2014 IEEE 28th International Conference on Advanced Information Networking and Applications

A Cloud Trust Evaluation System using Hierarchical Fuzzy Inference System for Service Selection

Chenhao Qu and Rajkumar Buyya
The Cloud Computing and Distributed Systems (CLOUDS) Lab
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: cqu@student.unimelb.edu.au, rbuyya@unimelb.edu.au

***Abstract*—Cloud computing is an utility computing paradigm that allows users to flexibly acquire virtualized computing resources in a pay-as-you-go model. To realize the benefits of using cloud, users need to first select the suitable cloud services that can satisfy their applications' functional and non-functional requirements. However, this is a difficult task due to large number of available services, users' unclear requirements, and performance variations in cloud. In this paper, we propose a system that evaluates trust of clouds according to users' fuzzy Quality of Service (QoS) requirements and services' dynamic performances to facilitate service selection. We demonstrate the effectiveness and efficiency of our system through simulations and case studies.**

***Keywords*—Cloud Computing, Trust, Service Selection, QoS, IaaS, Hierarchical Fuzzy Inference System**

I. Introduction

Cloud is a highly agile and flexible utility computing paradigm that allows users to acquire virtualized computing resources in a pay-as-you go model [1]. Its merits, including no upfront investment, just-in-time resource provisioning, and fast deployment, is attracting more and more organizations to migrate their existing applications and develop new systems on cloud. Among all cloud service models, Infrastructure-as-a-service (IaaS) provides the most flexible service for users trying to deploy their own applications. To realize the benefits of using IaaS cloud, users need to ensure the trading service providers can fully satisfy their applications' functional and non-functional requirements. However, there exists plenty of IaaS cloud providers offering similar services with different pricings and performances, which creates difficulty for users to find the most suitable service. Therefore, it is essential to develop automatic systems to dynamically identify satisfactory IaaS services according to different application requirements.

Observed by experiments [2], VM performances within the same cloud are not static. Schad et al. [2] pointed out that one influencing cause is the performance heterogeneity of hardware in the underlying physical infrastructure, e.g., different types of CPU, memory, and disk used in the physical hosts. Besides that, the interference from colocated VMs also affect performance greatly [3]. Thus, the surge of VM requests in peak hours often causes performance degradations. Some other providers, e.g., eApps[1], dynamically allocate CPU to VMs on the same host according to their priorities, which makes VM performances even more unstable. Since the aggregated potential variability is considerable [2], it is necessary to take it into account at the selection phase to improve the cost-efficiency of using cloud.

Trust management systems have been utilized to help users to select satisfactory and trustworthy services in many scenarios [4]. Applied in cloud context, they should be able to capture personalized requirements and preferences to provide customized services, as cloud users often have diverse expectations for services when deploying different applications.

Many state-of-art cloud service selection systems [5][6][7] [8][9][10][11] need users to submit static weights to model preferences for attributes, which requires expert knowledge and is time-consuming. To let users smoothly adopt cloud, a more nature way is to let them express their vague preferences in linguistic phrases. Besides, sometimes it is also difficult for users to determine QoS requirements in accurate values, e.g., users need to conduct complex tests to determine the actual CPU power needed to process 50 transactions in parallel. Similarly, by using approximate linguistic descriptors, users can define requirements more easily and quickly.

In this paper, we propose a new personalized trust evaluation system to support cloud service selection. In particular, we measure trust of clouds as their satisfactory degree to specific user requirements based on their past performances. We employ membership functions and fuzzy hedges to capture users' subjective requirements and preferences for different QoS attributes and then use a hierarchical fuzzy inference system to derive trust levels. By analyzing past benchmark results, our system can identify services that are likely to meet all QoS requirements in the whole application life cycle. Through simulations and case studies, we demonstrate both the effectiveness and efficiency of our approach.

The remainder of this paper is organized as follows. We introduce some background about trust management and fuzzy inference in next section. Then we discuss related works and their limitations. After that, we present our trust evaluation system, followed by the performance evaluation. In section VI, we illustrate the usage of our system with two case studies. Finally, we conclude the paper and identify future directions.

II. Background

A. Trust Management System

Trust has different definitions under different contexts. In system world, trust is usually defined as the subjective belief that the system or component, the user intending to interact with, will behave as expected [4]. Such belief should be derived from strong evidence, such as past performances, peer recommendations, and certificates. Trust management systems are designed to aggregate trust from above evidence in real

[1]eApps http://www.eapps.com/

time and provide trust query services to parties in concern. They have played important roles in helping users to select the satisfactory services in many scenarios. However, since state-of-art trust management systems are usually developed for systems dedicated to certain missions, they are unsuitable for cloud where users run diverse applications for different purposes. Besides, they usually use user ratings as evidence, which is also invalid for clouds because users' diverse expectations for services are likely bias their ratings. For existing trust management systems and trust evaluation techniques, interested readers can find more details in the survey done by Jøsang et al. [4]. Different from previous systems, our trust evaluation system is able to provide customized service according to users' individual expectations.

B. *Hierarchical Fuzzy Inference System*

Fuzzy inference has been widely used to solve control and reasoning problems in uncertain environments due to its ability to handle inaccurate inputs. Figure 1 shows a typical fuzzy inference module. It has three major components:

1) **Inference Engine:** It defines the fuzzy logic operators and defuzzifier used in the inference process.
2) **Membership Functions:** A membership function defines to what degree the fuzzy element belongs to the corresponding fuzzy set. It maps crisp values to membership degrees between 0 and 1. In fuzzy inference system, each input and output variable has its own set of membership functions.
3) **Rulebase:** It is a set of "If-Then" rules that defines the inference model. The rule structure is like: "If *antecedent* Then *consequent*", where *antecedent* and *consequent* are fuzzy propositions connected by "AND" or "OR" operators.

The inference process usually involves five major steps:

1) **Fuzzification:** input crisp values into the membership functions to obtain corresponding membership degrees of each input variable regarding specific fuzzy set.
2) **Applying Fuzzy Operations:** obtain the membership degree of the *antecedent* using "AND" and "OR" operators.
3) **Implication:** obtain the fuzzy set of each rule using the defined implication operator.
4) **Aggregation:** aggregate output fuzzy sets of all rules using the defined aggregation operator.
5) **Defuzzification:** transform the aggregated fuzzy set into a crisp value using the defined defuzzification algorithm.

A hierarchical fuzzy inference system is connected by multiple atomic fuzzy inference modules with outputs of the low level modules serving as inputs to the upper level modules. It brings two major benefits compared with a non-hierarchical one. The first is that it can reduce the number of "If-Then" rules, which greatly simplifies the system design. The second benefit is that it enables the system to compute partial solutions when the task can be clearly partitioned or there are functional dependencies in the system. For further information of these systems, readers can refer to Torra's survey [12].

III. Related Work

Generally, state-of-art IaaS cloud selection approaches can be classified into two categories, namely, service ranking approaches, which aim to rank clouds according to their performances; and matchmaking approaches, which compare user requirements to actual service offerings.

For service ranking approaches, many teams [5][6][7][8][9][10][11] have investigated using Multi-criteria Decision Making (MCDM) methodologies to rank clouds. These systems depend on static weight assessment for preference modeling, which is time-consuming. Furthermore, except the work by Rehman et al. [10] and Wang et al. [11], none of them considered performance variations. Hamzeh et al. [13] developed a fuzzy based MCDM approach that uses linguistic descriptors to model preferences. Still, they didn't address the performance variation problem. Noor et al. [14] and Habib et al. [15] evaluated trust of clouds according to user feedbacks. But they ignored users' diverse requirements. Supriya et al. [16] also employed hierarchical fuzzy inference system to evaluate trust of providers. However, they required users to manually tune the inference system for each query regarding their expectations.

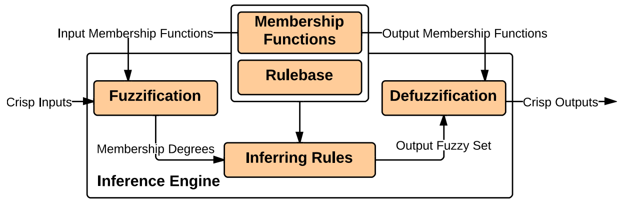


Fig. 1. A Typical Fuzzy Inference Module

The first important cloud service matchmaking system is proposed by Dastjerdi et al. [17]. They adopted description logic to match user's QoS goal and services' self-advertised Service Level Agreement (SLA) contracts. Sundareswaren et al. [18] proposed a time-efficient selection algorithm for cloud brokers based on B+ tree indexing. Redl et al. [19] employed Support Vector Machine algorithms to find the closest cloud service to user requirements. None of them considered performance variations, and only the approach by Sundareswaren et al. [18] takes user preferences into account.

Our system addresses the limitations in the previous works. It uses fuzzy linguistic descriptors and hedges to help users to quickly and easily define their requirements and preferences, and it considers performance variations in clouds when evaluating trust, which improves cost-efficiency for cloud users.

Apart from our work, fuzzy logic has been applied to address other selection problems. Nepal et al. [20] employed fuzzy set operations and fuzzy hedges for preference modeling in their web service selection system. Wang [21] used a fuzzy based MCDM algorithm to rank web services. Song et al. [22][23] utilized fuzzy inference system to evaluate trust in grid and P2P systems for resource selection.

IV. Proposed System

A. *System Architecture*

Figure 2 illustrates the general architecture of our proposed system. There are two major steps involved in the usage of the system. The first step, which is shown in dashed lines, captures users' subjective perceptions of different QoS attributes through tuning fuzzy membership functions, which is introduced in detail in Subsection IV-B. The second step, which is shown in solid lines, involves the whole application deployment process, including requirement submission, discovery, trust evaluation, selection, and deployment.

The components of the system are explained below:

1) Web Interface: This layer provides users or cloud brokers with the entrance to all the services. Users can submit their functional and non-functional requirements, and change their perceptions through graphical interfaces.

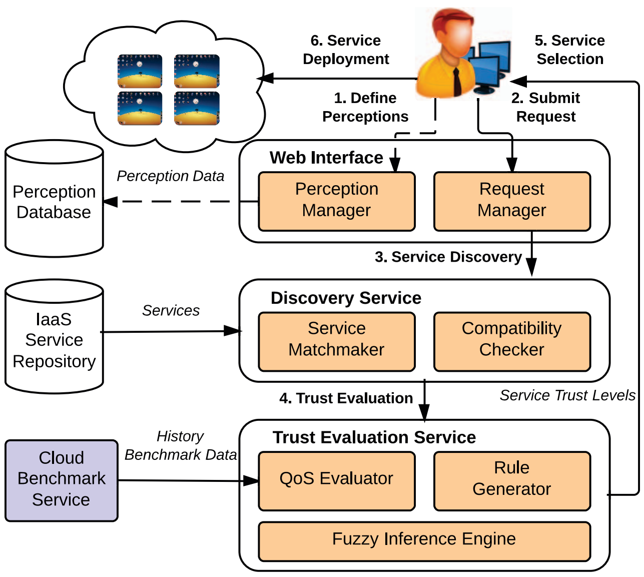


Fig. 2. Architecture of the Proposed Trust Evaluation System

2) Discovery Service: This component retrieves services that can meet users' functional requirements (i.e., number of cores, memory amount, storage volume, budget and geographical location) and some static QoS requirements (e.g., security and privacy) from the IaaS service repository. Another important function of it is to check the compatibility of services. It filters services that cannot satisfy users' business policies or are incompatible to software platforms required by their applications. Several tools and techniques have been developed for these purposes, such as the systems proposed by Dastjerdi et al [17] and Chunqing el al. [24].

3) Trust Evaluation Service: It is the core part of the system that evaluates the trust levels of functionally matched services. It takes user requirements and the services' past benchmark results as input and then outputs a list of services with their trust values regarding each attributes. Users or cloud brokers can then select the most suitable service based on the obtained trust values along with other objections (e.g., cost).

4) Cloud Benchmark Service: These services continuously monitor the performances of clouds by running benchmark applications on a number of dynamically launched VMs in a certain time interval and publishing the results to the public. They provide valid data traces regarding low level metrics that are required to fairly compare cloud services' QoS. An example of such service is CloudHarmony[2]. It now monitors 63 cloud data centers all over the world. However, as the number of providers continues to grow, benchmarking cloud services becomes financially unsustainable. The best solution is to advocate cloud providers to perform these standard tests by themselves and publish their results through open API. In addition, a trusted third party should regularly audit the clouds to ensure the integrity of the published results.

B. Modeling Requirements and Preferences

1) Requirement Types: To search the satisfactory services, our system requires users to first submit requirements for attributes they concern. It supports two types of requirements, namely, numerical requirement and linguistic requirement. Blended submission of different types of requirements in the same query is valid in our system. Section VI illustrates how different types of requirements can help users to define their expectations in selection process with two case studies.

a) Numerical Requirements: numerical requirements are numerical values with corresponding units. When submitting these requirements, users expect the services to have higher performances than the submitted values. The numerical requirements for some attributes, e.g., availability, can be easily extracted from the application's Service Level Objectives (SLOs). While for other attributes, e.g., CPU and network speed, they cannot be effortlessly determined because performance SLOs are usually defined in high level metrics, e.g., response time and throughput. This requires users to perform tests to transform high level SLOs to low level requirements.

b) Linguistic Requirements: linguistic requirements are submitted as fuzzy linguistic descriptors (e.g., $High$, $Medium$, and Low) which are semantic approximations of the numerical requirements. Since human beings are accustomed to use these linguistic descriptors to make rough estimations, they can easily submit meaningful requirements in this form according to application nature or preliminary results obtained through simple test or simulation. They are useful when it is uneconomic to perform complex tests or the task is urgent.

2) QoS Attributes: Table I shows an example of the hierarchy of the cloud QoS attributes, which we use in the prototype. We choose the attributes and their metrics according to SMI framework[3] and the work done by Garg et al. [6][7]. One can easily add extra attributes to the example model or use different metrics to measure existing attributes, e.g., IOPS (Input/Output Operations Per Second) for memory and disk performances. Furthermore, the system allows users to only submit requirements for the attributes they concern. In such cases, the system considers the performances of the overlooked attributes as fully satisfactory for all services in evaluation.

In general, we classify all leaf attributes in the hierarchy into two categories, namely, dynamic attributes and static attributes. Dynamic attributes are susceptible to performance variations. Therefore, their performances need to be quantified by benchmark traces. Compared with these attributes, the performances of other attributes can be considered static, i.e., security attributes, as their variations are negligible and unmeasurable. Our system quantifies the performances of security attributes in the form of single values instead of series of traces. They can be evaluated by cloud security benchmarks, e.g., the framework proposed by Garcia et al. [25], or expert ratings. We only allow users to submit linguistic requirements for static attributes, as for them, services can be binarily filtered at service discovery phase with given numerical thresholds.

Besides QoS, cost is also important to utility. Our system provides three different ways to balance users' QoS and cost objections. The first way is to specify a budget at service discovery phase. In this way, users can identify the most satisfactory service within an acceptable budget. For the second method, users can select the most economical cloud service among those have acceptable trust levels. The third way is to submit linguistic requirements for cost during the trust evaluation phase if users only have vague objections for cost.

3) QoS Inputs and Membership Functions: As the first step of trust evaluation, our system retrieves corresponding benchmark results from cloud benchmark services and then analyzes

[2]CloudHarmony http://cloudharmony.com/

[3]C.S.M.I.C http://csmic.org/understanding-smi/

TABLE I. AN EXAMPLE OF HIERARCHY OF CLOUD ATTRIBUTES AND METRICS FOR TRUST EVALUATION

Attributes			Metrics	Types	Requirements
Performance	CPU	CPU Speed	Maximum number of instructions executed by a single core in unit time (MIPS)	Dynamic	Numerical
	Memory	Memory Read	Maximum amount of data transferred from memory in unit time (Mb/s)		
		Memory Write	Maximum amount of data written to memory in unit time (Mb/s)		
	Storage	Disk Read	Maximum amount of data transferred from secondary storage in unit time (Mb/s)		
		Disk Write	Maximum amount of data written to secondary storage in unit time (Mb/s)		
	Network	Inbound	Maximum amount of data transferred into VM in unit time (Gbit/s)		Linguistic
		Outbound	Maximum amount of data transferred out of VM in unit time (Gbit/s)		
Assurance	Availability		The proportion of time that the VM is accessible (%)		
	Failure Rate		Average number of failures for one VM in one hour		
Elasticity	VM Startup Time		Time consumed to allocate, boot up, and configure VM (s)		
	VM Shutdown Time		Time consumed to shut down and deallocate VM (s)		
Security	Platform Security		Score of security mechanisms that protect virtualized platform	Static	Linguistic
	Data Security		Score of security mechanisms that protect users' data		
	Network Security		Score of security mechanisms that protect VMs from network attacks		
	Site Security		Score of security mechanisms that protect the data center		
	Security Policy		Score of policies that users can employ to implement their own security strategies		
Cost	VM Cost		The cost to deploy a VM ($/hr or $/month)		
	Data Transfer Cost		The cost to transfer data in or out of data center ($/Gb)		
	Storage Cost		The cost of secondary storage ($/Gb/Month)		

the traces to derive the QoS inputs of the hierarchical fuzzy inference system according to user requirements. Following, regarding the two types of requirements, we respectively introduce how the system calculates QoS inputs from benchmark traces and their associated input membership functions.

a) Retrieving Benchmark Traces: The benchmark traces used are key to the quality of trust evaluation. To retrieve representative traces, the system should consider multiple factors. In our system, we select traces according to the variability of attributes and expected running time of VMs.

Different attributes have different levels of variability. According to Schad et al. [2], VM startup time has very high variability in short time. For such attributes, the system should retrieve traces within a short time window, e.g., 10 minutes, to facilitate a valid trust evaluation.

The expected running time is also important when retrieving traces. Suppose a user wants to deploy a VM on Wednesday from 7:00 am to 2:00 pm, we should refer to traces that were benchmarked on nearest Wednesday or weekday within the same range of time. This is because clouds are likely to suffer more variations at busy hours on weekdays in their local time. Furthermore, if the user plans to deploy VMs for a long term, it is also better to consider the traces within a large time window to ensure the selected service is likely to be satisfactory in the whole life cycle of the application.

b) Numerical Requirements: For numerical requirements, the system calculates the QoS inputs of the i_{th} service regarding the j_{th} attribute as follow:

$$p_{i,j} = \frac{n^{satisfy}{}_{i,j}}{n^{total}{}_{i,j}} \tag{1}$$

where for the performance of i_{th} service regarding the j_{th} attribute, the numerator and denominator of the equation respectively stands for the number of benchmark traces that can satisfy the numerical requirement and the total number of traces retrieved. In fuzzy inference process, the membership functions associated with QoS inputs of numerical requirements are shown in Figure 3(a).

c) Linguistic Requirements: For linguistic requirements, the system calculates statistical indicators of the benchmark traces as the QoS inputs. In Section V, we test our system with different indicators, i.e., median, mean, first quartile, and five percentile.

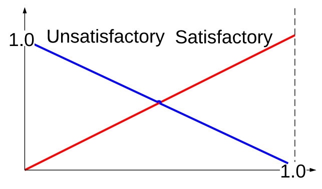
We use triangular membership functions to model linguistic requirements in fuzzy inference. Figure 3(b) shows an example of the membership functions. They represent users' personal perceptions that how quantitative performance data are mapped to qualitative linguistic descriptors for each attribute. Since the perceptions are purely subjective, we allow users to individually adjust the functions. They can either submit functions for each size of VMs or normalize their perceptions in a per-CPU-core-base. This is a one-time job and brings no extra burden to users when submitting requests. To further ease the task, especially for inexperienced users, the system provides a default setting of functions for each attribute.

4) Modeling Preferences: In our system, user preferences are modeled in both attribute and requirement level. Firstly, at attribute level, users express their preferences by submitting higher requirements for more important attributes, e.g, $High$ for CPU and Low for memory. Secondly, at requirement level, users can express their preferences by selecting one of the importance levels in Table II for each requirement. This allows users to make tradeoffs among their requirements in selection. For example, suppose a user wants to deploy a security sensitive application on cloud, he might prefer to select the services that can definitely satisfy his security requirements even at the cost of enduring more performance degradations for other attributes within an acceptable level. In this case, he can submit $Very\ Important$ for security requirement. To increase the room for tradeoff, he can submit negative importance levels, e.g., $Unimportant$, for requirements that he is willing to sacrifice more performances. We implement this mechanism using linguistic hedges, which are adapter functions that change the shapes of original membership functions. Following equations shows the formal definitions of importance levels for the two types of requirements:

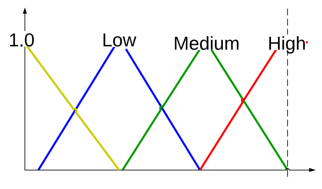
$$Numerical : d = \begin{cases} satisfactory(x)^{a_i} \\ unsatisfactory(x)^{\frac{1}{a_i}} \end{cases} \tag{2}$$

$$Linguistic : d = \begin{cases} f(x)^{a_i} & \text{if } x \leq peak \\ f(x)^{\frac{1}{a_i}} & \text{if } x > peak \end{cases} \tag{3}$$

where $satisfactory(x)$ and $unsatisfactory(x)$ respectively



(a) Numerical Requirements



(b) Linguistic Requirements

Fig. 3. Membership Functions for Hierarchical Fuzzy Inference System

represent $satisfactory$ and $unsatisfactory$ membership functions shown in Figure 3(a). $f(x)$ is a triangular membership function shown in Figure 3(b), and $peak$ is the x value of its maximum membership point. a_i is the importance coefficient of the i_{th} importance level. By applying positive importance hedges, e.g, $Important$, the system further penalizes the services that cannot fully satisfy the requirements in trust evaluation according to the amounts of performance deficiency. Oppositely, for negative hedges, e.g, $Unimportant$, it decreases the penalties. Table II shows the default importance levels and coefficients used in our prototype.

TABLE II. IMPORTANCE LEVELS AND IMPORTANCE COEFFICIENTS

Importance Level	Importance Coefficient
Very Important	1/2
Important	2/3
Neutral	1
Unimportant	3/2
Very Unimportant	2

C. Proposed Hierarchical Fuzzy Inference System

1) Overview: Our system dynamically constructs the hierarchical fuzzy inference system for each query in accordance to the generated hierarchy of attributes. Figure 4 shows an example of it. There are three types of inference modules in the system. For non-leaf attributes, their trust values are evaluated by higher level inference modules, which take outputs of corresponding sub-inference modules as inputs. At leaf level, the system generates inference modules according to the types of requirements, namely, numerical inference modules and linguistic inference modules.

Higher level inference modules use membership functions shown in Figure 3(a) as input membership functions. The input membership functions for leaf level modules have been introduced in Subsection IV-B. All modules use the functions defined in Figure 3(a) as output membership functions. Rulebases for each module are generated according to user requirements, the details of which is introduced in next part.

All inference modules share the same inference engine. Our prototype employs **Product** for "AND" operator, **MAX** for "OR" and Aggregation operator, and **Center of Maximum** (COM) defuzzifier. For each output variable with n membership functions, COM calculates the crisp trust value t as follow:

$$t = \frac{\sum_{i=1}^{n} x_i \mu_i}{\sum_{i=1}^{n} \mu_i} \tag{4}$$

where x_i is the x value of i_{th} membership function's maximum membership point and μ_i is the membership degree aggregated for the i_{th} membership function. The reason we choose COM is that it is both intrinsically plausible and timely efficient.

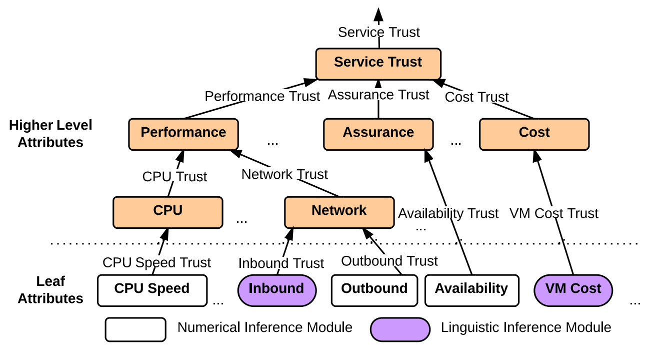


Fig. 4. Example of Hierarchical Fuzzy Inference System for Trust Evaluation

2) Generating "If-Then" Rules: The system dynamically generates the fuzzy rules for each module according to user requirements and employed rule generation strategy. In our prototype, we adopt a pessimistic strategy where output trust level is satisfactory only if all input variables are satisfactory, as we suppose users expect all submitted requirements should be individually satisfied. Following, we describe it in detail as an example of valid rule generation strategies.

a) Higher Level Inference Modules: The inputs of these modules are trust levels of their corresponding sub-attributes. Suppose attribute A have sub-attributes B and C, the generated rule set is like this:

Rule 1 If B_trust is $satisfactory$ AND C_trust is $satisfactory$ then A_trust is $satisfactory$

Rule 2 If B_trust is $not\_satisfactory$ OR C_trust is $not\_satisfactory$ then A_trust is $not\_satisfactory$

b) Leaf Level Inference Modules: For each leaf level inference module, the input is a single value obtained through analyzing benchmark traces. If user submits numerical requirement for attribute A, the generated rules are as follow:

Rule 1 If A is ($importance\ level$) $satisfactory$ then A_trust is $statisfactory$

Rule 2 If A is ($importance\ level$) $not\_satisfactory$ then A_trust is $not\_satisfactory$

Otherwise, suppose user submits linguistic requirement $Medium$ for attribute A, the result is as follow:

Rule 1 If A is at_least ($importance\ level$) $Medium$ then A_trust is $satisfactory$

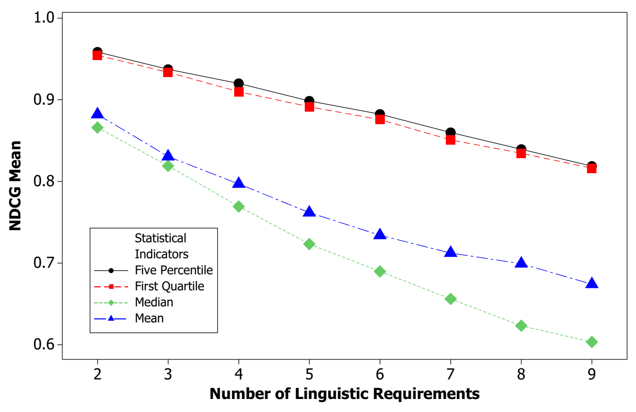
Rule 2 If A is at_most ($importance\ level$) Low then A_trust is $not\_satisfactory$

Low is the closest inferior linguistic descriptor to $Medium$. at_least is the linguistic hedge that transforms the membership function of $Medium$ into function "$\geq Medium$". Similarly, at_most transforms function of Low into "$\leq Low$". Following equations show their formal definitions regarding triangular membership function $f(x)$:

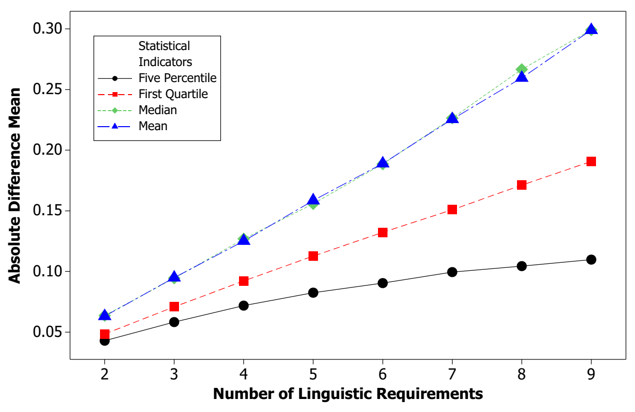
$$d_{at\_least} = \begin{cases} f(x) & \text{if } x \leq peak \\ 1 & \text{if } x > peak \end{cases} \tag{5}$$

$$d_{at\_most} = \begin{cases} 1 & \text{if } x \leq peak \\ f(x) & \text{if } x > peak \end{cases} \tag{6}$$

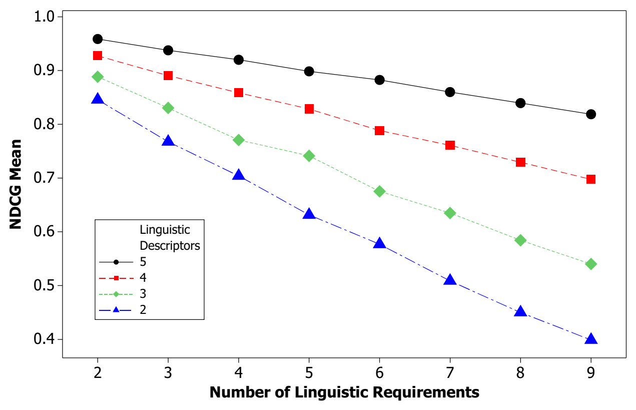
where $peak$ is the x value of the maximum membership point of the triangular membership function.



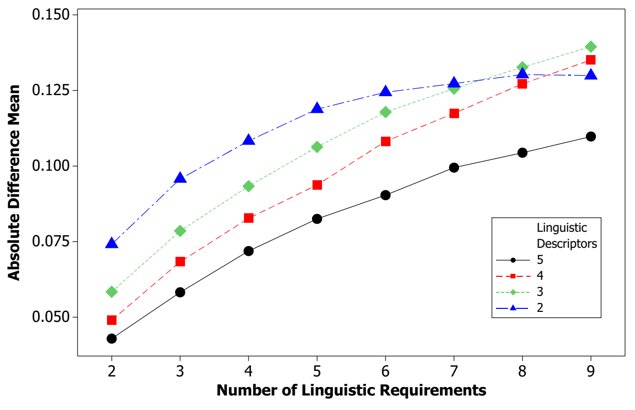
(a)



(b)



(c)



(d)

Fig. 5. (a) NDCG mean and (b) absolute difference mean using different numbers of linguistic requirements and different statistical indicators. (c) NDCG mean and (d) absolute difference mean using different numbers of linguistic requirements and different numbers of linguistic descriptors.

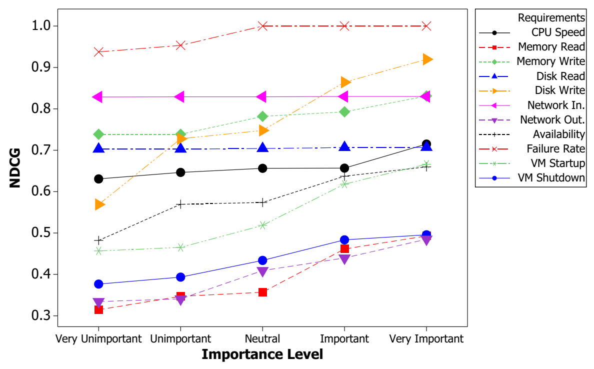


Fig. 6. Effect of different importance levels on the ranking of services.

V. Performance Evaluation

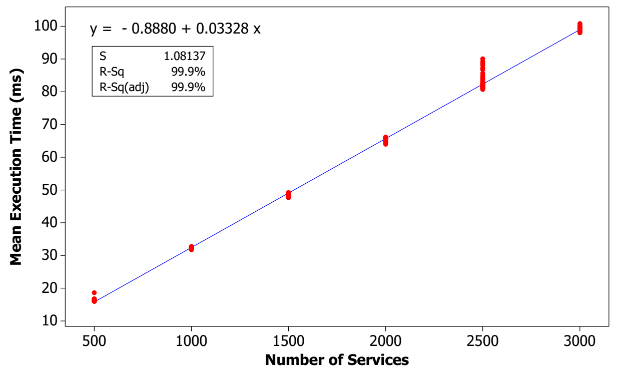
A. Generating Benchmark Traces for Simulations

Schad et al. [2] ran performance benchmark applications (e.g., Ubench and Bonnie++) on newly launched VMs every hour in a month on Amazon EC2. They did not only quantify the performance variations regarding multiple attributes in a cloud, but also successfully identified their performance distributions. From the data analysis, they found that the performance instability in Amazon is mainly caused by hardware heterogeneity and workload increase in peak time. Since these are common problems faced by all cloud providers, we assume the performance distributions of other services also generally follow the same patterns as the ones observed in Amazon but with different levels of variability and means.

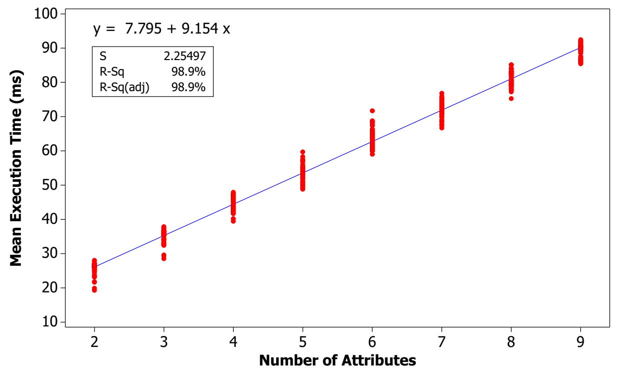
We test our system through simulations using synthetic benchmark traces. Each service trace consists of 50 to 200 benchmark results for each of the 11 dynamic attributes defined in Table I. Some benchmark results are generated according to the known distribution patterns mentioned by Schad et al. [2]; others are normally generated. In total, we created 3000 services with similar comprehensive performances but diverse performances regarding each attribute. This avoids the system always returning the same omnipotent services and enables us to study how our system makes tradeoffs among attributes.

B. Linguistic Requirements vs Numerical Requirements

In the first experiment, we test the validity and accuracy of using linguistic descriptors to approximate numerical requirements. To compare the results obtained by the two, we dynamically convert some of the numerical requirements in 25 numerical queries into linguistic requirements. This is done by selecting the linguistic descriptor that produces the highest membership degree with given numerical values and predefined membership functions. By doing so, we assume users are able to intuitively submit the closest linguistic descriptors to numerical requirements, which is the ideal situation. We use average Normalized Discount Accumulative Gain (NDCG) at position 10 to measure whether our system still can rank most satisfactory services at the top when some of the requirements are submitted in linguistic descriptors. The relevance scores used to calculate NDCG are trust levels derived from



(a) Mean Execution Time vs Number of Services



(b) Mean Execution Time vs Number of Attributes

Fig. 7. Mean execution time with different numbers of services and attributes

original numerical queries. In addition, we measure absolute differences of trust levels obtained by the paired queries to gauge evaluation accuracy. Apart from number of linguistic requirements, we also consider other factors that may influence accuracy, i.e., statistical indicators used to calculate the QoS inputs and number of linguistic descriptors (Low, $High$ or Low, $Media$, $High$). For each number of linguistic requirements, we run 55 tests with different combinations of attributes submitted in linguistic requirements. We use 5 linguistic descriptors for tests using different statistical indicators, and five percentile for tests using different numbers of linguistic descriptors. Figure 5 reports the average results of the tests.

According to the results, both the quality of ranking and evaluation accuracy decrease when the number of linguistic requirements grows, since they bring extra uncertainties into the system. In addition, the decreases are generally linear. This indicates limiting the number of linguistic requirements can largely improve the chance of finding the most satisfactory service. Comparing different statistical indicators, it shows that conservative indicators, i.e., five percentile and first quartile, produce better results. This is because they are more sensitive to variability and hence increase the system's differentiability. The data also reveal that increasing number of linguistic descriptors can improve both quality of ranking and evaluation accuracy. However, on the other hand, it makes more difficult for users to come out with sensitive judgements if the number of descriptors is large. Further studies are required to find the number of descriptors that can best balance the two factors.

C. *Effectiveness of Preference Modeling*

In the second experiment, we demonstrate the effectiveness of our preference modeling approach. For a query where all its requirements are submitted with $Neutral$ importance level, we select one requirement a time and change its importance level to construct new queries. We also use NDCG at position 10 as metric, but the relevance scores used in this experiment are the trust values of the selected requirement with $Neutral$ importance level. In this way, we show whether our system ranks services that have lower performance deficiencies to the selected requirement higher when the importance level of the requirement increases.

According to the results shown in Figure 6, average NDCG remains unchanged for some requirements when their importance level goes up, and the scale of NDCG increase for other requirements also varies. This is because the space for tradeoff is limited. For example, though one cloud can fully satisfy requirement A, as long as its performance deficiencies to other requirements are unacceptable, the system considers it unsatisfactory no matter how the user increases A's importance level. This ensures tradeoffs are only made within acceptable level and returned services can reasonably satisfy all requirements.

D. *System Scalability*

The third experiment tests the scalability of our approach. We first measure mean execution time of 25 queries each with requirements for 11 attributes when the number of services is increased from 500 to 3000. Next we test the mean time spent to evaluate 25 queries for 3000 services when the number of leaf attributes considered is increased from 2 to 9. We run 55 tests for each number of services and attributes on a PC with Intel i7-2600 CPU and 4 Gb RAM using single thread.

As observed in Figure 7, our approach is timely efficient and linearly scalable. The execution time can be further reduced if we parallelize the evaluation process.

VI. Case Studies

In this section, we demonstrate how inexperienced users and expert users can make use of our system to select the satisfactory service and improve their cost-efficiency. Table III shows the basic information of the three IaaS cloud services involved in the following illustrations.

TABLE III. IaaS Cloud Services

Service	CPU Core	Memory(Gb)	Disk(Gb)	Price($/hr)
A	1	1.7	20	0.07
B	1	2.0	25	0.10
C	1	1.8	18	0.09

A. *Case 1*

A user who worked for a pharmaceutical company wanted to run a simulation in cloud. For functional requirements, he required the service to have 1 core, more than 1.5 Gb RAM and 18 Gb secondary storage. Though he had limited knowledge of computer science, he knew the program is CPU and data intensive but requires little network communications when the simulation starts. Hence he submitted $High$ requirements for attributes related to CPU, memory and disk performances and Low requirements for network attributes.

Our system filtered service C at discovery phase as it could not provide enough storage. Then it retrieved benchmark traces of A and B for the past few hours to evaluate trust of their current performances. The trust values obtained were 0.85 for A and 0.91 for B, which were all acceptable to the user. Since service A offered a cheaper price, user decided to select A.

B. Case 2

A game company developed an online game and planed to deploy it on cloud with a budget of \$0.09/hr for each single core instance. They agreed with their end users that the service should be available 99.999% every year. Through preliminary experiments, they estimated the application requires 7 MIPs CPU and 0.8Mb/s inbound and outbound network speed for a single instance to support 50 requests in parallel. They also identified network as the bottleneck to their application performance, so they promoted the importance level of network to $Very\ Important$. Their application is undemanding for disk as it depends on the database deployed on local servers, so they ignored disk I/O attributes. However, they lack enough time to conduct further tests to determine the thresholds of memory I/O due to development delays and incoming online date. Therefore, they decided to utilize linguistic requirements to first find a reasonably satisfactory service for their application and then redeploy it to the most satisfactory one later. They were aware that there are many sequential memory read and write operations in their program. Thus they submitted $High$ requirements for both memory read and write speed attributes.

Since B was over-budget, our system only retrieved traces regarding A and C. The traces, in this case, contained benchmark results conducted on A and C in the previous month as the application was supposed to run for a long time. The returned trust levels were 0.41 for A and 0.92 for C. Though C was more expensive, they still decided to deploy their application on C because A suffered more variations and likely would require them to deploy more VMs in peak hours.

After further development and tests, they identified the numerical requirements for memory I/O. Apart from that, they are now able to migrate the instances among different clouds without down time. To increase cost-efficiency, the company check the trust levels of all the cloud services every two hours and dynamically migrate their application to service that offers the best utility at the moment.

VII. Conclusions and Future Directions

In this paper, we proposed an effective trust evaluation system using hierarchical fuzzy inference system for IaaS service selection. The contributions of our system can be concluded as follows: 1) it enables trust evaluation of IaaS clouds according to user requirements, 2) it eases the IaaS selection process for both inexperienced and expert users by modeling their vague requirements and uncertain preferences with linguistic descriptors and hedges, and 3) it improves cost-efficiency and service stability when using clouds by considering performance variations in the selection phase.

In future, we are going to develop selection polices for cloud users and brokers based on our trust evaluation system to further automate cloud deployment.

References

[1] R. Buyya *et al.*, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] J. Schad, J. Dittrich, and J.-A. Quian-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," in *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, 2010, pp. 460–471.

[3] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, pp. 200–209.

[4] Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, 2007.

[5] ur Rehman *et al.*, "IaaS cloud selection using MCDM methods," in *Proceedings of the 2012 IEEE Ninth International Conference on e-Business Engineering (ICEBE)*, pp. 246–251.

[6] S. K. Garg *et al.*, "SMICloud: A framework for comparing and ranking cloud services," in *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*, pp. 210–218.

[7] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012 – 1023, 2013.

[8] Z. ur Rehman, F. K. Hussain, and O. K. Hussain, "Towards multi-criteria cloud service selection," in *Proceedings of the 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pp. 44–48.

[9] M. Menzel and R. Ranjan, "CloudGenius: Decision support for web server cloud migration," in *Proceedings of the 21st International Conference on World Wide Web (WWW)*. 2187967: ACM, pp. 979–988.

[10] Z. u. Rehman, O. K. Hussain, and F. K. Hussain, "Multi-criteria IaaS service selection based on QoS history," in *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 1129–1135.

[11] S. Wang, Z. Liu, Q. Sun, H. Zou, and F. Yang, "Towards an accurate evaluation of quality of cloud service in service-oriented cloud computing," *Journal of Intelligent Manufacturing*, pp. 1–9, 2012.

[12] V. Torra, "A review of the construction of hierarchical fuzzy systems," *International Journal of Intelligent Systems*, vol. 17, pp. 531–543, 2002.

[13] H. M. Alabool and A. K. Mahmood, "Trust-based service selection in public cloud computing using fuzzy modified vikor method," *Australian Journal of Basic and Applied Sciences*, vol. 7, no. 9, pp. 211–220, 2013.

[14] T. H. Noor and Q. Z. Sheng, "Trust as a service: a framework for trust management in cloud environments," in *Proceedings of the 12th International Conference on Web Information Systems Engineering (WISE)*. Springer, 2011, pp. 314–321.

[15] S. M. Habib, S. Ries, and M. Muhlhauser, "Towards a trust management system for cloud computing," in *Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, pp. 933–939.

[16] M. Supriya *et al.*, "Estimating trust value for cloud service providers using fuzzy logic," *International Journal of Computer Applications*, vol. 48, no. 19, pp. 28–34, 2012.

[17] A. V. Dastjerdi *et al.*, "An effective architecture for automated appliance management system applying ontology-based cloud discovery," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, pp. 104–112.

[18] S. Sundareswaran *et al.*, "A brokerage-based approach for cloud service selection," in *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pp. 558–565.

[19] C. Redl *et al.*, "Automatic SLA matching and provider selection in grid and cloud computing markets," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing (GRID)*, pp. 85–94.

[20] S. Nepal *et al.*, "A fuzzy trust management framework for service web," in *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS)*, pp. 321–328.

[21] P. Wang, "QoS-aware web services selection with intuitionistic fuzzy set under consumer's vague perception," *Expert Systems with Applications*, vol. 36, no. 3, pp. 4460–4466, 2009.

[22] S. Song *et al.*, "Trusted P2P transactions with fuzzy reputation aggregation," *Internet Computing, IEEE*, vol. 9, no. 6, pp. 24–34, 2005.

[23] S. Song, K. Hwang, and M. Macwan, *Fuzzy Trust Integration for Security Enforcement in Grid Computing*. Springer, 2004, pp. 9–21.

[24] C. Chunqing *et al.*, "A systematic framework enabling automatic conflict detection and explanation in cloud service selection for enterprises," in *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pp. 883–890.

[25] J. L. Garcia *et al.*, "Benchmarking cloud security level agreements using quantitative policy trees," in *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*. 2381932: ACM, pp. 103–112.