A cloud brokerage architecture for efficient cloud service selection

Venkata Nagarjuna Dondapati

A CLOUD BROKERAGE ARCHITECTURE FOR EFFICIENT CLOUD SERVICE SELECTION

by

VENKATA NAGARJUNA DONDAPATI

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2014

Approved by

Dr. Dan Lin, Advisor

Dr. Wei Jiang

Dr. Zhaozheng Yin

ABSTRACT

The expanding cloud computing services offer great opportunities for consumers to find the best service and best pricing, which however raises new challenges on how to select the best service out of a huge pool. It is obvious time-consuming for consumers to collect the necessary information and analyze all service providers to make the decision. It is also a highly demanding task from a computational perspective, because the same computations may be conducted repeatedly by multiple consumers who have similar requirements. In this paper, we propose a novel brokerage-based architecture in the cloud, where the cloud brokers is responsible for the service selection. In particular, we design a unique indexing technique for managing the information of a large number of cloud service providers. We then develop an efficient service selection algorithm that recommend potential cloud service providers to the cloud consumers. We carry out extensive experimental studies on real and synthetic cloud data, and demonstrate a significant performance improvement over the existing approaches.

ACKNOWLEDGEMENTS

I owe a debt of gratitude to all those who have helped me with this thesis. First of all, I would like to thank my advisor Dr. Dan Lin, who gave me an opportunity to work on this research. Her suggestions and encouragement carried me through difficult times. Her valuable feedback contributed greatly to this thesis. Secondly, I would also like to express my gratitude to Dr. Wei Jiang and Dr. Zhaozheng Yin for serving on my thesis committee and for taking time to review this work.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

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 [27]. While this has fueled the large growth in cloud services, the growing number of cloud services make it difficult for the potential users to weigh and decide which options suit their needs best. Without external helps, cloud service consumers must manage payments, governance, data movement, customization and enrichment, with providers and their services. This can rapidly become a difficult task [3], [8]. There is clearly a need of an additional computing layer on top the base service provisioning to enable tasks such as discovery, mediation, and monitoring. This additional layer of computing is referred to as a brokerage system.

In general, a cloud broker is an intermediary between users and service providers, in charge of aggregating, integrating or customizing cloud services [7], [15]. One of the best known brokers is CloudSwitch [6], established in 2008 with service for only Amazon EC2. It has the ability to provide federated services on demand and make the cloud a secure and seamless extension of the enterprise data center. RightScale [21] is another cloud broker that offers a cloud management platform to facilitate deployment and management of applications across multiple clouds. Recently, Dell has also claimed an interest in cloud services brokerage, and has been working in partnership with VMWare to provide new brokerage infrastructure and services [25].

Among various responsibilities that a cloud broker can carry, the very first important task could be to help cloud consumers select the appropriate cloud service providers (CSPs) that satisfy their requirements. The selection of CSPs requires addressing a number of interesting questions raised by the unique characteristics of the cloud computing environments. First, cloud services are enriched with non-standardized representation of the cloud providers' properties. Also, the Service Level Agreements (SLAs) of cloud providers often vary in format and content. Therefore, existing web service selection algorithms [13], [16], [24] cannot be directly applied to the cloud domain. Secondly, a cloud user may have a service requirement that cannot be fulfilled by any single CSP, thus requiring an aggregation of CSPs. Aggregating CSPs is nontrivial, as cloud service providers build complex relationships with one another through subcontracting mechanisms. For example, when aggregating service providers that rely on the same contractor for storage space, the broker should avoid overextending the actual storage space.

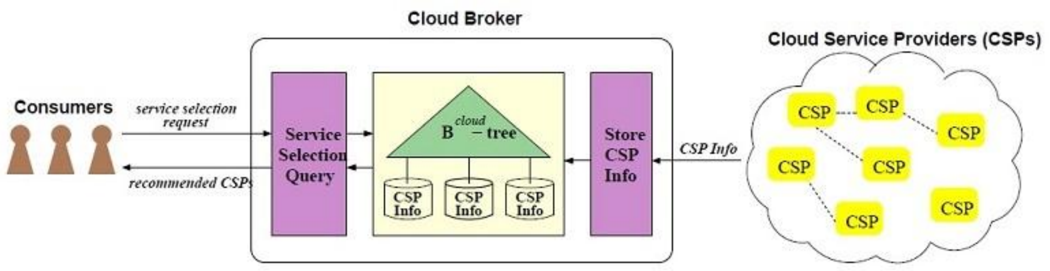


Figure 1.1. An Overview of the Cloud Brokerage Model

Bearing these challenges in mind, we propose a comprehensive brokerage-based architecture to support cloud service selection. The overall architecture is illustrated in Figure 1. The cloud broker has a collection of CSPs' profiles and built an efficient index structure (namely the B^{cloud}-tree) for retrieval of desired CSPs as well as management of profile 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 queries our proposed B^{cloud}-tree to identify the CSPs that satisfy consumers' requirement.

A preliminary version of this work appears in [26], where we presented the basic idea of the cloud brokerage and an approach called CSS (Cloud Service Selection). In this paper, we present a whole new approach that is much more efficient and accurate than the CSS approach. Specifically, our new major contributions include the following: (i) we define a more general type of service selection query that allow users to specify service selection requests that contain intervals of desired values (e.g., a price range) whereas the existing approach can only deal with exact match (e.g., a single price value); (ii) we propose a new indexing structure, the Bcloudtree, to manage the CSPs' information, along with a new query algorithm that achieves better efficiency and query accuracy compared to the CSS approach; (iii) we also carried out extensive experiments that compare our two approaches and a baseline approach using the data collected from top cloud providers [22].

2. RELATED WORKS

There have been some high level discussion on service provider selection and brokerage-based frameworks in the cloud [7], [11], [15], [22]. For example, Gartner [8] introduced different types of cloud brokerage including arbitrage, aggregation and intermediation. Others [7], [15] discussed possible responsibilities of cloud brokers such as service monitoring and service aggregation. However, to the best of our knowledge, there is not any existing work that provides a specific solution to the cloud service selection problem. The only academic effort in this direction is from Buya et al. [4] who provide a general description of the key role of cloud broker services for a market-oriented cloud service. In addition, although not specific to cloud brokers, Xin et al. [28] have considered collaborative protocols among cloud service providers for resource sharing. In particular, Xin et al. use trust as the only criteria to select service partners. As we show next, our work considers a much wider range of factors during the service selection. Han et al. [9] describe a time recommendation system for cloud computing [**?**], which employs static approach to provide a ranking of available cloud providers. The recommendation is done based on QoS and Virtual Machine (VM) platform factors of difference CSPs. Our approach considers multiple factors, and provides a ranking based on variable user queries expressed as a function of these factors. Recently, Pawluk et al. [19] proposed a broker service to facilitate resource allocation decisions (RAD), and formulate the solution as a multi criteria optimization problem. This RAD problem has been earlier studied by Hosseini et al. [14], who adopted a static approach without considering the possible change of the resource topology over time. The research on the RAD problem is orthogonal to ours.

While cloud brokerage and selection is a relatively unexplored territory, service selection problem have been studied in great depth in the context of Web services. To date, most of works on Web service selection are based on QoS (Quality of Service) [20]. For instance, Kalepu et al [13] propose an objective measure of QoS based on the extent up to which the Web service meets its service level agreements. Paolucci et al. [18] proposed solution based on DAML-S, a DAML-based language for service description, and then they perform a semantic matching between the request and a service advertisement. Zeng et al. [30] developed a middleware that composes multiple Web services to meet a single user's need. The goal is to maximize user satisfaction while satisfying user and service provider constraints at the same time. Bentallah et al. [2] also propose various algorithms for Web service composition including fast composition, scalable composition and distributed composition. Unlike works on Web service domain, our work is unique in several aspects. First, our service selection approach is based on efficient indexing and querying of service provider information. Such techniques have never been leveraged in Web service domain. Second, we deal with much more complicated properties and relationships pertaining to the service providers due to the complexity of the cloud.

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 CSPs to the user.

The realization of 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 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 CSPs against the user's requirements.

To address the first challenge, we leverage indexing techniques that groups CSPs according to the similarity of their properties. Then, we develop efficient query algorithms to identify CSPs 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 CSPs considered in our work, and give the formal definition of a user's service request.

3.1 PROPERTIES OF CSPS

This work focuses on ten properties most commonly acknowledged as relevant properties of CSPs. We discuss the approach used to identify these properties in Section 6.1.

- Service Type (p1). This denotes the type of service provided, which could vary between service on-demand, and reserved instances, or refer to specialized services such as custom IPs in case of Amazon or caching in case of Windows Azure.
- Security (p2). This denotes the level of security and/or privacy that can be achieved using the various options provided by the cloud provider. 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 [23], 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.

- Quality of service (QoS, p3). QoS is determined by the cloud broker which analyze the collected information about service providers over time, and ratings provided by other vendors. It is briefly represented using value high, low or medium.
- Measurement units (p4). 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 (p5). This indicates how long a service is reserved for. For example, the price could be charged per hour, per month or per year.
- Instance sizes (p6). 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 (p7). This indicates that the operating system could be Linux or Windows.
- Pricing (p8). This is the actual price for the usage of the cloud service.
- Pricing sensitivity to regions (p9). This denotes if the price varies by region.
- Subcontractors (p10). 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 $Q = <(QP_1 : V_1), (QP_2 : V_2), . . . , (QP_k : V_k)>$, where $k \leq 10$, QP_i $(1 \leq i \leq k)$ is the property that the user requests the service provider 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 $Q = \{(QP_1 : I_1), (QP_2 : I_2), . . . , (QP_k : I_k)\}$, where $k \leq 10$, QP_i $(1 \leq i \leq k)$ is the property that the user requests the service provider 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 simplified GUI with check boxes and drop down lists 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 CSP that matches the most number of query conditions. For the interval query, the cloud broker aims to return m CSPs 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 CSPs rather than a very large number (e.g., 100) of CSPs that would be hard to choose from.

3.3 EXISTING WORK

Since the new approach presented in this paper is compared to the existing work, we review the existing work in more details as follows. In [24], 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 multidimensional index, called iDistance [10] 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 e1, ..., e10) where ei is the encoding of property pi, we further generate the integrated encoding by concatenating the bits representing the service type with the XOR-ed results of the remaining property encodings.

$$Ecsp = e1 || (e2 \oplus e3 \oplus e4 \oplus e5 \oplus e6 \oplus e7 \oplus e8 \oplus e9 \oplus e10) \qquad (1)$$

After obtaining the encoding for all the CSPs, we employ the k-means algorithm [8] to cluster the CSPs based on the Hamming distance between their encodings Ecsp. Then we leverage the idea of the iDistance [10], [27] to generate the indexing key Keycsp for each CSP.

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

In Equation 2, Eck 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 Dh 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 p1, pi, pj, pk have been encoded:

*CSP*1*(*e1*=0*, ei*=1*, ej = 5, ek = 9*)*

*CSP*2*(*e1*=0*, ei*=9*, ej = 1, ek = 5*)*

*CSP*2*(*e1*=0*, ei*=9*, ej = 5, ek = 1*)*

$$Ecsp1 = 0||(1 \oplus 5 \oplus 9)$$

$$Ecsp2 = 0||(9 \oplus 1 \oplus 5)$$

$$Ecsp3 = 0||(9 \oplus 5 \oplus 1)$$

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 CSP1 and

CSP2 differ in all three properties considered, and CSP1 and CSP3 differ in two properties out of three.

In a 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 new approach also supports the interval query (Definition 3.2) whereas the existing work does not.

4. MANAGING INFORMATION OF CLOUD SERVICE PROVIDERS

In face of a large number of cloud service providers, it is important to design an efficient data structure to facilitate information management and retrieval. The specific design goals are the following.

- If the broker needs to update a CSP's information such as change of properties and available resources, the broker should be able to quickly locate where the CSP 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 CSPs rather than checking all CSPs against the user's request.

To achieve the above goals, we propose a Bcloud-tree which provides quick access to individual CSPs and also groups CSPs with similar properties to facilitate queries. In what follows, we elaborate how to construct the Bcloud-tree to realize the design goals.

4.1 THE STRUCTURE OF B^{cloud} - TREE

The B^{cloud}-tree is a multi-level balance tree as shown in Figure 3. 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 CSP: $\{\kappa, ID, p_1, p_2, ..., p_{10}\}$, where κ is the indexing key of the CSP whose identity is ID, and p_i is the ith property of the CSP ($0 \leq i \leq 10$). The internal nodes of the B^{cloud}-tree serve as the search directory which contain indexing keys and pointers to children nodes.

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.

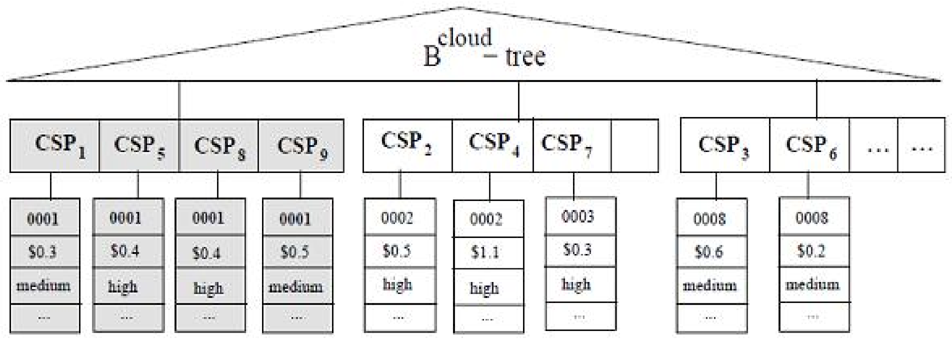


Figure 4.1. An Example of B^{cloud} - Tree

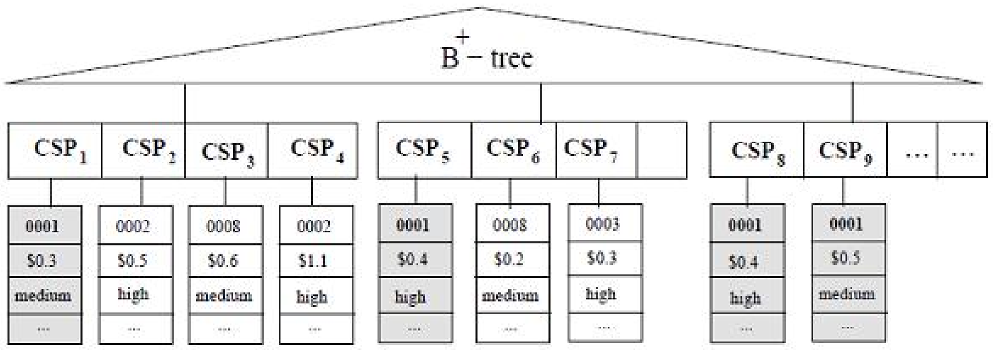


Figure 4.2. An Example of B^{+} - Tree

Moreover, we can also leverage the same suite of efficient B+-tree's algorithms to search, insert, delete and update the information of a CSP (i.e., an entry in a leaf node).

For example, to search the properties of a CSP 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 CSP's indexing key. The search just needs to access h nodes to locate any given CSP where h is the height of the tree.

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 CSPs. How the CSPs are stored has a great impact on the efficiency of answering a user's service request. Reconsider the example query Q_1 = {(ServiceType:0001), (Cost:\$0.6/min)} in the previous section. The B+-tree that indexes CSPs simply based on their IDs may store CSPs as shown in Figure 3. We can observe that to answer the user's request, the Bcloud-tree (Figure **??**) 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 CSPs that offer service "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 CSPs with similar properties receive nearby indexing keys. We present the detailed algorithm of key generation in the following subsection.

4.2 THE CONSTRUCTION OF 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 service providers, and then convert them into a single key value. It is obviously time consuming to compare every pair of CSPs to identify which of them are similar to each other. This is also impractical concerning the changing of CSP's properties, joining of new CSPs or leaving of old CSPs. Therefore, we propose a hash style key generation method, which allows us to directly compute the indexing key for any given CSP based on its properties and ensures that most of the resulting keys have the following properties: the more similar the CSPs 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 CSPs 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 CSP 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 CSPs 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 CSP'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. For example, the property "quality of service" can be described using "high", "medium", "poor". Correspondingly, we convert it to numbers "3"(high), "2"(medium), "1"(poor).
- **Relationship property**: This refers to the specifical property "Subcontractor" (P_{10}) which describes the relationship among the CSPs 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 CSP 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 CSP. Recall that the indexing key should ensure that CSPs 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 [15]. 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 CSPs, 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 deviation does not significantly affect the accuracy.

The steps of generating the indexing keys are the following.

1. Given a service provider $CSP_1(p_1, ..., p_{10})$ and its property encodings denoted as $CSP_1(e_1, ..., e_{10})$, 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, we 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.
- The last step is to convert Z into a decimal value, which would be the indexing key of the CSP1: $Z \rightarrow \kappa$.

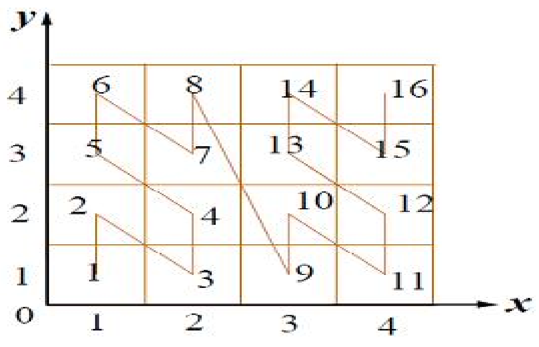


Figure 4.3. An Example of Z - curve

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, 50cents/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 CSP1 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 Z-value ‘0000010001111000’. Its corresponding decimal value is 1144, which will be used as the indexing key for this CSP.

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

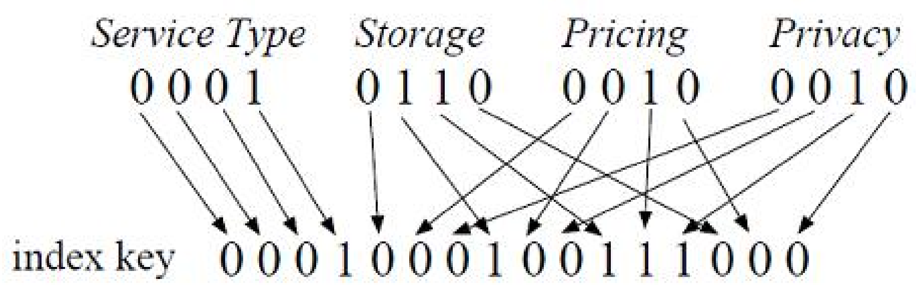


Figure 4.4. Generating the Index Key Through Interleaving

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 CSPs who may satisfy the query. Based on the query encodings, we then search the B−tree to locate the candidate CSPs. The last step is to further exam the properties of the candidate CSPs and their relationship to find the best combination of service providers that addresses the user's needs. We detail the key 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 /∈ 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 Q2 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 CSPs 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 CSP, 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 a subset of CSP's properties and ranges of possible values for the querying properties. The task is to find the indexing keys of the CSPs 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 CSP. Take the previously normalized interval query Q'$_2$ as an example. All of the following CSPs satisfy the query:

CSP2: **(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}$:[000])>. CSP10: **($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])>. CSp13: **(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 CSPs satisfying the query may have very different property values for non-querying properties.

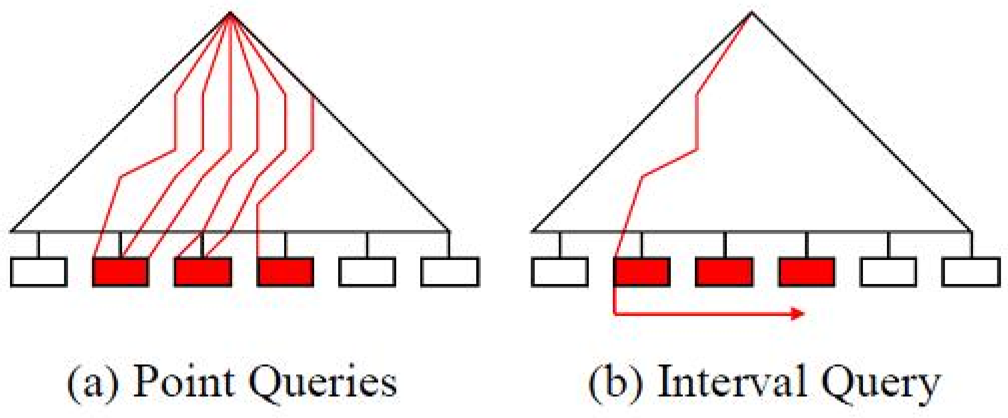


Figure 5.1. Point Query vs. Interval Query for Service Selection

Instead of computing the indexing key for each possible combination, we propose to compute the boundaries of the indexing keys belonging to all the CSPs 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 CSP 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 CSPs accesses much fewer tree nodes as shown in Figure 6(b), and hence will be more efficient.

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 $Q=(QP_1:[V^l_1 , V^u_2], ..., QP_k:[V^l_k , V^u_k], NP_1 : [[V^l_{k+1}, V^u_{k+1}], ..., NP_n : [V^l_{10}, V^u_{10}])$ be a normalized interval query, where V^l_j and V^u_j denote the lower and upper bounds of the interval respectively. Let κ^l be the indexing key computed from $(QP_1:V^l_1 , ..., QP_k:V^l_k, NP_1 : V^l_{k+1},..., NP_n : V^l_{10})$, and let κ^u be the indexing key computed from $(QP_1:V^u_1 , ..., QP_k:V^u_k , NP_1 : V^u_{k+1}, ..., NP_n : V^u_{10})$. Then, for any $CSP=(p_1,.., p_{10})$ whose $p_j \in [V^l_j , V^u_j]$ (for all $1 \leq j \leq 10$), its indexing key κ_{csp} will be in the range: $\kappa^l \leq \kappa_{csp} \leq \kappa^u$.

The range of the one-dimensional query interval obtained from Theorem 5.1 may contain false positives which are the CSPs whose indexing keys falling 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 CSP has only two properties and hence each CSP 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 CSPs 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 CSPs 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).

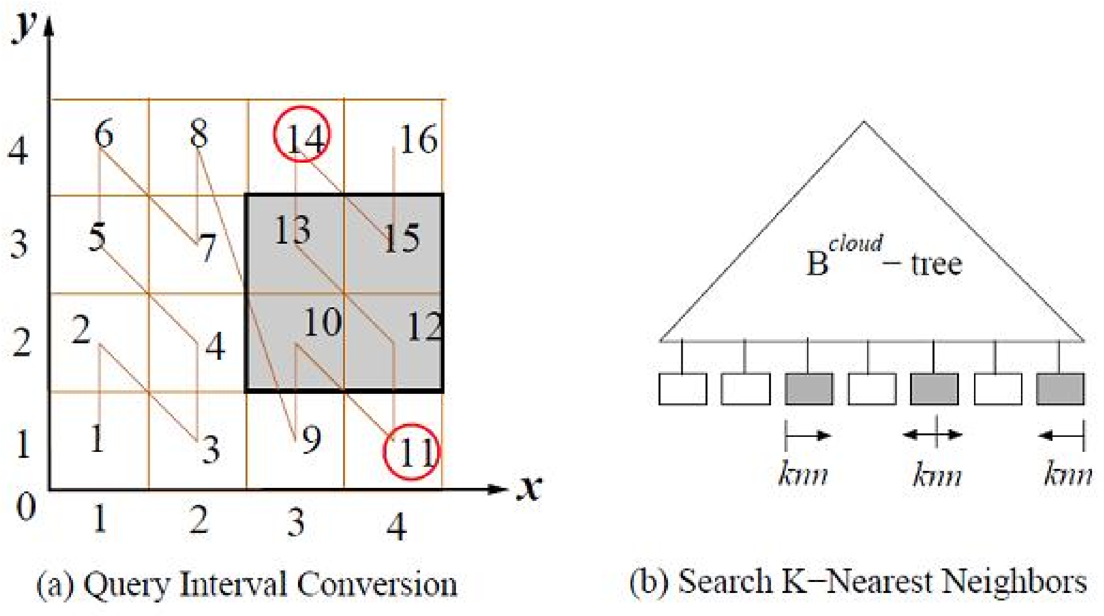


Figure 5.2. Query Encoding

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

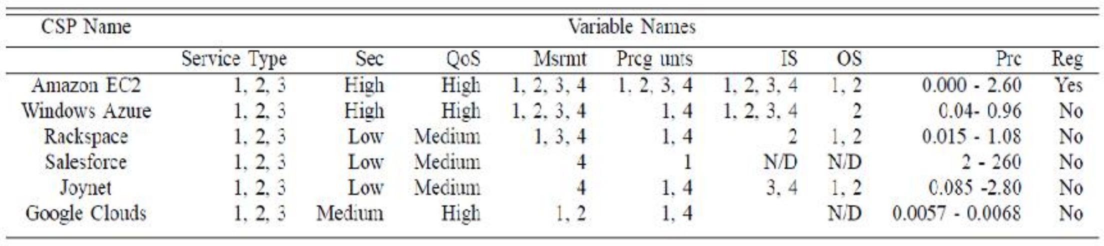
From the obtained candidate service providers, the refinement phase further evaluates the actual properties of the candidate service providers and finds the service providers or the combinations of service providers that satisfy the query requirements.

The first step is to check whether the properties of a candidate service provider fully satisfies the query requirements. If so, the candidate service provider will be added to the query result. After examining all candidate service providers, if the result set is not empty, we will directly return the CSPs in the result set to the users. If the result set is still empty, we will pursue the second step to take into account combined service provided by multiple CSPs.

To find the combination of the CSPs to satisfy the query requirements, we employ a greedy algorithm as follows. First, we sort the candidate CSPs based on each querying properties. Suppose that there are n querying properties. We obtain n sorted lists of candidate CSPs. If the querying properties are given in a decreasing order of importance in the query, we will start from the sorted list of the first (i.e., the most important) querying property. We first select the service provider on top of the first sorted list. We remove the satisfied querying properties from subsequent process, and adjust partially satisfied querying properties. For example, if the user requests 20GB of storage space, while the selected service provider only has 5GB available, we adjust the querying property on storage space to 15GB (=20GB-5GB) and look for more service providers.

As long as there are unsatisfied querying properties, we continue the selection of service providers by looking into the list of the next important unsatisfied property, and repeat the process. The selection process stops when all querying properties are satisfied. This set of service providers 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.

Table 5.1. Cloud Service Provider Properties and Domains



CSP 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

The possibility of a collision between service providers 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 service providers 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 service providers CSP_1 and CSP_2, who both lease 50 GB of storage space from a subcontractor CSP_3. CSP_3 can guarantee both CSP_1 and CSP_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, CSP_1 and CSP_2 can fulfill this requirement together by depending on CSP_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 service providers 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 service providers in the same compound service. If any two service providers 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.

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 DATA SETS

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 [20]. 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 [21]. 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. 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 the existing work, the CSS algorithm [], 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.

$$\text{Accuracy} = N_q / N_{base}$$

6.2.1 Performance of Exact Queries : First, we evaluate the performance of exact queries by varying the number of CSPs and the number of querying properties.

- **Effect of the Number of CSPs**

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 CSPs ranging from 1000 to 10,000. Figure 8 shows the query processing time of the three approaches: the B^{cloud}-tree, the CSS algorithm and the baseline approach (the results are plotted on a log scale). It is not surprising to see that the baseline approach is the slowest since it needs to check all the CSPs for each service selection query. In contrast, our proposed new approach, the B^{cloud}-tree, performs best among the three. Specifically, the B^{cloud}-tree is about 100 times faster than the baseline approach and more than 10 times faster than the existing CSS approach when the number of CSPs is more than 5000. Such good performance of the B^{cloud}-tree is attributed to the better grouping of similar CSPs using the new encoding method as described in Section 4.2. As similar CSPs 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 CSPs) in order to find answers of reasonable accuracy, because the encoding used by the CSS may give totally different CSPs the same encoding as shown in Example 2.1. The accuracy of the query results of the three 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. 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 CSPs. This is because the more CSPs, the higher chance of having more CSPs 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).

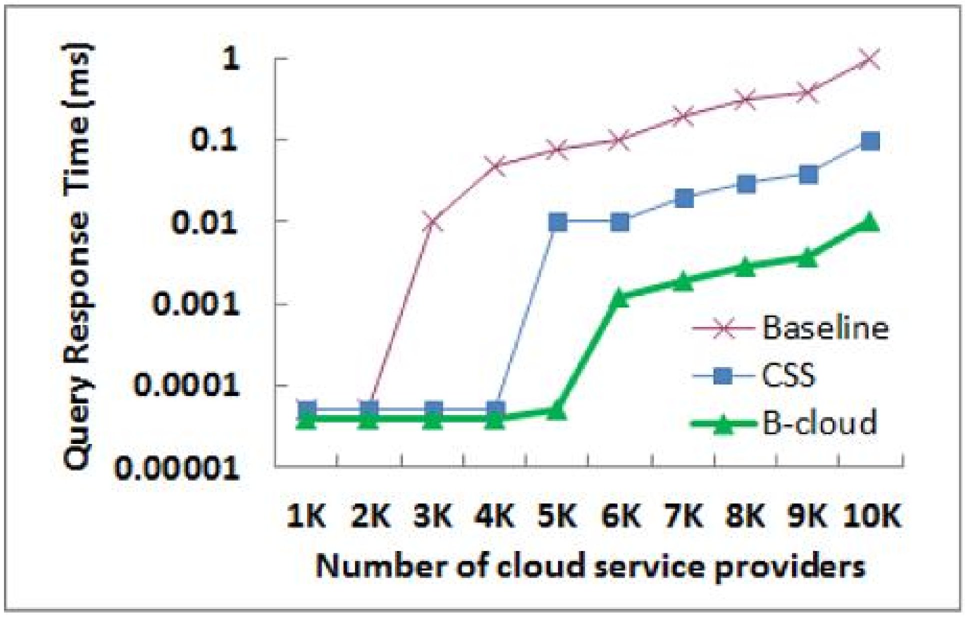


Figure 6.1. Effect of the Number of CSPs on Service Selection Time (Exact Queries)

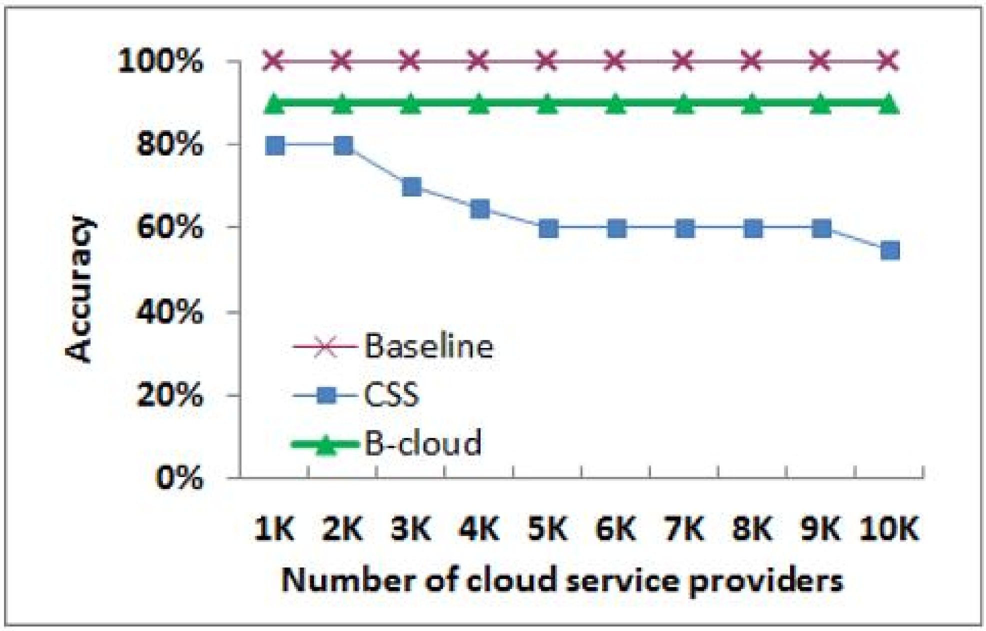


Figure 6.2. Effect of the Number of CSPs on Accuracy (Exact Queries)

- **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 CSPs. As for the CSS approach, its query processing time is dominated by the total number of CSPs since the value of k in the k-nearest neighbor is set to 10% of the number of CSPs. 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.

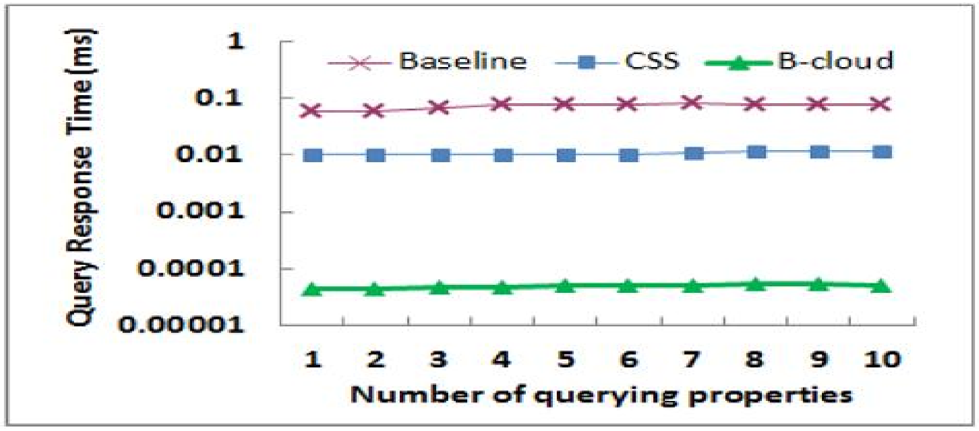


Figure 6.3. Effect of the Number of Querying Properties on Service Selection Time (Exact Queries)

In terms of accuracy of the query results, Figure 11 shows that the accuracy in both the Bcloud-tree and the CSS approach decreases with the increase of the query properties. This is because the more query properties, the fewer number of CSPs satisfying the query, and hence the higher chance of missing qualifying CSPs 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 CSPs closer than that in the CSS approach and hence misses fewer qualifying CSPs.

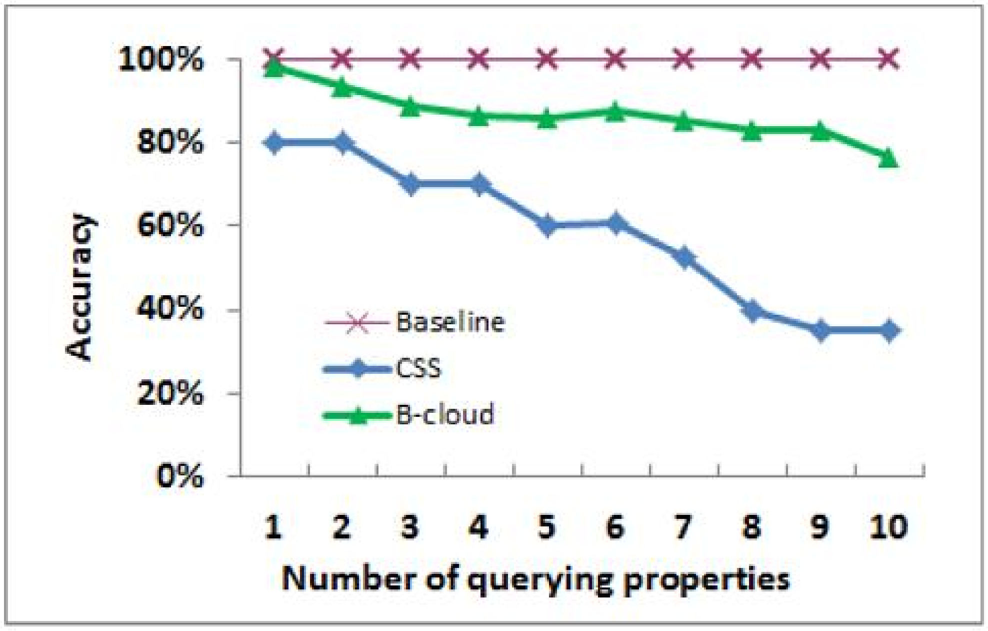


Figure 6.4. Effect of the Number of Query Properties on Accuracy (Exact Queries)

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 CSPs 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 CSPs, the query range of the properties, and the number of querying properties.

- **Effect of the Number of CSPs 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.

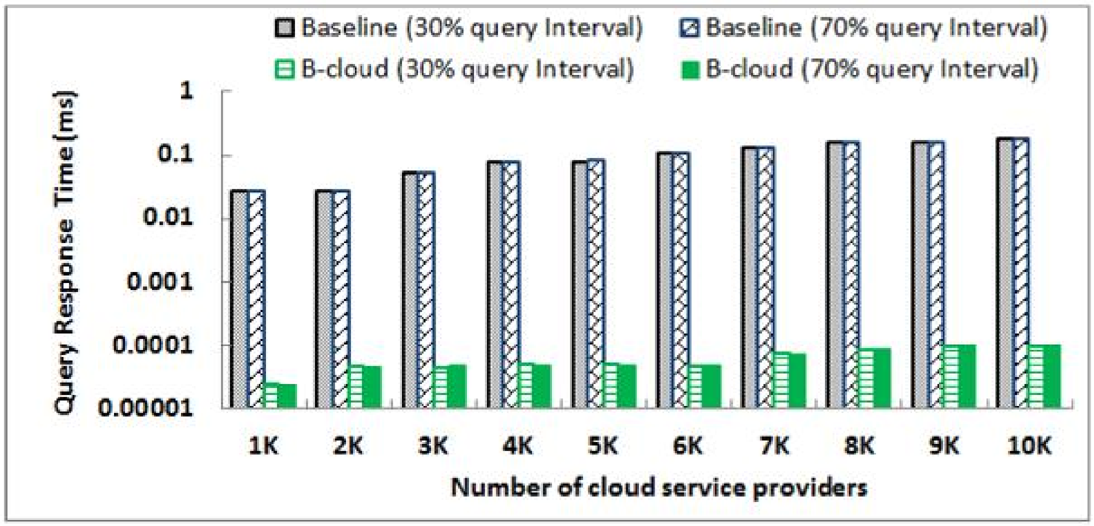


Figure 6.5. Effect of the Number of CSPs and the Querying Range on Service Selection Time

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 CSPs and significantly reduces the number of candidate CSPs 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 CSPs. 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 CSPs 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).

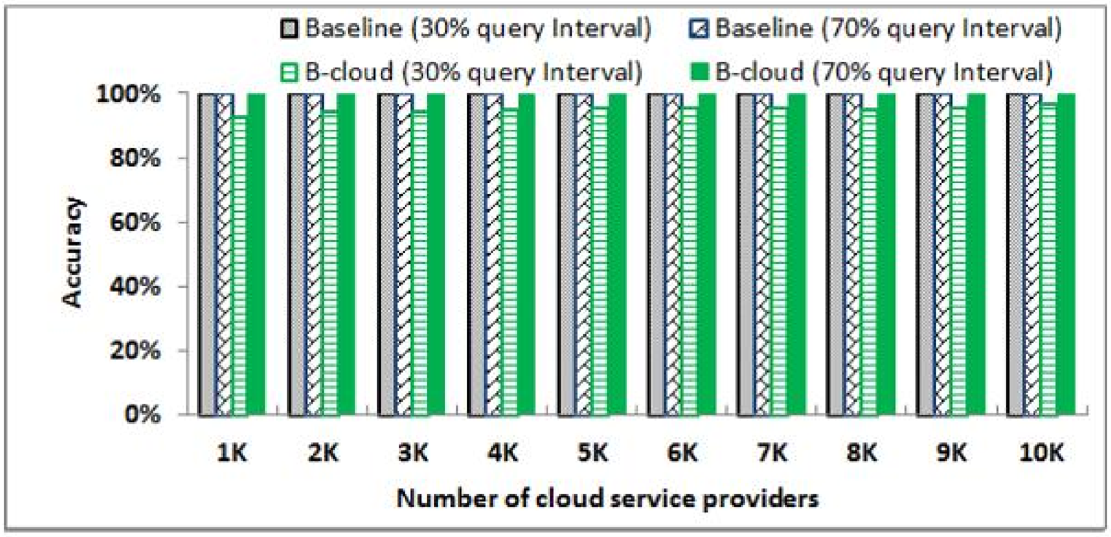


Figure 6.6. Effect of the Number of CSPs and the Querying Range on Accuracy

Figure 13 shows the accuracy of the query results, whereby 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 CSPs 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 CSPs which is due to the similar reason that more CSPs 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 CSPs. 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.

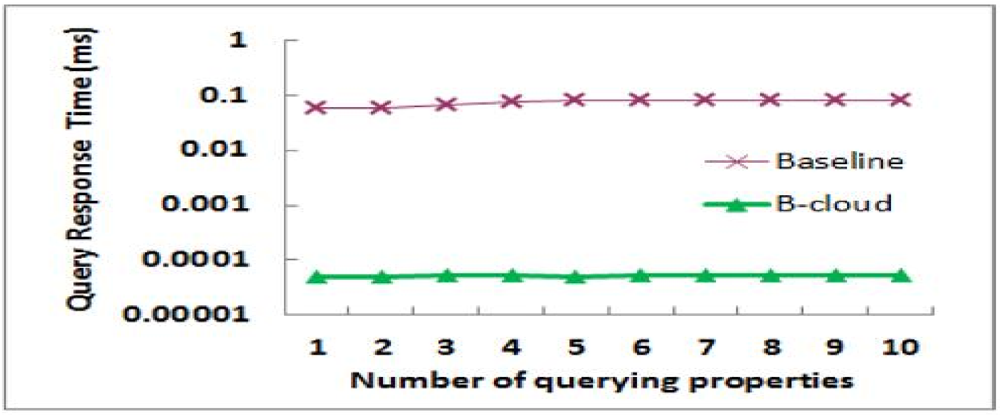


Figure 6.7. Effect of the Number of Querying Properties on Service Selection Time

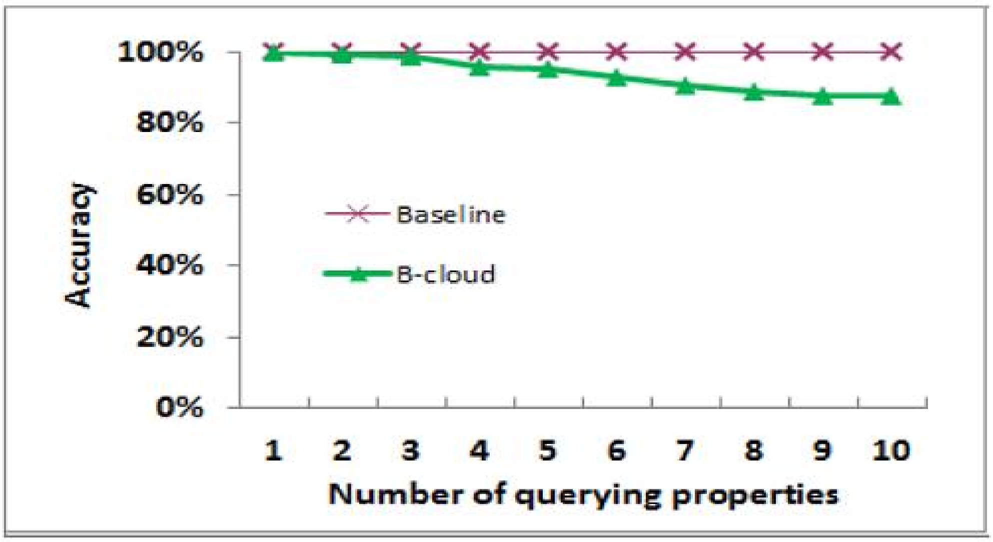


Figure 6.8. Effect of the Number of Querying Properties on Accuracy (Interval Queries)

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 CSPs exist and hence the higher chance to miss qualifying CSPs using the B^{cloud}-tree.

7. CONCLUSION

In this thesis, I presented 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, I 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, I further developed an efficient service selection query algorithm that quickly retrieves desired service providers based on the users' service requests. The 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, I 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.

BIBLIOGRAPHY

1. B. Benatallah, M. Dumas, Q. Sheng, and A. Ngu. Declarative composition and peer-to-peer provisioning of dynamic web services. In Proceedings on 18th International Conference on Data Engineering, pages 297–308. IEEE, 2002.

2. J. Burt. Gartner predicts rise of cloud service brokerages. In http://www.eweek.com/c/a/Cloud-Computing/Gartner-Predict-Rise-of-Cloud-Service-Brokerages-759833/.

3. 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.

4. CloudSwitch. http://www.cloudswitch.com/.

5. 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.

6. Gartner. Cloud services brokerages: The dawn of the next intermediation age. In http://www.gartner.com/technology/research/cloudcomputing/cloud-services-brokerage.jsp.

7. 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.

8. 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.

9. Ivan. Cloud business trend: Cloud brokerage. In http://www.cloudbusinessreview.com/2011/04/26/cloud-business-trendcloud-brokerage.html.

10. 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.

11. S. Kalepu, S. Krishnaswamy, and S. Loke. Verity: a qos metric for selecting web services and providers. In Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on, pages 131–139. IEEE, 2003.

12. A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. Teregowda. Decision support tools for cloud migration in the enterprise. In IEEE International Conference on Cloud Computing (CLOUD), pages 541–548, 2011.

13. R. Miller. Cloud brokers: The next big opportunity? In http://www.datacenterknowledge.com/archives/2009/07/27/cloudbrokers-the-next-big-opportunity.

14. A. Mondal, K. Yadav, and S. Madria. Ecobroker: An economic incentive-based brokerage model for efficiently handling multiple-item queries to improve data availability via replication in mobile-p2p networks. Databases in Networked Information Systems, pages 274–283, 2010.

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

16. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. The Semantic WebISWC 2002, pages 333–347, 2002.

17. 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.

18. J. Rao and X. Su. A survey of automated web service composition methods. Semantic Web Services and Web Process Composition, pages 43–54, 2005.

19. RightScale. http://www.rightscale.com/.

20. SearchCloudComputing at TechTarget. Newservers: 2011 top cloud computing provider, 2011. http://searchcloudcomputing.techtarget.com/feature/Top-10-cloudcomputing-providers.

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

22. N. Sriharee, T. Senivongse, K. Verma, and A. Sheth. On using ws-policy, ontology, and rule reasoning to discover web services. Intelligence in Communication Systems, pages 246–255, 2004.

23. J. Stickeleather. Cloud service brokerage. In http://en.community.dell.com/dell-blogs/enterprise/b/itexecutive/archive/2011/02/22/cloud-service-brokerage.aspx.

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

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

26. 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.

27. 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.

28. L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. IEEE Transactions on Software Engineering, 30(5):311–327, 2004.

VITA

Venkata Nagarjuna Dondapati was born in India. In May 2012, he received his B.S. in Information Technology from Jawaharlal Nehru Technological University, Hyderabad. He received his M.S. degree in Computer Science from Missouri University of Science and Technology, Rolla in May 2014. He worked as an intern in a few organizations during his Masters program.

Venkata Nagarjuna Dondapati has been a member of the Institute of Electrical and Electronics Engineers (IEEE) since 2013. He had served as the vice - president for the CS - IEEE branch at Missouri University of Science and Technology during the academic year 2013 - 2014.