set service, based on the service provider measures. Figure 8 shows the variation of *QoS (ResponseTime, Availability, Throughput, Successability, Reliability, Compliance, BestPractice, Latency, Documentation)* of a three services (*steamboat, TravelViaSteamboat and ManageSteamboat*) given by the *QoSBroker*.

As a result, the *QoSBroker* marks services *steamboat* and *TravelViaSteamboat* as ***Good*** services. However, it marks *ManageSteamboat* as a ***Poor*** service. Based on the *QoS* primary results, the weak service is excluded, while the *QoS* of the two other good services is studied, in order to select the best of them. Indeed, the traditional selection method shows that the *QoS* of the two good services is approximately equal. Thus, it is difficult to decide which service is better.

Based on the *QoSDW* model, the service consumer is capable of requesting more analysis details about the discovered services. Indeed, the *QoSDW* Analyser supports the consumer by a preliminary report, which analyses the *QoS* of first sub-service level of the discovered set of services.

Figures 9 and 10 show respectively the variation of quality of sub-services of both *steamboat* and *TravelViaSteamboat* services.

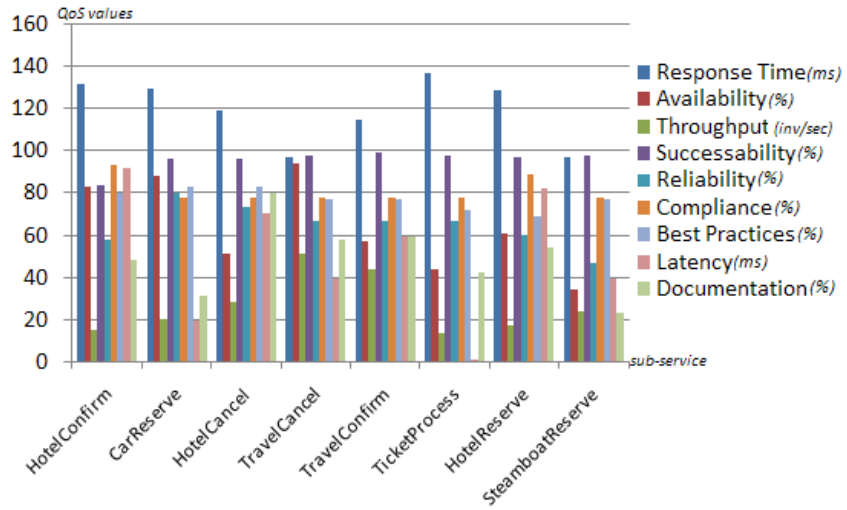

Figure 9: *QoS* of the sub-services of *steamboat* service.

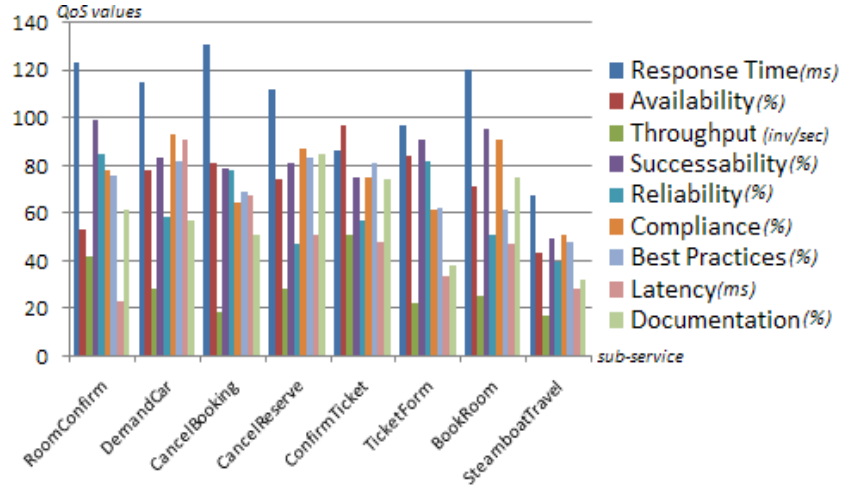

Figure 10: *QoS* of the sub-services of *TraveVialSteamboat* service.

Sometimes, the first report is not beneficial in selecting the best service, so a more advanced report is requested from the *QoSDW Analyser*. In our example, the *QoSDW AnalySer*, in its second report, detects a fatal sub-service in the tree of *TravelViaSteamboat* service (as shown in figure 10, the *SteamBoatTravel* service suffers from weak qualities in which: Response Time= 65 ms, Throughput= 17 invokes per second, Latency= 34 ms, Availability= 53 %, Reliability= 40 % and Best Practice= 43 %). The final report concludes that the *SteamBoat* service is the best service to be selected. However, if sometimes results are not convincing, a service consumer can query the *QoSDW Analyser*, using *OLAP* queries, and build a much advanced cloud service analysis (as discussed in section 4).

7. Conclusion

Clouds aim to power the next generation data centres by exposing them as a network of virtual services. Cloud users are able to access and deploy applications from anywhere in the world on demand at competitive costs depending on users' *QoS* requirements. However, there are still many problems covering the selection of cloud services and even if this level is introduced later, it needs a long time and much effort to change the whole web structure. This research introduces a *QoSDW* model that improves the selection process of cloud services. As a summary of the *QoSDW* model, the service business flow is mapped into a star database schema on the cloud provider side. At the cloud broker, an *OLAP* Cube is built from the stored schemas. Depending on the obtained Cube, advanced analysis steps are applied to *QoS* using a *QoSDW Analyser*. Indeed, *QoSDW Analyser* returns reports about *QoS* that improves service selection and helps in reaching a better dynamic service composition. As a future work, our goal is to achieve a logic layer for cloud services, which support service autonomy in case of selection and composition procedures.

Acknowledgement

This work is supported by the Department of Computer Science at the University of Quebec at Chicoutimi, the Ecole Doctorale des Sciences et des Technologies at the Lebanese University and the AZM association.

References

1. Buyya R., Yeo C, Venugopal S, (2008), Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. 2008 10th IEEE Int. Conf. High Perform. Comput. Commun
2. Wu J, Yang F, (2007), QoS Prediction for Composite Web Services. System. 86–94
3. Dan A, Davis D, Kearney R, Keller A, King R, Kuebler D, Ludwig H, Polan M, Spreitzer M, Youssef A, (2004), Web services on demand: WSLA-driven automated management
4. Yu T, Lin K, (2004), Service selection algorithms for Web services with end-to-end QoS constraints. Proceedings. IEEE Int. Conf. e-Commerce Technol. 2004. CEC 2004
5. Keskes N, Lehireche A, Rahmoun A, (2009), Web Services Selection Based on Context Ontology and Quality of Services, king faisal Univ., Saudi Arabia.
6. Raj R, Sasipraba T, (2010), Web Service Selection Based on QoS Constraints. Sathyabama Univ., Chennai, India
7. Squicciarini A, Carminati B, Karumanchi S, (2011) A Privacy-Preserving Approach for Web Service Selection and Provisioning. 2011 IEEE Int. Conf. Web Serv. 33-40
8. Garg S, Versteeg S, Buyya R, (2011), SMICloud: A Framework for Comparing and Ranking Cloud Services. 2011 Fourth IEEE Int. Conf. Util. Cloud Comput. 210–218
9. Rehman Z, Hussain O, Hussain F, (2012), Iaas Cloud Selection using MCDM Methods. 2012 IEEE Ninth Int. Conf. E-bus. Eng. 246–251
10. Wang H, Lee C, Ho T, (2007), Combining subjective and objective QoS factors for personalized web service selection. Expert Syst. Appl. 32, 571–584

11. Anita M, (2012), An efficient QoS-Based Ranking Model for Web Service Selection with Consideration of User's Requirement. Thesis and dissertations, Ryerson University, Ontario, Canada
12. Nallur V, Bahsoon R, (2012), A Decentralized Self-Adaptation Mechanism for Service-Based Applications in the Cloud
13. Karray A, Teyeb R, Ben Jemaa M, (2013), A heuristic approach for web-service discovery and selection, International Journal of Computer Science & Information Technology (IJCSIT), Vol 5, Issue 2
14. Smith J, Nair R, (2005), Virtual machines: versatile platforms for systems and processes
15. Liu W, (2005), Trustworthy service selection and composition - reducing the entropy of service-oriented Web. INDIN '05. 2005 3rd IEEE Int. Conf. Ind. Informatics
16. Yu Q, and Bouguettaya A, (2010), Guest Editorial: Special Section on Query Models and Efficient Selection of Web Services, IEEE Transactions on Services Computing, Vol. 3, No. 3
17. Spyratos N, (2006), A Functional Model for Data Analysis, Lecture Notes in Computer Science, Publisher Springer Berlin Heidelberg, Volume 4027, pp 51-64
18. Cardoso, J (2002), Quality of Service and Semantic Composition of Workflows
19. Al-Masri E, Mahmoud Q, (2007), Discovering the best web service. 65. p. 1257

No index entries found.