A Cloud Brokerage Architecture for Efficient Cloud Service Selection

Dan Lin, Anna Cinzia Squicciarini, Venkata Nagarjuna Dondapati, Smitha Sundareswaran

Abstract—The expanding cloud computing services offer great opportunities for consumers to find the best service and best pricing. Meanwhile, it also raises new challenges for consumers who need to select the best service out of such a huge pool since it will be time-consuming for consumers to collect the necessary information and analyze all service providers to make the decision. Therefore, in this paper, we propose a novel brokerage-based architecture in the cloud, where the cloud brokers is responsible for the service selection. We also design an efficient indexing structure, called B^{cloud}-tree, for managing the information of a large number of cloud service providers. We then develop the service selection algorithm that recommends most suitable cloud services to the cloud consumers. We carry out extensive experimental studies on real and synthetic cloud data, and demonstrate a significant performance improvement over previous approaches.

Index Terms: Cloud computing, Cloud brokerage, Service selection, B^{cloud}-tree, Indexing, Querying

1 Introduction

Cloud services offer an elastic and scalable variety of storage space and computing capabilities, which are crucial to most business owners, especially small and medium sized businesses [31]. To facilitate cloud users in terms of cloud service discovery, mediation and monitoring [7], [19], many cloud brokerage mechanisms have been proposed [2], [4], [7], [15], [19], [23], [34]. Among various responsibilities that a cloud broker can carry, the very first important task is to help cloud consumers select the appropriate cloud services that satisfy their requirements. Existing works [12], [17], [24], [25], [32] typically focus on what criteria to be considered (e.g., QoS, price) and how to consider selection criteria (e.g., which criteria is more important to cloud users). With the significant increase of cloud adoption [21] and the large number of newly emerging cloud providers and various types of cloud services, a new challenge rises during the cloud selection, that is how to efficiently select the best cloud services from a huge pool. For example, a single broker like Equinix [8] has already had more than 500 registered cloud providers, and a single cloud provider like Amazon provides more than 70 types of cloud services. Multiplying the numbers of current cloud providers and their service types while considering the further growth of these numbers in the near future, cloud service selection has turned into a more and more time consuming task. There is a clear need to develop an efficient mechanism to better serve the cloud users.

In order to speed up the cloud service selection process, we propose a generic cloud brokerage architecture which contains an efficient indexing structure and a powerful query engine for the service selection. The overall architecture is illustrated in Figure 1. The cloud broker has a collection of cloud providers' profiles which include the types of services, pricing and other system information. We propose a B^{cloud}-tree for the cloud broker to classify and store various types of cloud services as well as managing service information updates. The cloud broker takes cloud consumers' requirements as input and groups similar queries that issued during the same time interval. Then, the cloud broker invokes the query engine to search the B^{cloud}-tree to identify the services that satisfy the consumers' requirements. The main contributions of our work are summarized as follows.

- We propose a new indexing structure, the B^{cloud}-tree, which is capable of storing, indexing and handling updates of a large volume of dynamic cloud service information.
- We proposed an efficient query algorithm that supports the generic type of service selection queries, i.e., queries that allow users to specify service selection requests that contain intervals of desired values (e.g., a price range) of multiple properties.
- We studied real cloud providers and evaluated our proposed approach using datasets generated based on real scenarios. Our experimental results have shown the significant performance improvement of our approach against the existing method.

The remaining of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the brokerage-based service selection architecture. Section 4 presents an indexing approach to manage cloud services' information and Section 5 presents the service selection algorithm. Section 6 reports the experimental results. Finally, Section 7 concludes.

- *D. Lin is with the Department of Computer Science, Missouri University of Science & Technology, Rolla, MO, USA. E-mail: lindan@mst.edu*
- *A. Squicciarini is with the College of Information Sciences & Technology, The Pennsylvania State University, University Park, PA, USA. E-mail: asquicciarini@ist.psu.edu*
- *V. N. Dondapati the Department of Computer Science, Missouri University of Science & Technology, Rolla, MO, USA. E-mail: nvdrnc@mst.edu*
- *S. Sundareswaran is with the College of Information Sciences & Technology, The Pennsylvania State University, University Park, PA, USA. E-mail: sus263@psu.edu*

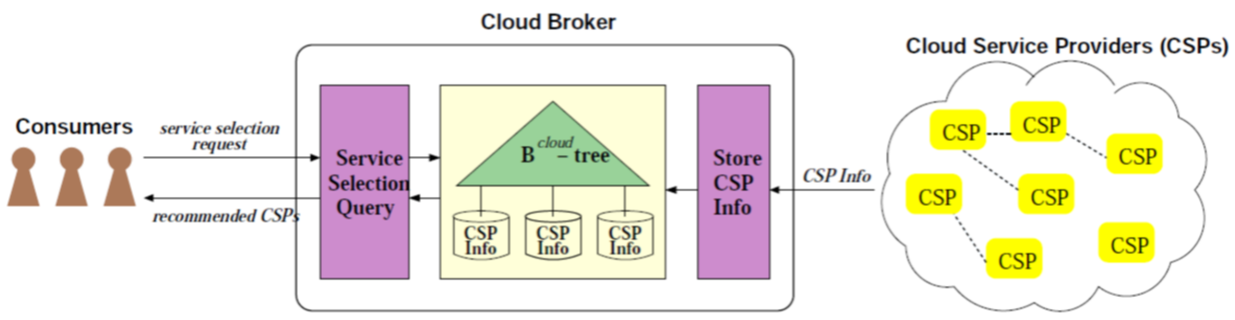


Fig. 1. An Overview of the Cloud Brokerage Model

2 Related works

Cloud brokerage has attracted increasing attention from both industry and research communities. In industry, one of the earliest cloud brokers could be CloudSwitch [6], established in 2008 with service for only Amazon EC2. CloudSwitch has the ability to provide federated services on demand and make the cloud a secure and seamless extension of the enterprise data center. RightScale [26] is another cloud broker that offers a cloud management platform to facilitate deployment and management of applications across multiple clouds. More recently, Equinix [8] has also attracted more than 500 cloud providers to register.

Many research efforts have also been devoted into cloud brokerage [2], [4], [7], [15], [19], [23], [34] which propose different types of brokerage framework, service aggregation, resource sharing and allocation, etc. Specifically for the cloud service selection task carried by the cloud brokers, one early effort is by Han et al. [12] who proposed a ranking of available cloud providers based on QoS and Virtual Machine (VM) platform factors. To reduce the number of QoS criteria for evaluation and improve selection accuracy, Qi et al. [24] propose a service selection method based on weighted Principal Component Analysis dedicated to multimedia service selection in the cloud. Since many works consider mainly cost and performance as the selection criteria which may not be sufficient, Rehman et al. [32] defined a general mathematical model to support multi-criteria cloud service selection. To move theory to practice, Jrad et al. [17] developed a cloud broker system that is capable of automatically selecting cloud services based on userdefined requirement parameters and the service level agreement attributes of the cloud providers. The limitation is that their algorithm needs to scan all cloud providers and compare the user's requirement with each of the service provider, which could be time consuming when the number of cloud providers is large. Recently, Qu et al. [25] extend the service selection criteria from objective metrics such as price to subjective metrics such as user feedback in order to make the service selection more effective. To further improve the accuracy of subjective metrics, Esposito et al. [9] address uncertainty in the expression of subjective preferences from customers by integrating the fuzzy set theory and game theory into the service selection process. Similarly, Sun et al. [1] also leverage the fuzzy logic to handle uncertainty during the service selection and provides a multi-criteria-based service ranking. Chang et al. [5] added another selection criteria which aims to maximize the data survival probability or the amount of surviving data. They propose a dynamic programming algorithm based on the famous knapsack problem. Unlike most past works and our work which select a single service type for a cloud user, Wang et al. [33] point out the need to have a combined cloud service and propose an adaptive learning mechanism to help cloud users to combine different service types into an integrated cloud service.

In addition, there have been recent studies on security and trustworthiness issues in cloud service selection [3], [10], [11], [18], which are orthogonal to our work.

As a summary of the state-of-the-art cloud service selections, a recent survey can be found in [29] which classifies the cloud service selection approaches into four main categories: (i) MCDM (Multi-Criteria Decision Making)-based approaches; (ii) Optimization-based approaches; (iii) Logic-based approaches; and (iv) others. Among these categories, the optimization-based approaches are the most related to our work as some of which also consider the efficiency aspect. The one that specifically targets large-scale scenarios is by He et al. [14]. They propose a QoS-driven approach to help cloud service developers to compose multi-tenant SaaS (Software-as-a-Service), which achieves SaaS providers' optimization goals while fulfilling the end-users' different levels of QoS constraints. They employ the skyline and greedy algorithms to reduce the search space of the optimization formulae in a large-scale scenario. Unlike their work which aims to find a combination of cloud services, our work is focused on finding the single best service. Moreover, we also employ a different way, i.e., the indexing technique to address the efficiency problem. Another most related work is our own previous work [30] which will be discussed in Section 3.3.

3 The Cloud Brokerage Model

As shown in Figure 1, our proposed cloud brokerage model supports three types of entities: (i) cloud users (consumers); (ii) cloud brokers; and (iii) cloud service providers (CSP). The cloud broker serves as a middle man in-between the CSPs and users. Specifically, the cloud broker, which may have a contract with the CSPs, maintains latest information about the CSPs such as their service types, unit costs, and available resources. When a user is looking for certain kind of cloud services, he/she submits the desired service requirements to the cloud broker. The broker searches its database and recommends the best available cloud services to the user.

Realizing this brokerage architecture includes two key technical challenges. The first challenge is that the cloud broker should efficiently manage the potentially large amount of CSPs' information and ensure that they are up-to-date. For example, the available resources of a CSP should be timely updated once a new type of service is established or a service is terminated. The second challenge is that the broker should respond to a user's request promptly. This requires the broker to have the ability to quickly retrieve, compare and rank all the cloud services against the user's requirements.

To address the first challenge, we leverage indexing techniques that groups cloud services according to the similarity of their properties. Then, we develop efficient query algorithms to identify cloud services that satisfy users' requirements. We elaborate these two techniques in Section 4 and 5 respectively. For better understanding of our approach, we first introduce the properties of cloud services considered in our work, and give the formal definition of a user's service request.

3.1 Properties of Cloud Services

A CSP may provide various types of cloud services. This work focuses on ten properties most commonly acknowledged as relevant properties of cloud services. We discuss the approach used to identify these properties in Section 6.1.

- `Service Type` (p_1). This denotes the type of service provided, such as storage service, computing service, database service, management tools, mobile services, etc.
- `Security` (p_2). This denotes the level of security and/or privacy that can be achieved using the various options provided by the cloud provider.
- `Quality of service` (QoS, p_3). QoS is determined by the cloud broker which analyze the collected information about service providers over time, and ratings provided by other vendors.
- `Measurement units` (p_4). This represents in what terms the service can be charged. Measurement can be in terms of memory used, the number of the transactions, the number of connections or data transfers, or the time taken for the data transfer.
- `Pricing Units` (p_5). This indicates what price is charged per time unit and whether there are any upfront fees. For example, p_5 can be used to represent: "charge per hour (no upfront fee)", "charge per hour with upfront fee", "charge per month (no upfront fee)", "charge per month with upfront fee", etc.
- `Instance sizes` (p_6). This refers to the amount of resources used at a given instant by the user. The size may vary from micro (in case of Amazon EC2) to small, medium, large, or extra large to something such as quadruple extra large provided by Amazon EC2.
- `Operating system` (p_7). This indicates that the operating system could be Linux or Windows.
- `Pricing` (p_8). This is the actual price for the usage of the cloud service.
- `Pricing sensitivity` to regions (p_9). This denotes if the price varies by region.
- `Subcontractors` (p_{10}). This indicates if subcontractors are present, and if so, what kind of services they provide.

3.2 Types of User Requests

A user sends a service selection query to the broker which specifies what properties and values he/she expects from the service providers. Our brokerage system supports two types of queries as defined in the following.

Definition 3.1: An ***exact query*** *on cloud services is in the form of* $Q = \langle (QP_1 : V_1), (QP_2 : V_2), \ldots, (QP_k : V_k) \rangle$*, where* $k \leq 10$*,* QP_i *(*$1 \leq i \leq k$*) is the property that the user requests the cloud service to possess, and* V_i *describes the user expected value of property* QP_i*.*

Definition 3.2: An ***interval query*** *on cloud services is in the form of* $Q = \langle (QP_1 : I_1), (QP_2 : I_2), \ldots, (QP_k : I_k) \rangle$*, where* $k \leq 10$*,* QP_i *(*$1 \leq i \leq k$*) is the property that the user requests the cloud service to possess, and* I_i *describes the range of user expected values of property* QP_i*.*

An example of an exact query may look like: $Q_1 =$ ⟨(ServiceType:0001), (Cost:60cents/min)⟩. An example of an interval query may look like: $Q_2 =$ ⟨(ServiceType:[0001]), (pricing:[$0.3/min, $0.8/min]), (security:[medium, high])⟩.

At the user-end, a GUI to enter user's input is provided, to facilitate property selection. The user needs not enter values for the entire list of properties, but only those that are relevant to him/her. The use of such GUI also ensures that the input for the cloud broker is in a machine recognizable form so that the broker does not need to further clean the user's input before processing it.

For the exact query, the cloud broker aims to return the best match, i.e., the cloud service that matches the most number of query conditions. For the interval query, the cloud broker aims to return m cloud services that match the query conditions in a descending order of the number of conditions being satisfied. Here m is a value provided by the user. For example, a user may be interested in 10 ($m = 10$) or 20 ($m = 20$) potential cloud services rather than a very large number (e.g., 100) of cloud services that would be hard for him/herself to choose from.

3.3 Our Previous Work

Since the new approach presented in this paper is compared to our previous work, we review our previous work in more details as follows. In [30], we proposed a CSP-index to index CSPs (cloud service providers) according to the similarity of their properties. The CSP-index is developed based on a multi-dimensional index, called iDistance [16] with the B$^+$-tree as the base structure. In order to capture the similarity among CSPs, we proposed an encoding technique that encoded each CSP's properties using a bit array. The bit array is of the same size for every CSP and consists of 10 sections corresponding to the 10 properties identified in Section 4.1. The number of bits used for each section is based on the domain of each property and the encoding differs according to the types of the properties. Based on individual property encodings (denoted as e_1, ..., e_{10}) where e_i is the encoding of property p_i, we further generate the integrated encoding by concatenating the bits representing the service type with the XOR-ed results of the remaining property encodings.

$$E_{csp} = e_1 || (e_2 \oplus e_3 \oplus e_4 \oplus e_5 \oplus e_6 \oplus e_7 \oplus e_8 \oplus e_9 \oplus e_{10}) \quad (1)$$

After obtaining the encoding for all the CSPs, we employ the k-means algorithm [13] to cluster the CSPs based on the Hamming distance between their encodings E_{csp}. Then we leverage the idea of the iDistance [16], [36] to generate the indexing key Key_{csp} for each CSP.

$$Key_{csp} = S \cdot k + D_h(E_{csp}, E_{c_k}) \quad (2)$$

In Equation 2, E_{c_k} is the encoding of the cluster center that the CSP belongs to, S is a scaling value that separate indexing values from different clusters, and D_h is the Hamming distance between the CSP and its cluster center. The obtained index keys are used to insert CSPs to a B$^+$-tree. Using Equation 2, CSPs with similar properties are likely to obtain closer indexing keys and hence will be placed nearby in the B$^+$-tree.

One limitation in the CSP-index is that it not only puts similar CSPs together but may also group dissimilar CSPs, which can in turn affect the query efficiency. This is clearly shown in the following example.

Example 3.1: For ease of illustration, we consider only four properties per CSP. Consider the following three CSPs whose properties p_1, p_i, p_j, p_k have been encoded:

CSP$_1$(e_1=0, e_i=1, $e_j = 5$, $e_k = 9$)

CSP$_2$(e_1=0, e_i=9, $e_j = 1$, $e_k = 5$)

CSP$_2$(e_1=0, e_i=9, $e_j = 5$, $e_k = 1$)

$E_{csp_1} = 0||(1 \oplus 5 \oplus 9)$

$E_{csp_2} = 0||(9 \oplus 1 \oplus 5)$

$E_{csp_3} = 0||(9 \oplus 5 \oplus 1)$

The above three CSPs will obtain exactly the same encoding since XOR operation ignores the order of the properties. Consequently, the three CSPs will be placed nearby in the index. However, we can observe that they are not very much similar in that CSP$_1$ and CSP$_2$ differ in all three properties considered, and CSP$_1$ and CSP$_3$ differ in two properties out of three.

In summary, the XOR encoding allows even dissimilar CSPs to receive the same encoding as long as these CSPs have the same set of values regardless what properties the values are for. We call such a problem "encoding collision". The *encoding collision* problem becomes much more severe with the increase of the number of the properties associated with a CSP. When 9 properties are considered (excluding the service type property), a same set of values can be assigned to the 9 properties in $9! = 362880$ ways. In other words, 362880 CSPs share the same encoding (and index key) while many of these CSPs are not similar at all. Such encoding collision significantly affects the query efficiency. In this work, we propose a new encoding approach that addresses this problem by taking into account the order of properties, and the experimental results show that our new approach is hundreds of times faster. Moreover, our previous approach does not distinguish different types of services under the same CSP which may introduce uncertainty during the query. To overcome this problem, our new approach treats various service types of the same CSP independently as well as supporting the more general type of query, i.e., the interval query (Definition 3.2).

4 Managing Cloud Service Information

As aforementioned, due to the huge amount and heterogeneity of the cloud services and resources, it is a very challenging task for cloud consumers to identify the cloud services that meet their needs best. Thus, in our proposed cloud brokerage system, the cloud broker takes over the service selection tasks for end users. In order to facilitate the cloud brokers to answer a potentially large number of cloud end users' requests in a timely fashion, it is important to design an efficient data structure to facilitate information management and retrieval on cloud services. The specific design goals are the following.

- If the broker needs to update a cloud service's information such as change of properties and available resources, the broker should be able to quickly locate where the cloud service is stored rather than scanning the whole database.
- Given a user's service selection request, the broker should be able to narrow down the search within a small group of eligible cloud services rather than checking all cloud services against the user's request.

To achieve the above goals, we propose a B^{cloud}-tree which provides quick access to individual cloud service and also groups cloud services with similar properties to facilitate queries. In what follows, we elaborate how to construct the B^{cloud}-tree to realize the design goals.

4.1 The Structure of the B^{cloud}-tree

The B^{cloud}-tree is a multi-level balance tree as shown in Figure 2. Each node in the B^{cloud}-tree is of the same size (typically the size of a disk page), and contains a number of equal-size entries. An entry in the leaf node stores the following information of a cloud service: $\langle \kappa, ID, p_1, p_2, ..., p_{10} \rangle$, where κ is the indexing key of the cloud service whose identity is ID, and p_i is the i^{th} property of the cloud service ($0 \leq i \leq 10$). The internal nodes (including the root) of the B^{cloud}-tree serve as the search directory which contain indexing keys and pointers to children nodes. Each internal node contains n indexing keys and n+1 pointers in the form of $\langle pnt_1, \kappa_1, pnt_2, \kappa_2, pnt_3, ..., \kappa_n, pnt_{n+1} \rangle$, whereby pnt_i points to the child node whose indexing keys fall into the range of $[\kappa_{i-1}, \kappa i]$.

The B^{cloud}-tree has a similar structure as the B$^+$-tree. The biggest advantage of basing the B^{cloud}-tree on the B$^+$-tree is that the B$^+$-tree provides the great foundation for our new index structure to be easily integrated to existing systems since the B$^+$-tree is widely adopted in commercial database systems. Moreover, we can also leverage the same suite of efficient B$^+$-tree's algorithms to insert, delete and update the information of a cloud service (i.e., an entry in a leaf node). For example, to update the properties of a cloud service with a known indexing key, we start from the root of the B^{cloud}-tree, and look for the internal pointer that leads to the range of keys including the cloud service's indexing key. Only h nodes need to be

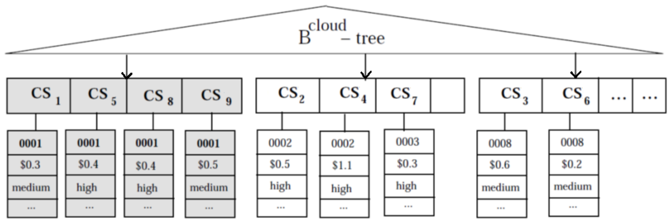


Fig. 2. An Example B^{cloud}-tree

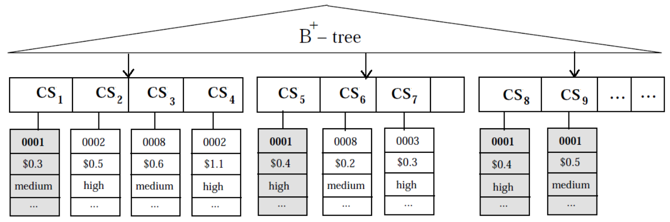


Fig. 3. An Example B$^+$-tree

accessed for this update where h is the height of the tree and typically 3 (sufficient for 1M cloud services). More discussion about the computational complexity is in Section 5.5.

In the design of the B^{cloud}-tree, the main challenge is the generation of the indexing key. Specifically, the indexing keys determine the storage order of cloud services. How the cloud services are stored has a great impact on the efficiency of answering a user's service request. Reconsider the example query $Q_1 = \langle$(ServiceType:0001), (Cost:\$0.6/min)$\rangle$ in the previous section. The B$^+$-tree that indexes cloud services simply based on their IDs may store them as shown in Figure 3. We can observe that to answer the user's request, the B^{cloud}-tree (Figure 2) just needs to access one leaf node (shaded node) while the traditional B$^+$-tree needs to conduct a linear search to check all the leaf nodes because the cloud services of type "0001" (shaded nodes) may be stored in any leaf node. Based on the observation, the desired indexing keys in the B^{cloud}-tree should be generated in the way that cloud services with similar properties receive nearby indexing keys. We present the detailed algorithm of key generation in the next subsection.

4.2 The Construction of the B^{cloud}-tree

As aforementioned, the novelty of the B^{cloud}-tree is the construction of the indexing keys that can speed up the query processing. The difficulty lies in how to capture the similarity among cloud services, and then convert them into a single key value. It is obviously time consuming to compare every pair of cloud services to identify which of them are similar to each other. This is also impractical concerning the changing of cloud service's properties, joining of new cloud services or removing of old services. Therefore, we propose a hash style key generation method, which allows us to directly compute the indexing key for any given cloud service based on its properties and ensures that most of the resulting keys have the following properties: the more similar the cloud services are in terms of their properties, the closer their indexing keys will be.

The algorithm to generate the index keys consists of two major steps. The first step is to encode the properties of the cloud services so that they can be input to the key generation function. The second step is to compute the indexing key. We elaborate each step in the following.

4.2.1 Property Encoding

The property encoding transforms various types of cloud service properties into decimal values. The encoding algorithms differs according to the types of the properties.

- **Service Type** (p_1): Since services may be described in different ways, we employ the oneR mining algorithm on the service type to identify cloud services that perform similar services. According to the mining result, service types falling into the same group will be assigned the same encoding. For example, if there are total 100 types of services identified, the domain of the encoding will be from the numerical value 1 to 100.
- **Properties with continuous values**: This category includes the properties: pricing (p_8), instance sizes (p_6) (e.g., requested storage capacity), pricing unit (p_5) and measurement unit (p_4). For such type of properties, we first partition the domain of the corresponding property into n ranges, where n is a tunable system parameter. Then we represent each range using one bit. If a cloud service's property falls into a given range, the respective bit will be set to 1. Finally, we convert the binary representation to a decimal value.

  Example 4.1: Assume that the domain of storage space provided is divided into four ranges: [10G,∞), [1G,10G), [500M,1G), [0,500M). If the storage capacity of a service provider is 800M to 2G, the second and the third bits will be set to 1, resulting in the encoding '0110'.
- **Properties with categorical values**: This category includes the following properties: security (p_2), quality of service (p_3), operating system (p_7), pricing sensitivity (p_9). Such properties are typically represented by a categorical value. For each of the categorical value, we assign a distinct numerical value. Specifically, the security property has one of the three values: high, medium and low, which is obtained as follows. If the service provider satisfies three or more of the Information Security guidelines listed in the Security Guidance for Critical Areas of Focus in cloud Computing published by the cloud Standards group [27], apart from the compliance or risk management guidelines, the service providers are classified as having high security. If they satisfy only the compliance, legal or risk management guidelines they are classified as having medium security. Else they are classified as having low security. Accordingly, when the service provider offers advanced security services such as Access Control, offered by Windows Azure, or it adopts a number of security standards (e.g., secure connections), the level of security is labeled as *high*. When there are security options, but these options are limited to secure passwords and encryption as in case of

Rackspace, the level of security that can be achieved is considered to be *low*. When the features do not include detailed access control options, but still provide improved security through automatic security updates, as Google does for its Clouds, then the security level is considered to be *medium*. The property "quality of service" is also described using "high", "medium", "poor" based on the review ratings generated by the cloud broker or the cloud users over time. Correspondingly, we convert these categorical values into numbers "3"(high), "2"(medium), "1"(low).

- **Relationship property**: This refers to the specifical property "Subcontractor" (P_{10}) which describes the relationship among the cloud services built upon subcontracting. We represent the relationship using a binary bit array with three bits. The first bit is set to 1 if subcontractors are present. The second bit is set to 1 if the subcontractor provides computational or storage services. The third bit is set to 1 if the subcontractor provides security, privacy, or search related services. Then, the binary value is converted into a decimal value.

In addition, if a service provider does not have specific value for certain properties, the encoding of that property will be set to the default value 0.

4.2.2 Indexing Key Generation

A cloud service is described by 10 decimal values, using the property encoding described in the previous section. Next, we aim to generate the indexing keys based on the encoded properties of the cloud service. Recall that the indexing key should ensure that the cloud services with similar properties (i.e., the encoding value of the properties) have close indexing keys. This will ensure that they will be placed closer in the index, speeding up the subsequent service selection. To achieve this, we leverage one of the space-filling curves, the Z-curve [22]. The reason to choose the Z-curve is many-fold. First, Z-curves have been implemented in many commercial databases just like our base structure, the B$^+$-tree, facilitating potential deployment of our approach in practice. Second, the Z-curve can be computed efficiently. Third and the most important, the Z-curve maps a data point in a multi-dimensional space to a single-dimensional value. The resulting one dimensional value has the proximity property, that is, for points which are closer in multi-dimensional space, they are very likely to receive closer one-dimensional value. Figure 4 shows an example of data points in the 2-dimensional space and their Z-curve values (in each grid cell). For example, consider the data points (1,1), (1,2), which are near each other in the 2-dimensional space. Their Z-curve values are 1 and 2 respectively, which are also closer to each other. If we consider the coordinates of the data points as the properties of cloud services, the corresponding Z-curve values can then be used as the indexing keys. Notice that there are some cases where the Z-curve values cannot fully preserve the proximity properties, such as data points (2,4) and (3,1) which are far from each other in 2-dimensional space but receive nearby Z-curve values 8 and 9 respectively. However, our experimental results (in Section 6) show that such small

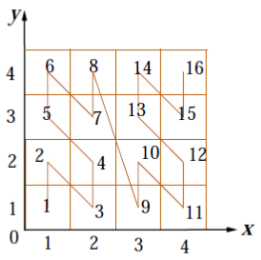


Fig. 4. An Example Z-Curve

deviation does not significantly affect the accuracy.

In fact, we apply the Z-curve mapping only to the last 9 properties excluding the service type to ensure that cloud services of the same type are grouped together. The steps of generating the indexing keys are the following.

- Given a service $CS_1(p_1, ..., p_{10})$ and its property encodings denoted as $CS_1(e_1, ..., e_{10})$, the broker converts each encoding into a binary representation: $e_i \rightarrow be_i$ $(1 \leq i \leq 10)$. The binary representation is normalized to be of the same length for all the properties. The normalization is done by taking the longest binary value as the standard length l of all binary values. If an initial binary value does not have l bits, we fill up its beginning bit positions with 0.
- Then, the broker concatenate the binary encoding of the service type with the result of interleaving the binary values of the remaining 9 property's encodings. Let $[be_i]_j$ denote the j^{th} bit of be_i. The interleaving process is as follows, where the symbol "$||$" denote bit concatenation.

$$Z = [be_1]||[||_{j=1}^{l}(||_{i=1}^{9}[be_i]_j)] \quad (3)$$

- The last step is to convert Z into a decimal value, which would be the indexing key of the CS_1: $Z \rightarrow \kappa$.

For better understanding, we step through the key generation processing using the following example.

Example 4.2: Consider the following small set of properties with divided value domains for example:

- *Service type: 0001, 0002,...,1000*
- *Storage: [10G, ∞), [1G, 10G), [500M, 1G), [0, 500M)*
- *Pricing: [50cents/min, 1 Dollar/min], [10 cents/min, 50 cents/min), [0, 10cents/min)*
- *Quality of service: 3-High, 2-medium, 1-poor*
- *Security: 3-High, 2-medium, 1-poor*

Suppose that a service provider CS_1 provides service type '0001', 800M to 2G storage space to each end user at 10 cents/min with medium service quality and medium privacy protection. The corresponding encoding of each property is: '0110'(storage), '010'(pricing), '010'(service quality), '010'(privacy). After normalization, we have the following binary values for each property:

service type: '0001'
storage: '0110'
pricing: '0010'
privacy: '0010'

Next, we interleave the property encodings as shown in Figure 5 where the arrows indicate the interleaving order, and obtain the encoding '0001000100111000'. Its corresponding decimal value is 4408, which will be used as the indexing key for this cloud service.

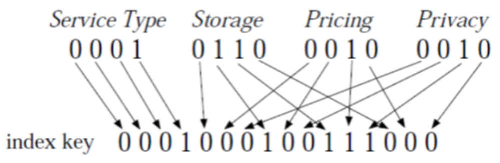


Fig. 5. Generating the Index Key through Interleaving

Once the indexing key is generated, the insertion and deletion in the B^{cloud}-tree resembles that in the B^{+}-tree.

5 Cloud Service Selection

The B^{cloud}-tree helps the broker arrange the service providers in a way that facilitates fast information retrieval. Recall that two common types of service selection are defined in Section 3.2: (i) the exact query; (ii) the interval query. Both types of queries can be answered by the following four major steps: (1) Query normalization and grouping; (2) Query encoding; (3) Searching the B^{cloud}-tree; (4) Refining the search results and considering special criteria. The query encoding converts the user query into the form of intervals of indexing keys that cover cloud services which may satisfy the query. Based on the query encodings, we then search the B$^{-}tree$ to locate the candidate cloud services. The last step is to further examine the properties of the candidate cloud services and their relationship to find the best combination that addresses the user's needs. We elaborate the major steps of each phase in the following subsections.

5.1 Query Normalization

For both exact queries and interval queries, the first step is the query normalization which fills in the non-query properties using the property domains, so that all the 10 properties are associated with a query domain. Since the exact query is a special case of an interval query, we define the query normalization on the interval query directly.

Definition 5.1: Query Normalization: Let Q=(QP_1:I_1, ..., QP_k:I_k) be an exact or an interval query. For each property $p_j \notin QP_i$ ($1 \leq i \leq k$), create a query interval NP_j = (p_j : D_1) where D_j is the domain of the property p_j. Then, a normalized Q' will be in the form of Q=(QP_1:I_1, ..., QP_k:I_k, NP_1, ..., NP_n), where n is the total number of properties not listed in Q.

For example, the interval query Q_2 in Section 3.2 will be normalized to the following query Q_2'.

Q_2' = ⟨(**p_1:[0001]**), (**p_2:[2..3]**), (**p_8:[\$0.3/min..\$0.8/min]**), (p_3:[1..3]), (p_4:[1..4]), (p_5:[1..4]), (p_6:[1..4]), (p_7:[1,2]), (p_9:[0..1]), (p_{10}:[000, 111])⟩.

In Q_2', the first three properties (highlighted in bold) are the query properties provided by the user, and the remaining properties are the results of the normalization using the domains of the corresponding properties (details about the property domains are shown in Table 1 in Section 6).

It is worth noting that in the case that multiple requests are received at the same timestamp (which may occur in large-scale systems), the broker will group the same queries and then perform normalization to avoid repeated efforts.

5.2 Query Encoding

The second step of service selection is to convert the normalized user query into the ranges of indexing keys so that the search of cloud services can be conducted in the B^{cloud}-tree. The simplest case is an exact query with all 10 properties given by the user. For such type of exact query, we just need to treat the query as a new cloud service, and then encode the query properties and generate the indexing keys using the same algorithm presented in the previous section.

However, most of service selection queries include only a subset of cloud service's properties and ranges of possible values for the querying properties. The task is to find the indexing keys of the cloud services whose properties fall into the query ranges. A naive approach is to consider all the combination of values in the query ranges, compute the indexing keys, and then search the B^{cloud}-tree using the obtained indexing keys to locate the qualifying cloud service. Take the previously normalized interval query Q_2' as an example. All of the following cloud services satisfy the query:

CS_2: (**p_1:[0001]**), (**p_2:[2]**), (**p_8:[\$0.3/min]**), (p_3:[1]), (p_4:[2]), (p_5:[4]), (p_6:[1]), (p_7:[2]), (p_9:[0]), (p_{10}:[000])⟩.

CS_{10}: (**p_1:[0001]**), (**p_2:[2]**), (**p_8:[\$0.5/min]**), (p_3:[3]), (p_4:[4]), (p_5:[2]), (p_6:[2]), (p_7:[1]), (p_9:[1]), (p_{10}:[111])⟩.

CS_{13}: (**p_1:[0001]**), (**p_2:[3]**), (**p_8:[\$0.4/min]**), (p_3:[2]), (p_4:[3]), (p_5:[1]), (p_6:[4]), (p_7:[1]), (p_9:[0]), (p_{10}:[110])⟩.

Observe that the cloud services satisfying the query may have very different property values for non-querying properties.

Instead of computing the indexing key for each possible combination, we propose to compute the boundaries of the indexing keys belonging to all the cloud services that satisfy the query, and then execute interval queries rather than single point queries in the B^{cloud}-tree. As shown in Figure 6 (a), conducting multiple point search to locate each possible cloud service results in accessing a large number of internal nodes of the B^{cloud}-tree (highlighted as red paths); whereas, conducting an interval search to locate the cloud services accesses much fewer tree nodes as shown in Figure 6(b), and hence will be more efficient.

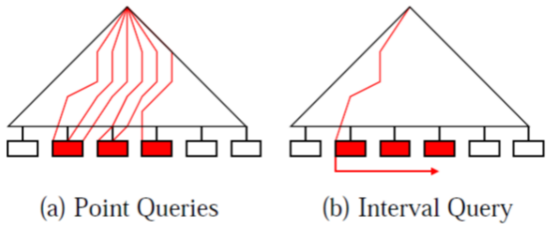


Fig. 6. Point Query vs. Interval Query for Service Selection

To compute the query interval, we leverage the properties of the Z-curve. We note that the minimum and maximum indexing key values of a query are reached by taking all the lower and upper boundaries of the query properties respectively. The formal definition of the boundaries of the query indexing keys is given below.

Theorem 5.1: Let Q=(QP_1:$[V_1^l, V_1^u]$, ..., QP_k:$[V_k^l, V_k^u]$, $NP_1 : [V_{k+1}^l, V_{k+1}^u]$, ..., $NP_n : [V_{10}^l, V_{10}^u]$) be a normalized interval query, where V_j^l and V_j^u denote the lower and upper bounds of the interval respectively. Let κ^l be the indexing key computed from (QP_1:V_1^l, ..., QP_k:V_k^l, $NP_1 : V_{k+1}^l$, ..., $NP_n : V_{10}^l$), and let κ^u be the indexing key computed from (QP_1:V_1^u, ..., QP_k:V_k^u, $NP_1 : V_{k+1}^u$, ..., $NP_n : V_{10}^u$). Then, for any CS=(p_1,.., p_{10}) whose $p_j \in [V_j^l, V_j^u]$ (for all $1 \leq j \leq 10$), its indexing key κ_{cs} will be in the range: $\kappa^l \leq \kappa_{cs} \leq \kappa^u$

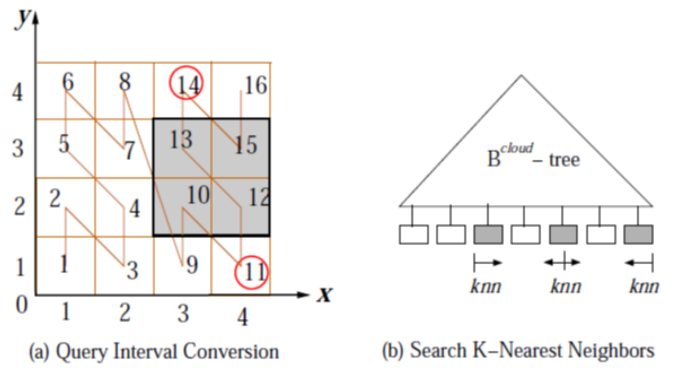


Fig. 7. Query Encoding

The range of the one-dimensional query interval obtained from Theorem 5.1 may contain false positives which are the cloud services whose indexing keys fall in the one-dimensional query interval converted from the boundaries of the query properties, but whose actual properties do not satisfy the query conditions. For illustration, Figure 7(a) shows an example when each cloud service has only two properties and hence each cloud service is represented as a point with the two properties being the coordinates. Consider a service selection query (denoted as the shaded box in the figure) on two properties x and y which requires qualifying cloud services to have their properties in the following ranges: $3 \leq x \leq 4$ and $2 \leq y \leq 4$. This query will be converted into a 1-dimensional interval [10, 15] according to the Z-curve. Then, only the cloud services whose index keys are in [10, 15] will be evaluated. As shown in the figure, there are two index key values 11 and 14 (circled in the figure) which are in the 1-dimensional query interval but not the original 2-dimensional query (the shaded box). In order to reduce checking such false positives, we propose a search algorithm that takes three values of indexing keys: the minimum, the median, and the maximum. We then conduct the k nearest neighbor search for these three values as shown in Figure 7b (detailed search algorithm is presented in the following subsection).

5.3 Search in the B^{cloud}-tree

The output from the last step will contain three indexing key values: minimum boundary (denoted as κ_{min}), median (κ_{med}), and maximum (κ_{max}). For each of the query indexing key, we search from the root of the B^{cloud}-tree and locate the leaf node that contains the range of keys including this query indexing key. It may happen that the leaf node does not have the exact value of the querying key since there is not such a service provider who matches the query requirements. In this case, the broker will find the closest value to the querying indexing key and start the search. From the located starting value (either the exact querying indexing key or the closest one), we will find its k nearest neighbors. The search of the k nearest neighbors differs a bit as shown in Figure 7b: for κ_{min}, we look for k neighbors which have key values greater than it; for κ_{med}, we look for k neighbors from both sides; for κ_{max}, we look for k neighbors that are smaller than it. During the search, if the leaf node containing the query indexing key does not have k entries, we expand the search to its neighboring leaf nodes along the search direction until k nearest neighbors are found.

Here, the chosen of the value of k, i.e., the number of neighbors to be considered, is critical to the overall performance. If too few neighbors are retrieved, the broker may not find the service provider which fully satisfy the query requirements. This is because the B^{cloud}-tree stores service providers according to the similarity between all of their properties, in order to be versatile for different queries. A specific query usually focuses on a smaller set of properties, and hence the k nearest neighbors retrieved based on all properties may not contain the best solution regarding the querying properties. On the other hand, if k is too large, it will slow down the search process as well as the subsequent refinement phase. This value of k is therefore decided by a trial and error process. We will present the tuning of the k in the experiments.

5.4 Refinement

From the obtained candidate service providers, the refinement phase further evaluates the actual properties of the candidate service providers and finds the service providers or the combinations of service providers that satisfy the query requirements. As for service combination, we only consider the combination of cloud storage services in this work. It is worth noting that our query algorithm can serve as the foundations for selecting candidate services for complex service combinations [20] that involve various aspects such as software and platforms.

Algorithm 1 outlines the major steps in the refinement phase. The first step is to check whether the properties of a candidate service fully satisfies the query requirements. If so, the candidate service will be added to the query result. After examining all candidate services, if the result set is not empty, we will directly return the services in the result set to the users. Otherwise, that means none of the candidate service fully satisfy the user's requirement. If the dissatisfaction occurs only because of lack of storage capacity by a single cloud service provider, we will pursue the following steps to identify combined services.

To find the combinations of the services that satisfy the user's storage request, we first sort the candidate services obtained from the original query based on their storage capacities in a descending order. Then, we enumerate all possible combinations of services that can satisfy the storage request. Specifically, we select the service on top of the sorted list and deduct the the storage capacity offered by this service from the desired goal. Then, we consider the second candidate service in the list and repeat the same process until the desired storage capacity is reached or exceeded. For example, if the user requests $20GB$ of storage space, while the selected service only has 5GB available, we adjust the querying property on storage space to 15GB (=20GB-5GB) and look for more candidate services. Once a solution is found, we start the process by considering the next possible combination. The obtained combined set of services is sent to the next phase to verify possible collision among them caused by the shared subcontractors. If there exists any collision in current solution, the collision checking step will return a ranking for this solution to indicate its collision degree. The higher the ranking, the less the collision or collusion. The service selection process will be repeated to find other possible combinations of service providers until a better solution cannot be found. The final output of the service selection algorithm may contain a ranked list of solutions to let end users to make the final decision.

The possibility of a collision between cloud services needs to be considered during their selection. *Collision* is the occurrence of a lack of a promised immutable resource due to the dependence of the selected services on the same contractor who promises the resource to all of them, not accounting for a simultaneous demand from all. For example, let us consider two services CS_1 and CS_2, who both lease 50 GB of storage space from a subcontractor CS_3. CS_3 can guarantee both CS_1 and CS_2 the 50GB provided there are no restrictions on the region in which the storage servers are located, being that it has a part of its servers in USA and the rest in China. When a user requests a 100GB of storage space, CS_1 and CS_2 can fulfill this requirement together by depending on CS_3. However, if the user specifies that the servers should be located only in USA, then a collision occurs if the shared subcontractor is not taken into account.

To detect collision, we verify the subcontractor encoding (p_{10}) of the services involved in a compound service to see if they have any subcontractor in common. The verification is conducted by checking the encoding of the property p_{10} for each pair of cloud services in the same compound service. If any two services have at least two common bits set to 1, that means they have subcontractors for same type of services. In that case, we further check the actual subcontractors to see if there is any collision. Only when there is no collision, the answer will be returned.

Algorithm 1 Refinement(Q,L_{cs})

```
Require: $L_{cs}$ is a set of candidate services that returned by
    the query $Q$
 1: $FinalResult \leftarrow \emptyset$
 2: for $i \leftarrow 1$ to $|L_{cs}|$ do
 3:   if $L_{cs}.[i]$ satisfies all query properties then
 4:     $FinalResult \leftarrow FinalResult \cup L_{cs}.[i]$
 5:   end if
 6: end for
 7: if $FinalResult$ is empty because of low in storage
    capacity then
 8:   $L'_{cs} \leftarrow$ sort $L_{cs}$ in descending order of storage capacity
 9:   CombineService($Q$,$L'_{cs}$,$\emptyset$,$\emptyset$,$m$)
10: end if
11: Check collision in $FinalResult$
12: Return $FinalResult$
```

Algorithm 2 CombineService(Q,L'_{cs},Now,$Combined$,m)

```
Require: $L'_{cs}$ is the sorted candidate services of the query
    $Q$, $Now$ is the set of current combined services and
    $Combined$ stores all sets of combined services found so
    far, $m$ is the index number of the service to be checked
    next.
 1: for $i \leftarrow m$ to $|L'_{cs}|$ do
 2:   if $L'_{cs}.[i]$ satisfies all query properties except $Q$.storage
      then
 3:     $Now.L \leftarrow Now.L \cup L'_{cs}.[i]$
 4:     $Now.storage \leftarrow Now.storage + L'_{cs}.[i].storage$
 5:     if $Now.storage \geq Q.storage$ then
 6:       $Combined \leftarrow Combined \cup Now.L$
 7:     else
 8:       CombinedService($Q$,$L'_{cs}$,$Now$,$Combined$,$i+1$)
 9:     end if
10:     $Now.L \leftarrow Now.L - L'_{cs}.[i]$
11:     $Now.storage \leftarrow Now.storage - L'_{cs}.[i].storage$
12:   end if
13: end for
14: Return $Combined$
```

5.5 Computational Complexity Analysis

We now analyze the computational complexity of the cloud information management and cloud service selection using the proposed B^{cloud}-tree.

Let n denote the total number of cloud services stored by the B^{cloud}-tree. Let f denote the average number of entries in a node of the B^{cloud}-tree. The height of the B^{cloud}-tree is $h = log_f(n)$. When $n = 10,000$ and $f = 100$, the height h is only 2. In other words, the height h is a very small constant.

To insert, delete or update a cloud service information in a B^{cloud}-tree, there are three steps. The first step is to simply convert the 10 service' properties to numerical values, which has the computational complexity of O(1). The second step is to generate the index key based on the properties, which is a concatenation of 10 binary values and the complexity is O(1) too. The third step is the actual insertion, deletion or modification in the B^{cloud}-tree. This involves the examination of the root node and one internal node at each level of the tree in the best case. In the worst case when there is need to split or merge nodes, the process will involve two nodes per level of the tree. The total number of comparisons can be estimated as $2f \cdot h = 2f \cdot log_f(n)$. Thus, the computational complexity for managing cloud information is O(log(n)).

As for the cloud service selection, we consider the general case, i.e., the interval query. Let Q denote the query and m denote the total number of services that satisfy the query. The query starts from the root and then traverse one internal node per level down to the leaf node. Then, the query traverse the leaf nodes that may contain qualifying services. The number of entries visited at root node and the internal nodes is $f \cdot (log_f(n) - 1)$. Next, we estimate the number of visited entries in the leaf nodes. According to [35], nearest neighbor search using the Z-curve will search about 1.5 times entries than the actual query results regardless of the dimensions. In our case, we search k nearest neighbors, which approximates $1.5^k \cdot m$ entries in the leaf nodes. By summing up all entries visited by a query from the root node to the leaf node, a search visited total $f \cdot (log_f(n) - 1) + 1.5^k \cdot m$ entries. Since $m \ll n$, the service selection complexity is O(log(n)) too.

6 Performance Study

In this section, we first describe the collection and generation of the datasets, and then present the performance evaluation of our algorithms. All the algorithms were implemented as C programs. The tests were conducted using a Sony Vaio F series Laptop, with a 8GB DDR3-SDRAM-1333, 640GB Hardrive and a Intel Core i7-2820QM quad-core processor (2.30GHz) with Turbo Boost up to 3.40GHz.

6.1 Generation of Testing Datasets

In order to identify what is the actual information that a broker should account for when performing the service selection, we studied the profile of the top ten cloud service providers [2]. Our analysis included providers offering storage services or the Platform as a Service (Rackspace, Salesforce, cloud Foundry from VMWare), enterprise cloud platforms (CloudSwitch from Verizon, IBM cloud), and service providers who offer multiple types of services (Microsoft Azure, Amazon EC2, and Google cloud).

To extract functional and non-functional properties of each provider, we first analyzed the providers' available manifests including documents related to security practices, privacy policies, the cloud documentations on getting started and other user guides, FAQs, white papers, Terms of use, and Service Level Agreements (SLAs). We then identified and extracted a set of common properties based on common business recommendations for service selection [27], [28]. Table 1 provides an excerpt of our data collection analysis which shows the first 9 properties as introduced in Section III.A.1. Specifically, `Service Type` of type 1 refers to service on-demand, 2 refers to reserved instances, while 3 refers to specialized services such as custom IPs in case of Amazon, or caching in case of Windows Azure. `QoS` should be determined by the cloud broker over time through continuous evaluation of the CSP. For simulation, we use the aggregated user ratings from existing cloud service review websites in our experiments. Specifically, if a cloud service provider receives review ratings above 4/5, we categorize its QoS as “High”. If a cloud service provider receives review ratings between 3/5 to 4/5, we categorize it as ”Medium”. For the remaining, we categorize it as “Low”. `Msrmt` stands for measurement units, where 1 refers to measurement in terms of memory used, 2 refers to measurement in terms of number of transactions, 3 stands for number of connections or data transfers done and 4 for the data transfer time. `Prcg unts` stands for Pricing Units, where 1 stands for per month, 2 per year, 3 per 3 years, and 4 per hour. `IS` denotes Instance sizes where 1 refers to Small and anything below small such as Micro in case of Amazon EC2, 2 for Medium, 3 for Large, while 4 for extra large and above such as Quadruple extra Large provided by Amazon. `OS` is the operating system. A value 1 corresponds to Linux, while 2 is Windows. `Prc` stands for pricing and is normalized to per hour for each SP. `Reg` stands for regions where `Yes` denotes that pricing varies by region, while `No` denotes there is no differential pricing across various regions. Based on the collected real data, we identified the acceptable values for each of the properties (using ranges), based on the maximum and minimum service levels offered for a given property by any of the service providers. This gave us our starting set of ten data points and shaped the representation of service providers. With the starting data points, we generated 10,000 data points representing synthetic providers. Each synthetic providers was generated using random combinations for each of the properties describing it. Specifically, we use a pseudo random number generator to generate a subset of the total possible 10^{10} combinations, and filter out the outliers wherein all the properties have either very low or very high values.

6.2 Experimental Results

We compare the query performance of the B^{cloud}-tree with our previous work, the CSS algorithm [30], the B+-tree which indexes the cloud services' ID (as described in Section 4.1), and a baseline approach which uses an exhaustive search to check all possible combinations of all service providers for a given query and find the service providers that match the query properties best. The performance is evaluated in terms of both efficiency and accuracy. Efficiency is measured using the processing time. Accuracy is measured by comparing the results obtained from our proposed query algorithms with that obtained from the baseline approach. Specifically, the query accuracy is defined in Equation 4, where N_q is the average number of query properties being satisfied in the results obtained from our proposed approach (B^{cloud}-tree or CSS), and N_{base} is the average number of query properties being satisfied in the results obtained from the baseline approach.

$$Accuracy = \frac{N_q}{N_{base}} \tag{4}$$

6.2.1 *Performance of Exact Queries*

First, we evaluate the performance of exact queries by varying the number of cloud services and the number of querying properties.

- Effect of the Number of Cloud Services

In this round of experiments, we generate 100 exact queries, each of which includes 5 (out of 10) randomly chosen query properties and values. We execute the 100 queries against different number of cloud services ranging from 1000 to 10,000. Figure 8 shows the query processing time of the four

TABLE 1
Cloud Service Provider's Properties and Domains

Cloud Provider's Name	Variable Names								
	Service Type	Sec	QoS	Msrmt	Prcg unts	IS	OS	Prc	Reg
Amazon EC2	1, 2, 3	High	High	1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4	1, 2	0.000 - 2.60	Yes
Windows Azure	1, 2, 3	High	High	1, 2, 3, 4	1, 4	1, 2, 3, 4	2	0.04- 0.96	No
Rackspace	1, 2, 3	Low	Medium	1, 3, 4	1, 4	2	1, 2	0.015 - 1.08	No
Salesforce	1, 2, 3	Low	Medium	4	1	N/D	N/D	2 - 260	No
Joynet	1, 2, 3	Low	Medium	4	1, 4	3, 4	1, 2	0.085 -2.80	No
Google Clouds	1, 2, 3	Medium	High	1, 2	1, 4		N/D	0.0057 - 0.0068	No

approaches: the B^{cloud}-tree, the CSS algorithm, the B+-tree and the baseline approach (the results are plotted on a log scale). It is not surprising to see that the B$^+$-tree and the baseline approach are the slowest. The baseline approach is slow because it needs to check all the cloud services for each service selection query. The B$^+$-tree is even slightly slower than the baseline approach because the B$^+$-tree not only needs to check all the cloud services in its leaf nodes (as explained in previous Figure 3) but also needs additional time to access its internal nodes. In contrast, our proposed new approach, the B^{cloud}-tree, performs best among the all. Specifically, the B^{cloud}-tree is about 100 times faster than the B$^+$-tree and the baseline approach, and more than 10 times faster than our previous CSS approach when the number of cloud services is more than 5000. Such good performance of the B^{cloud}-tree is attributed to the better grouping of similar cloud services using the new encoding method as described in Section 4.2. As similar cloud services are stored closer to each other in the B^{cloud}-tree, a query needs to retrieve much fewer number of nodes to find the answers. Specifically, we tested various values of k in the k-nearest neighbor search adopted by the B^{cloud}-tree and found that $k = 5$ already yields high accuracy for different datasets as reported in this experimental section, which is much smaller than that in the CSS approach. The CSS approach requires relatively a large k (k is set to 10% of the total number of cloud services) in order to find answers of reasonable accuracy, because the encoding used by the CSS may give totally different cloud services the same encoding as shown in Example 2.1.

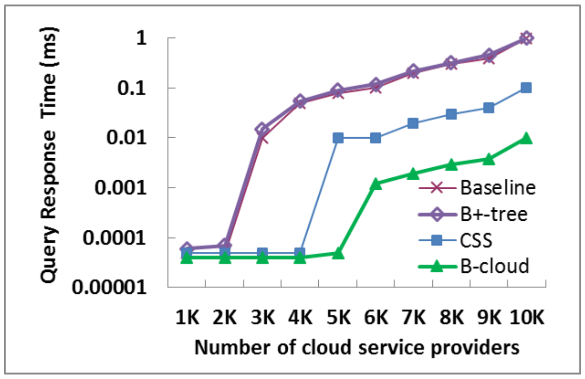


Fig. 8. Effect of the number of cloud services on service selection time (Exact Queries)

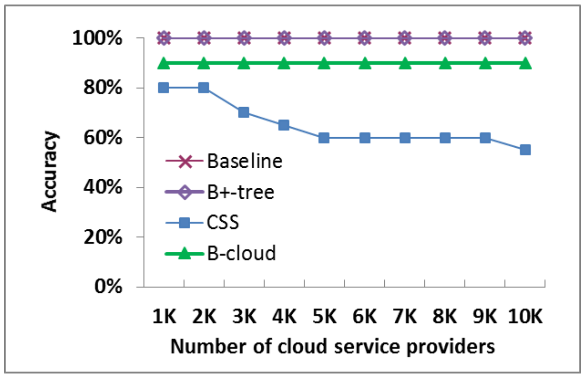


Fig. 9. Effect of the number of cloud services on accuracy (Exact Queries)

The accuracy of the query results of the four approaches is reported in Figure 9, where the baseline approach has always 100% accuracy since it is used as the base for comparison according to Equation 4. The B$^+$-tree achieves 100% accuracy since it needs to check all cloud services as well due to the lack of the search power of the B$^+$-tree on multiple search criteria. As we can see that the B^{cloud}-tree achieves much higher accuracy than the CSS approach which is mainly due to the effectiveness of the encoding method adopted by the B^{cloud}-tree. Moreover, the accuracy of the CSS approach decreases with the increase of the number of cloud services. This is because the more cloud services, the higher chance of having more cloud services with same encoding but totally different values of their properties caused by the ignorance of the order of properties in the encoding (as shown in Example 2.1).

Finally, since the B$^+$-tree has same accuracy as the baseline approach but is even worse than the baseline approach in terms of efficiency, we do not include the B$^+$-tree in the remaining experiments.

- Effect of Number of Query Properties

In this set of experiments, we vary the number of querying properties per request from 1 to 10 and test them in the dataset containing 5000 service providers. We report the average cost of 100 queries with the same number of querying properties for each setting. Figure 10 compares the average query processing time of the three approaches. Observe that the number of query properties does not affect the performance of the three approaches much. The reason for the baseline approach is straightforward since no matter what queries are, the baseline approach needs to check all the cloud services. As for the CSS approach, its query processing time is dominated by the total number of cloud services since the value of k in

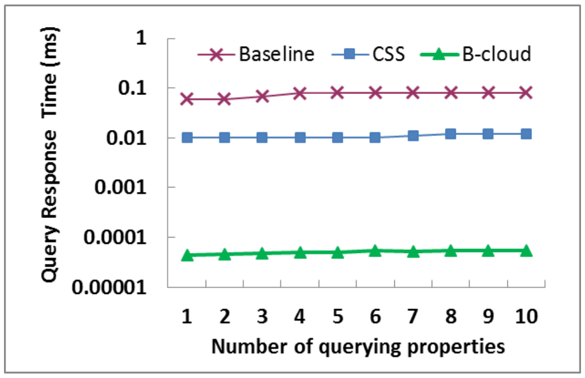


Fig. 10. Effect of the number of querying properties on service selection time (Exact Queries)

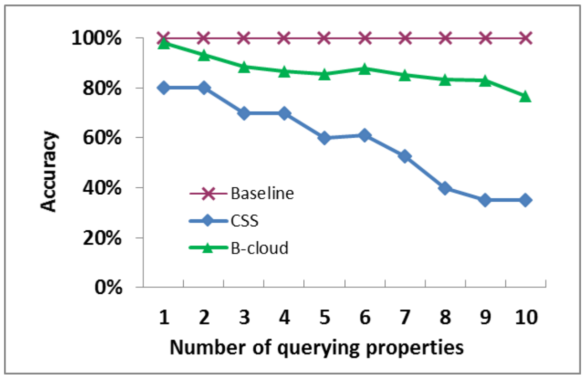


Fig. 11. Effect of the number of query properties on accuracy (Exact Queries)

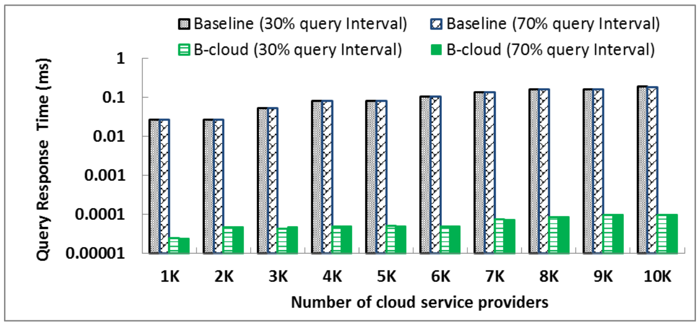


Fig. 12. Effect of the number of cloud services and the querying range on service selection time

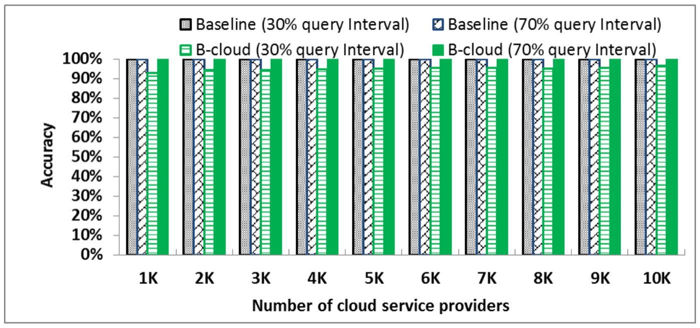


Fig. 13. Effect of the number of cloud services and the querying range on accuracy

the k-nearest neighbor is set to 10% of the number of cloud services. Regarding the B^{cloud}-tree, the more query properties, the slightly smaller the search range would be according to the query normalization algorithm in Section 5.1. However, the difference of the search range varied by the number of query properties may have too small impact on the processing time to be observed clearly.

In terms of accuracy of the query results, Figure 11 shows that the accuracy in both the B^{cloud}-tree and the CSS approach decreases with the increase of the query properties. This is because the more query properties, the fewer number of cloud services satisfying the query, and hence the higher chance of missing qualifying cloud services in the k-nearest neighbor search adopted by both approaches. However, we also observe that the B^{cloud}-tree always achieves higher accuracy than the CSS approach because the new encoding method in the B^{cloud}-tree gathers similar cloud services closer than that in the CSS approach and hence misses fewer qualifying cloud services.

6.2.2 *Performance of Interval Queries*

We now evaluate the performance of the interval queries which allow the users to specify a range of desired values for querying properties and the cloud broker will return 10 candidate cloud services that satisfy the query conditions best. Since the CSS algorithm does not support interval queries, we present the comparison results between the B^{cloud}-tree and the baseline approach. Specifically, we examine the effect of the number of cloud services, the query range of the properties, and the number of querying properties.

• **Effect of the Number of Cloud Services and Querying Intervals**:

For this round of experiments, we generate 100 interval queries with the following characteristics: (1) each query contain 5 (out of 10) randomly selected properties; (2) each query property is associated with a randomly generated interval that covers 30% (or 70%) of values in the corresponding domain respectively. Figure 12 reports the query processing time of the two approaches. As shown in the figure, our proposed B^{cloud}-tree is hundreds of times faster than the baseline approach in all cases. The reason is similar to that for the exact queries. The B^{cloud}-tree utilizes indexing techniques to target the potential candidate cloud services and significantly reduces the number of candidate cloud services that need to be examined compared to the baseline approach. Another important observation is that the query processing time of the two kinds of query intervals (30% or 70%) is quite similar in both approaches. It is expected that the baseline approach will perform the same for various kinds of query settings since it always needs to check all cloud services. As for the B^{cloud}-tree, the query processing time is determined mainly by the value of k in the k-nearest neighbor search. For interval queries that require 10 cloud services in the results, we set k to 10 as well which provides 30 candidates for refinement as k nearest neighbor search is conducted in three directions (as presented in Section 5.3).

Figure 13 shows the accuracy of the query results, whereby

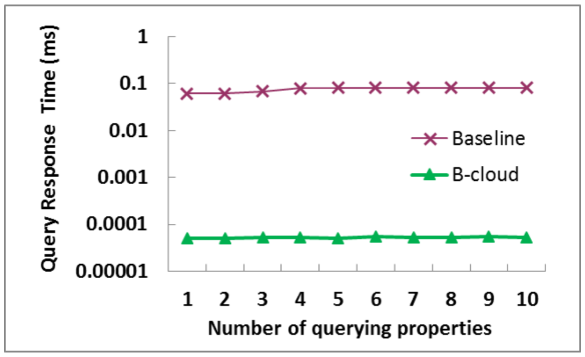


Fig. 14. Effect of the number of querying properties on service selection time (Interval Queries)

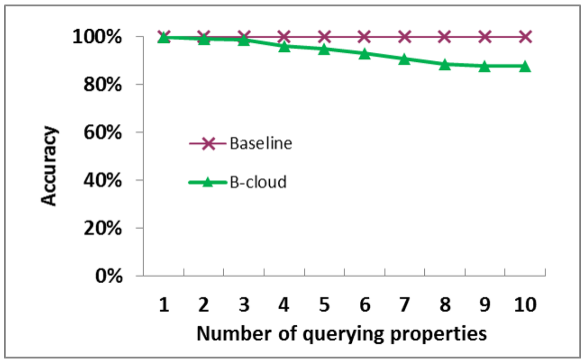


Fig. 15. Effect of the number of querying properties on accuracy (Interval Queries)

the baseline approach is considered 100% accurate as a comparison base. We can see that the B^{cloud}-tree achieves higher accuracy when the query interval becomes larger. This is because that the larger the query intervals, the more cloud services may satisfy the query conditions and hence improves the query accuracy. In addition, we also see a slight increase of the query accuracy in the B^{cloud}-tree approach with the increase of the number of cloud services which is due to the similar reason that more cloud services may satisfy the query conditions when the dataset is larger.

• **Effect of the Number of Querying Properties**:

We also evaluate the effect of the number of query properties by varying the query properties from 1 to 10 and keeping the same query interval to 30%. Figure 14 reports the average time to process the queries on 5000 cloud services. From the figure, we can see that the performance of both approaches are not affected by the number of query properties. As previously mentioned, the baseline approach's performance is independent of any type of queries due to the exhaustive search, while the performance of the B^{cloud}-tree is dominated by the number of query results to be returned to the end user which in term determines the value of the k in the k-nearest neighbor search.

On the other hand, the query accuracy of the B^{cloud}-tree is affected by the number of query properties as shown in Figure 15. Specifically, the query accuracy drops when the number of query properties increases. The possible reason is that the more query properties, the fewer satisfying cloud services exist and hence the higher chance to miss qualifying cloud services using the B^{cloud}-tree.

6.2.3 Update Performance

Finally, we evaluate the update performance of the B^{cloud}-tree by varying the dataset from 1K to 10K. We first construct the B^{cloud}-tree. Then we perform 100 updates of cloud services and count the average I/O cost (i.e., number of index node accesses) per update. Table 2 reports the height of the B^{cloud}-tree and the average update cost. We can see that the B^{cloud}-tree requires extremely few I/O accesses per update. The update cost is logarithm to the total number of the cloud services and is determined by the height of the tree. The update performance is consistent with its base structure, the B^{+}-tree.

TABLE 2
Update Performance of the B^{cloud}-tree

Number of Cloud Services	1K	5K	10K
Tree Height	2	3	3
Update I/Os	2.4	3.3	3.6

7 Conclusion

In this paper, we presents a brokerage-based architecture for cloud computing systems, as well as an efficient cloud service selection algorithm that provides a list of recommended cloud service providers to the cloud consumers based on their needs. In particular, we designed a novel indexing structure, namely the B^{cloud}-tree, to facilitate the arrangement and retrieval of the information about service providers. On top of the B^{cloud}-tree, we further developed an efficient service selection query algorithm that quickly retrieves desired service providers based on the users' service requests. Our experimental results show that the B^{cloud}-tree achieves significant improvement in terms of both efficiency and accuracy compared to our prior work. In the future, we plan to build an automated parser for extracting manifest variables of cloud service providers, and design strategies to provide the cloud users an opportunity to negotiate some terms of the service level agreements with the potential service providers.

References

[1] Cloud-fuser: Fuzzy ontology and {MCDM} based cloud service selection. *Future Generation Computer Systems*, 57:42–55, 2016.

[2] S. at TechTarget. Newservers: 2011 top cloud computing provider, 2011.

[3] M. B. and T. F. Decision-making in cloud computing environments: A cost and risk based approach. *nformation Systems Frontiers*, 14(4):87193, 2012.

[4] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[5] C. W. Chang, P. Liu, and J. J. Wu. Probability-based cloud storage providers selection algorithms with maximum availability. In *41st International Conference on Parallel Processing*, pages 199–208, 2012.

[6] CloudSwitch. http://www.cloudswitch.com/.

[7] M. Eggebrecht. Is cloud brokerage the next big thing? In *http://www.ciozone.com/index.php/Cloud-Computing/Is-Cloud-Brokerage-the-Next-Big-Thingu.html*.

[8] EQUINIX. http://www.equinix.com/industries/cloud-providers/.

[9] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione. Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory. *IEEE Transactions on Computers*, PP(99):1–1, 2015.

[10] W.-J. Fan, S.-L. Yang, H. Perros, and J. Pei. A multi-dimensional trust-aware cloud service selection mechanism based on evidential reasoning approach. *International Journal of Automation and Computing*, 12(2):208–219, 2014.

[11] N. Ghosh, S. K. Ghosh, and S. K. Das. Selcsp: A framework to facilitate selection of cloud service providers. *IEEE Transactions on Cloud Computing*, 3(1):66–79, 2015.

[12] S.-M. Han, M. M. Hassan, C.-W. Yoon, and E.-N. Huh. Efficient service recommendation system for cloud computing market. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, pages 839–845, 2009.

[13] J. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28:100–108, 1979.

[14] Q. He, J. Han, Y. Yang, J. Grundy, and H. Jin. Qos-driven service selection for multi-tenant saas. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 566–573, 2012.

[15] Ivan. Cloud business trend: Cloud brokerage. In *http://www.cloudbusinessreview.com/2011/04/26/cloud-business-trend-cloud-brokerage.html*.

[16] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30:364–397, 2005.

[17] F. Jrad, J. Tao, A. Streit, R. Knapper, and C. Flath. A utility-based approach for customised cloud service selection. *Int. J. Comput. Sci. Eng.*, 10(1/2), Jan. 2015.

[18] J. Li, A. C. Squicciarini, D. Lin, S. Sundareswaran, and C. Jia. Mmbcloud-tree: Authenticated index for verifiable cloud service selection. *IEEE Transactions on Dependable and Secure Computing*, pp(1):1, 2015.

[19] R. Miller. Cloud brokers: The next big opportunity? In *http://www.datacenterknowledge.com/archives/2009/07/27/cloud-brokers-the-next-big-opportunity*.

[20] D. K. Nguyen, F. Lelli, M. P. Papazoglou, and W. J. van den Heuvel. Issue in automatic combination of cloud services. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 487–493, 2012.

[21] C. C. T. . S. of the Cloud Survey. http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2015-state-cloud-survey.

[22] J. Orenstein. Spatial query processing in an object-oriented database system. In *Proc. ACM SIGMOD*, pages 326–336, 1986.

[23] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski. Introducing stratos: A cloud broker service. In *International Conference on Cloud Computing (CLOUD)*, pages 891–898, 2012.

[24] L. Qi, W. Dou, and J. Chen. Weighted principal component analysis-based service selection method for multimedia services in cloud. *Computing*, 98(1-2):195–214, 2016.

[25] L. Qu, Y. Wang, and M. A. Orgun. Cloud service selection based on the aggregation of user feedback and quantitative performance assessment. In *2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*, pages 152–159, 2013.

[26] RightScale. http://www.rightscale.com/.

[27] C. Security Alliance. Security guidance for critical areas of focus in cloud computing. In *http://searchcloudcomputing.techtarget.com/feature/Top-10-cloud-computing-providers-of-2011*.

[28] J. Siegel and J. Perdue. Cloud services measures for global use: The service measurement index (smi). In *2012 Annual SRII Global Conference*, pages 411–415, 2012.

[29] L. Sun, H. Dong, F. Hussain, O. Hussain, and E. Chang. Cloud service selection: State-of-the-art and future research directions. *Journal of Network and Computer Applications*, 45:134–150, 2015.

[30] S. Sundareswaran, A. C. Squicciarini, and D. Lin. A brokerage-based approach for cloud service selection. In *IEEE International Conference on Cloud Computing*, 2012.

[31] S. Taylor, A. Young, and J. Macaulay. Small businesses ride the cloud: Smb cloud watch - U.S. survey results.

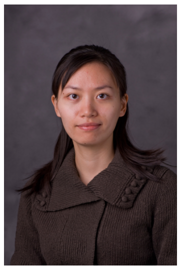
[32] Z. ur Rehman, F. K. Hussain, and O. K. Hussain. Towards multi-criteria cloud service selection. In *Proceedings of the Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2011, Seoul, Korea, June 30-July 02, 2011*, pages 44–48, 2011.

[33] X. Wang, J. Cao, and J. Wang. A dynamic cloud service selection strategy using adaptive learning agents. *Int. J. High Perform. Comput. Netw.*, 9(1/2):70–81, 2016.

[34] L. Xin and A. Datta. On trust guided collaboration among cloud service providers. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*, pages 1–8. IEEE, 2010.

[35] P. Xu and S. Tirthapura. A lower bound on proximity preservation by space filling curves. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1295–1305, 2012.

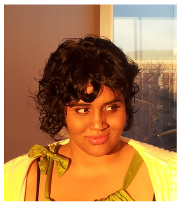
[36] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the distance: an efficient method to knn processing. In *International conference on Very large data bases*, pages 421–430, 2001.



Dan Lin is an associate professor at Missouri University of Science and Technology. She received the PhD degree in Computer Science from the National University of Singapore in 2007, and was a post doctoral research associate at Purdue University for two years. Her main research interests cover many areas in the fields of database systems, information security, cloud computing, and vehicular ad-hoc networks.

Anna Cinzia Squicciarini is an associate professor at the college of Information of Information Science and Technology, at the Pennsylvania State University. During the years of 2006-2007 she was a post doctoral research associate at Purdue University. Squicciarini's main interests include access control for distributed systems, privacy, security for Web 2.0 technologies and grid computing. Squicciarini earned her Ph.D. in Computer Science from the University of Milan, Italy, in 2006. Squicciarini is the author or co-author of more than 60 articles published in refereed journals, and in proceedings of international conferences and symposia. She is an IEEE member.

Venkata Nagarjuna Dondapati obtained his MS degree from Computer Science at Missouri University of Science and Technology. He received his Bachelors degree on Computer Science from Jawaharlal Nehru Technological University in 2012. His main research interests include Cloud Computing and Databases. He served as a Vice President for the CS IEEE branch at Missouri University of Science and Technology.



Smitha Sundareswaran received the bachelors degree in electronics and communications engineering in 2005 from Jawaharlal Nehru Technological University, Hyderabad, India. She received her PhD degree from the College of Information Sciences and Technology at the Pennsylvania State University. Her research interests include policy formulation and management for Distributed computing architectures.