International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 05 Issue: 10 | Oct - 2021 ISSN: 2582-3930

# Brokerage-Based Approach for Cloud Service Selection

Johnson Crastha b
*Information Sciences & Technology*
*Alva's institute of engineering and technology*

Chinmayi M k
*Information Sciences & Technology*
*Alva's institute of engineering and technology*

Dipthi palai
*Information Sciences & Technology*
*Alva's institute of engineering and technology*

***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 the huge pool. It is time-consuming for consumers to collect the necessary information and analyze all service providers to make the decision. This 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. Therefore, in this paper, we propose a novel brokerage-based architecture in the Cloud, where the Cloud brokers are 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 efficient service selection algorithms that rank potential service providers and aggregate them if necessary. We prove the efficiency and effectiveness of our approach through an experimental study with real and synthetic Cloud data.**

## I. Introduction

Cloud services offer an elastic and scalable IT task force in terms of storage space and computing capabilities which are essential to most business owners, especially small and medium-sized businesses [22]. 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 requirements the best. There is a need for an additional computing layer on top of 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.

Analogous to a stockbroker, a Cloud broker is essentially an intermediary between users and service providers, which helps the users choose services tailored to their needs. The importance of such a brokerage service is stressed by Gartner [3], [8] who defined different types of Cloud brokerage, including arbitrage, aggregation, and intermediation. Similarly, other recent work [7], [15] has acknowledged the increasingly important role of Cloud brokers and their multiple responsibilities ranging from service aggregation to monitoring. Companies like Dell have recently claimed an interest in Cloud services brokering, and have been working in partnership with VMWare to push out the same [21]. As acknowledged by this body of work, the first step for a broker to fulfill these responsibilities is to select the appropriate cloud service providers based on the user's requirements, which is the focus of this paper.

The selection of Cloud providers requires addressing several interesting questions raised by the unique characteristics of the Cloud computing environments. First, Cloud services may seem to resemble but are very different from Web services. For example, there is no standardized representation of the Cloud providers' properties. Also, the Service Level Agreements (SLAs) of Cloud providers often vary in format and content. Therefore, Web service selection algorithms [12], [16] cannot be directly applied to the Cloud domain. Secondly, a Cloud user may have a service requirement that cannot be fulfilled by any single service provider, thus requiring aggregation of service providers. Aggregating service providers is very challenging in the Cloud due to complex relationships among Cloud service providers that are built via subcontracting. For example, when aggregating service providers that rely on the same contractor for storage space, we should be careful to avoid overextending the actual storage space.

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. In particular, we propose an efficient indexing structure called the CSP (Cloud Service Provider) index, to manage the potentially large number of service providers. The CSP-index is built based on a novel encoding technique that captures similarity among various properties of service providers. With the aid of the CSP-index, we further design the service selection algorithm that considers aggregation of services and provides rankings of potential service providers. To evaluate our approach, we have collected real data from the top 10 Cloud providers listed by SearchCloudComputing in [19]. Our experimental study demonstrates both efficiency and effectiveness of our approach.

The remaining of the paper is organized as follows. Section II reviews related work. Section III describes the brokerage-based service selection architecture and data structure. Section IV presents the service selection algorithms. Section V reports experimental results. Finally, Section VI concludes the paper.

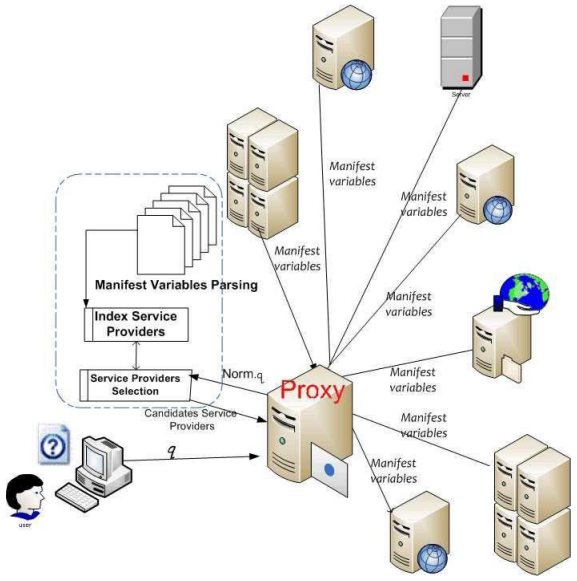


Figure 1. The Brokerage-based Cloud Service Selection

## II. Related Works

There have been some high-level discussion on service provider selection and brokerage-based frameworks in the cloud [7], [10], [15], [19]. 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. Also, although not specific to Cloud brokers, Xin et al.
[24] 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.

While cloud brokerage and selection is relatively unexplored territory, service selection problems have been studied in great depth in the context of Web services. To date, most of the works on Web service selection are based on QoS (Quality of Service) [18]. For instance, Kalepu et al [12] propose an objective measure of QoS based on the extent up to which the Web service meets its service level agreements. Paolucci et al. [17] proposed a 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. [26] 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. Benalla 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 the Web service domain. Second, we deal with much more complicated properties and relationships about the service providers due to the complexity of the Cloud.

Additionally, our work employs a recent high-dimensional data indexing technique, i.e., the distance [25]. High-dimensional data indexes [5], [6], [11], [13], [14], [23] are typically used for retrieval of similar multimedia objects such as images, and videos. It is not trivial to adapt it to the Cloud service provider selection. We propose a new encoding technique that represents multiple properties of service providers. We also design novel querying techniques that take into account unique characteristics in the Cloud such as the relationships among service providers, which do not exist in conventional high-dimensional data sets.

## III. The Brokerage-based Cloud Service Selection Architecture

In the brokerage-based Cloud service selection architecture, there are three types of entities: Cloud service provider, Cloud broker, and end-users. The Cloud broker, which has a contract with the Cloud service providers, collect their properties (e.g, service type, unit cost, and available resources), and the consumer's service requirements. The Cloud broker analyzes and indexes the service providers ac- cording to the similarity of their properties. Upon receiving the service selection request from an end-user, the Cloud broker will search the index to identify a ranked list of candidate providers based on how well they match the user requirements. This list forms the basis of the end-users' final decision.

The realization of the architecture includes two key technical issues. One is the construction of the index for managing the service providers. The other is the query algorithm for the service selection.

### *A. Indexing Cloud Service Providers*

In face of a large number of Cloud service providers, it is important to design an efficient index structure to facilitate information management and retrieval. Thus, we propose a Cloud Service Provider (CSP) index. The CSP-index is developed using the $B^+$-tree as the base structure since the $B^+$-tree is widely adopted in commercial database systems and provides a great foundation for our new index structure to be easily integrated into existing systems. In what follows, we first describe the data structure of the CSP-index and then present the index construction algorithm.

*1) Data Structure:* The internal nodes of the CSP-index have a similar format as the $B^+$-tree and serves as the search directory.

- `Service Type` ($p_1$). 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` ($p_2$). 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 [20], 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 several 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 the case of Rackspace, the level of security that can be achieved is considered *below*. 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, $p_3$). QoS is deterred-mined by the Cloud broker which analyzes the collected information about service providers over time, and ratings provided by other vendors. It is briefly represented using values high, low or medium.
- `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 transactions, the number of connections or data transfers, or the time taken for the data transfer.
- `Pricing Units` ($p_5$). This indicates how long service is reserved for. For example, the price could be charged per hour, per month, or year.
- `Instance sizes` ($p_6$). This refers to the number 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 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 de-notes if the price varies by region.
- `Subcontractors` ($p_{10}$). This indicates if subcontractors are present, and if so, what kind of services they provide.

*2) Index Construction:* The novelty of the CSP-index lies in the construction of the index keys for service providers that can speed up the query processing. Intuitively, service providers with similar properties should be stored close to each other. In this way, once the broker identifies a candidate service provider in the index, the broker can quickly locate other candidates with closely matching properties since they are stored together. To achieve this, we propose the following key generation method that captures the similarity among service providers accurately while being efficient. The algorithm consists of three major steps. The first step is to encode the properties of the service provider, and the second step is to encode the relationships among service providers built by subcontracts. Finally, the service providers are to be clustered based on the encoding to construct the index key. We elaborate on each step in the following.

**Step 1: Property Encoding.**

For each type of service provider, we encode their properties. The overall idea is to use a bit array to store the values of the service provider's properties. The bit array is of the same size for every service provider. The bit array consists of 9 sections corresponding to the first 9 properties identified in Section III-A1. The number of bits used for each section is based on the domain of each property. The encoding differs according to the types of properties.

The first property, service type, is treated specially. We employ the oneR mining algorithm to identify the same service that may be described in different ways. According to the mining result, service types falling into the same group will be assigned the same encoding.

For properties with continuous values, such as the cost, the storage capacity, we partition its domain into $n$ ranges and represent each range using a bit. *provider is 800M to 2G, the second and the third bits will be set to 1, resulting in the encoding '0110'.*

For properties with categorical values, we use a numerical

value to represent it. For example, the property "service quality" can be described using "high", "medium", "poor". Correspondingly, we convert it to numbers "3"(high), "2"(medium), "1"(poor). A typical example of a descriptive property is the privacy level. Also, if a service provider does not have a specific value for certain properties, the corresponding sections in the bit array will be set to '0'.

**Step 2: Relationship Encoding.**

The CSP-Index also stores relationships among the service providers built on 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.

**Step 3: Index Key Generation**

After the encoding, each service provider has a set of binary strings mapping each property, and a bit-array as the result of the relationship encoding. Then, we generate the integrated encoding by concatenating the bits representing the service type with the XOR-ed results of the remaining property encodings (as shown in Equation 1).

$$E_{spi} = p_{1i} \,||\, (p_{2i} \oplus p_{3i} \oplus p_{4i} \oplus p_{5i} \oplus p_{6i} \oplus p_{7i} \oplus p_{8i} \oplus p_{9i} \oplus p_{10i}) \quad (1)$$

Performing the XOR operations on the strings helps condense the resulting string to a small size, while still preserves the similarity between service providers. It is worth noting that the encoding may generate false positives, i.e., a small number of dissimilar service providers may receive similar encodings. Such false positives will be filtered out at the last step of the service selection query and will not affect the selection quality.

The encoding idea is illustrated by the following example.

*Example 3.2: Consider the following small set of properties for example*

- *Service quality: 3-High, 2-medium, 1-poor*
- *Privacy protection: 3-High, 2-medium, 1-poor*

*Suppose that a service provider $SP_1$ 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'(cost), '010'(service quality), '010'(privacy). The final integrated encoding for the service provider is then computed as follows:*

$$E_{sp1} = 0001 \,||\, (0110 \oplus 010 \oplus 010 \oplus 010) = 00010100$$

We employ the k-means algorithm [9] to cluster all the service providers based on the Hamming distance between their final encodings, where $k$ is equal to the number of service types. An auxiliary structure is maintained to store the cluster centers. Then, we leverage the idea of the distance [11], [25] to generate the indexing key $Key_s$. Using distance allows us to index the service points as data points, on a B$^+$-tree-like index structure, which as described earlier is particularly suited for this problem space due to its wide adoption in commercial database systems.

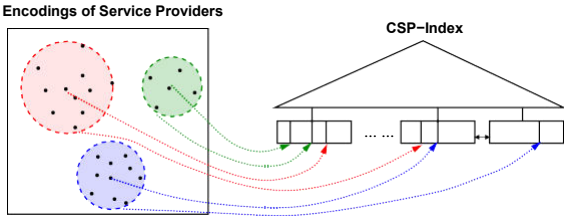


Figure 2. Structure of CSP-Index

To generate the index key, we first compute the Hamming distance (denoted as $D_h$) between the encoding of each service provider and its closest cluster center. Then, we add a scaling value $S$ to $D_h$ to form the index key $Key_s$. The scaling value is used to partition the dimensional space into regions, where each region holds a cluster of points close to each other. Therefore, the scaling value depends upon the number of regions we aim to generate. The value $S$ is directly proportional to the number of service types encountered since the number of regions must be proportional to the number of service types. $k$ is the constant used to stretch the index values so that the partitioning of the CSPs are easier. Equation 2 summarizes the key generation, where $E_{spi}$ is the property encoding of service provider I, and $E_{ck}$ is the encoding of the cluster center CK which is closest to the service provider $i$.

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

Once the indexing key is generated, the insertion and deletion in the CSP-index resemble that in the B$^+$-tree. An example CSP-index is shown in Figure 2.

## IV. Cloud Service Selection

Indexing helps the broker arrange the service providers in a way that facilitates fast information retrieval. We now proceed to discuss the detailed query algorithms for the service selection.

### A. Query Definition

A user sends a service selection query to the broker which specifies what properties and values he/she expects from the service providers. A formal definition of the service selection query is given below

*CSS Query Algorithm*

The Cloud Service Selection (CSS) query algorithm consists of four phases: (1) query encoding; (2) $k$ nearest neighbor search; (3) refinement; (4) consideration of special criteria. The query encoding converts the user query into the form of an index key of the CSP-index. Based on the query encoding, the $k$ nearest neighbor search returns $k$ candidate service providers whose index keys are similar to the query encoding and hence may satisfy the query requirements. The last two phases further exam the properties of the candidate service providers and their relationship to find the best combination of service providers that address the user's needs. We detail the key steps of each phase as follows.

**Step 1: Query Encoding.**

Given a user query, the Cloud broker treats the property requirements in the query as properties of a new service provider and encodes the properties in the same way as presented in Step 1 of the index construction in Section III-A. The obtained property encoding is compared with the cluster centers stored in the auxiliary structure. The cluster center with the smallest hamming distance to the property encoding is selected. Then, we use Equation 2 to generate the index key value for this query.

**Step 2: K-nearest neighbor search.**

Based on the obtained index key value of the query (denoted as $Key$ ), we search the CSP-index to find the $k$ candidate service providers whose property encodings are the $k$ nearest neighbors of $Key$. The search starts from the root of the CSP-index. We follow the path that contains the entry with the smallest hamming distance to $Key$, until reach the leaf node of the CSP-index. Then, we examine the property encodings stored in the leaf nodes and find the $k$ nearest values to $Key$. If the leaf node does not contain $k$ entries, we expand the search to its neighboring leaf nodes on both sides until $k$ nearest neighbors are found.

Here, the chosen value of $k$, i.e., the number of neighbors to be considered, is critical to the overall performance. If too few neighbors are retrieved, we may not find the service provider which fully satisfies the query requirements. This is because the CSP-index stores service providers according to the similarity between all of their properties, 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. Based on extensive experiments, we set $k$ to the $10^{th}$ of the total number of service providers.

**Step 3: Refinement.**

From the obtained candidate service providers, the refinement phase finds the service providers or the combinations of service providers that satisfy the query requirements.

The first step of the refinement is to further reduce the number of service providers that need to be fully examined. Specifically, we only consider top $k_2$ service providers in the $k$ candidates obtained from the previous step, where $k_2 \oplus k$. To find the top $k_2$ service providers, we create a new run-time index for $k$ candidates based on only the properties listed in the query. The key of each candidate provider in the run-time index is computed based on new property encoding similar to Equation 1: $Key_{sp_i} = QP_1 \oplus QP_2 \oplus .. \oplus QP_n$, where $QP_1, .. ., QP$R are the properties listed in the query. For example, if the user lists only service type, instance size, and cost in the query, then only these three properties are used to form the key for the run-time index. The run-time index helps to quickly order the $k$ candidate service providers according to their closeness to the query. Here the closeness is measured using the hamming distance between the index key and the query. Then, we execute a $k_2$-NN query to retrieve the top $k_2$ service providers.

The next step is to consider each querying property individually and sort the $k_2$ service providers in ascending order of their hamming distances for that property. Suppose that there are $n$ querying properties. We obtain $n$ sorted lists of service providers corresponding to each property. Recall that the querying properties are given in decreasing order of importance in the query. Therefore, we start from the sorted list of the first (i.e., the most important) querying property, and apply a greedy algorithm to find the best combination of service providers that meets the user's service requirements.

In particular, we first select the service provider on top of the first sorted list. We remove the satisfied querying properties from the 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 (discussed in Step 4) to verify possible collision and collusion among them caused by the shared subcontractors. If there exists any collision or collusion in the current solution, Step 4 will return a ranking for this solution to indicate its collision and collusion 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.

**Step 4: Consideration of Special Criteria**

The possibility of a collision or collusion 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 $SP_1$ and $SP_2$, the storage servers are located, being that it has a part However, if the user specifies that the servers be from a certain region, such as the USA, then a collision occurs if the shared subcontractor is not taken into account.
- *Collusion* is the ability of service providers or subcontractors to derive more information or meta-information about the data stored on their resources, without the explicit permission or even the knowledge of the user. It occurs due to the providers or subcontractors cross-referencing two or more data sets. For example, if the selected service providers for a given user request, uses the same subcontractor that provides a record maintenance service, this subcontractor is then possible to identify that the data stored by these service providers are linked to the same user and take advantage of such extra knowledge to infer user's information.

To detect collision and collusion, we verify the subcontractor encoding ($p_{10}$) of the service providers in the solution obtained from Step 3, to see if they have any subcontractor in common. The verification is conducted by computing $p^{i}_{10}$

$AN^{D\,PJ}_{10}$ pairwise for each pair of service providers $i, j$ in the solution. If any of the binary bit arrays that result from any of these ANDs have the first bit as 1, that means the service providers share a subcontractor. Otherwise, that means they do not have any subcontractor in common and the entire algorithm stops.

If there are shared subcontractors, we further quantify the degree of collision and collusion by assigning a ranking to the solution. Simply put, a solution that contains more shared subcontractors with more important properties will be assigned a lower rank. Specifically, we analyze the AND results of the subcontractor encodings ($p_{10}$) of each pair of service providers in the solution. If the AND result contains 1 in the second bit that represents storage space sharing, that means the corresponding pair of service providers may encounter a collision issue. If the AND result contains 1 in the third bit that represents security aspects, that means there may be a collision issue. According to the order of the properties listed in the query, we know which property is more important, the storage space or the security aspect. The final ranking is then determined by the weighted sum of the number of collision and collusion, where larger weight is given to the more important property. The ranking is returned to Step 3 to check if there is a need to find other solutions.

## V. 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 an 8GB DDR3-SDRAM-1333, 640GB Hardrive, and an Intel Core i7-2820QM quad-core processor (2.30GHz) with Turbo Boost up to 3.40GHz.

### *A. Generation of Testing Datasets*

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 [1]. 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 (Mi-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 documentation 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 [20]. Table I 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 the case of Windows Azure. `Marmot` stands for

Table I
CLOUD SERVICE PROVIDER ATTRIBUTES AND VARIABLE RANGES

| CSP Name | Variable Names | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | S | service Type | Sec | QoS | Marmot | Prcg units | 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 |
| Joined | | 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 |

measurement units, where 1 refers to measurement in terms of memory used, 2 refers to measurement in terms of several transactions, 3 stands for the number of connections or data transfers done, and 4 for the data transfer time. `Prcg units` stand 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 the 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 location-based prices. Based on the collected real data, we identified the acceptable values for each of the properties, according to 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 total of possible $10^{10}$ combinations and filter out the outliers. Pseudo-random number generator to generate a subset of the

*B. Experimental Results*

We compare our cloud service selection (CSS) algorithm with 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 as the number of different properties in the service providers returned by our solution and that by the baseline solution.

*1) Effect of Number of Service Providers:* In the first set of experiments, we compare the performance of our CSS algorithm with the baseline approach when the total number of service providers is increased from 1000 to 10000. For each set of service providers, we execute 100 service selection queries that contain 9 number of desired properties. Figure 3 shows the average service selection time.

Observe that our CSS algorithm significantly outperforms

the baseline approach and the performance gap between the two approaches are enlarged quickly with the increase of the number of service providers. Specifically, our CSS algorithm is about 100 times faster than the baseline approach when there are 10000 service providers. This demonstrates the pruning power of the CSP-index used by our approach. The CSP-index arranges service providers according to the similarity among their properties. Given a service request, the CSP-index helps to quickly direct the search to the group of service providers that may satisfy the querying properties. The baseline approach is very time consuming since it needs to check the property of each service provider and verify all possible combinations of service providers.

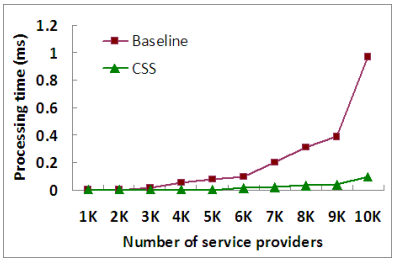


Figure 3. Processing time when varying the number of service providers

Next, we compare the results returned from our approach with that from the baseline approach to evaluate our query accuracy. Figure 4 reports the number of properties that are different in the service providers obtained from our approach when compared to the baseline approach. That is, each data point in the graph is represented as

$D = |Baseline_{prop} - CSS_{prop}|$, where $Baseline_{prop}$ is

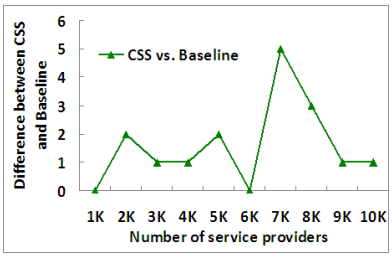


Figure 4. Accuracy of the CSS algorithm

the set of properties of the service providers obtained in the baseline approach and $CSS_{prop}$ is the set of properties of the service providers obtained using our approach. As shown, our CSS algorithm usually has just one or two properties that are different from the baseline approach. Considering the overall performance in terms of efficiency and accuracy, our approach is considerably better than the baseline approach.

*2) Effect of Number of Properties Required in the Service Selection:* To evaluate the effect of the properties, we vary the number of querying properties per request from 1 to 9 and test them in the dataset containing 5000 service providers. Two types of queries are considered. One denoted as "multiple" refers to queries that return more than one service provider for a single request. The other denoted as "single" refers to queries that can be satisfied by a single service provider. Figure V-B2 shows the results. Again, as in Figure 4, each data point in the graph is the number of properties that are different in the service providers obtained from the CSS algorithm compared to the properties obtained by the baseline approach. The first observation is that the results returned by our CSS algorithm have only minor differences from that of the baseline approach in most cases. Second, we notice that the CSS algorithm yields better accuracy, i.e., a fewer number of different properties when the query result contains just a single service provider. This is because it is easy to verify whether a single service provider matches the query requirement. There are more possibilities when selecting the combination of service providers and the service request needs to be fulfilled by multiple providers.

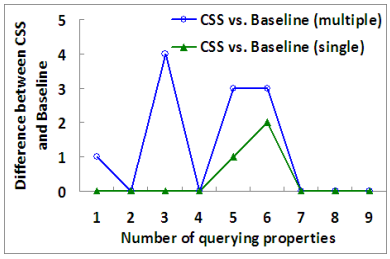


Figure 5. Effect of the number of query properties

## VI. CONCLUSION

In this paper, we proposed a brokerage-based architecture in the Cloud. Cloud brokers help end-users select and rank Cloud service providers based on the service requests. We also developed detailed algorithms to realize the proposed architecture. The next steps involved would be to refine the process of parsing the manifest variables, and providing the users an opportunity to negotiate some terms of the SLAs.

## REFERENCES

[1] B. Benatallah, M. Dumas, Q. Sheng, and A. Ngu. Declarative composition and peer-to-peer provisioning of dynamic Web services. In *Proc. of 18th International Conference on Data Engineering*, pages 297–308. IEEE, 2002.
[2] J. Burt. Gartner Predicts Rise of Cloud Service Brokerages. 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 platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.
[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of the International Conference on Very large databases (VLDB)*, pages 426–435, 1997.
[5] A. M. Despain and D. A. Patterson. X-tree: A tree-structured multi-processor computer architecture. In *Proc. of the annual symposium on Computer architecture*, pages 144–151, 1978.
[6] M. Eggebrecht. Is cloud brokerage the next big thing? http://www.ciozone.com/index.php/Cloud-Computing/Is-Cloud-Brokerage-the-Next-Big-Thingu.html.
[7] Gartner. Cloud services brokerages: The dawn of the next intermediation age.
[8] J. Hartigan and M. A. Wong. An algorithm as 136: A k-means clustering algorithm. *Royal Statistical Society. Series C (Applied Statistics)*, 28:100–108, 1979.
[9] Ivan. Cloud business trend: Cloud brokerage.
[10] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. distance: An adaptive b+-tree-based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30:364–397, 2005.
[11] S. Kale, S. Krishnaswamy, and S. Loke. Verity: a QoS metric for selecting web services and providers. In *Proc. of Web Information Systems Engineering Workshops*, pages 131–139. IEEE, 2003.
[12] N. Katayama and S. Satoh. The sr-tree: an index structure for high-dimensional nearest neighbor queries. In *Proc. of the ACM SIGMOD international conference on Management of data*, pages 369–380, 1997.
[13] K. I. Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: an index structure for high-dimensional data. *The VLDB Journal*, 3:517–542, 1994.
[14] R. Miller. Cloud brokers: The next big opportunity? http://www.datacenterknowledge.com/archives/2009/07/27/cloud-brokers-the-next-big-opportunity.
[15] A. Mondal, K. Yadav, and S. Maria. 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.
[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] J. Rao and X. Su. A survey of automated web service composition methods. *Semantic Web Services and Web Process Composition*, pages 43–54, 2005.
[18] SearchCloudComputing. Top 10 cloud computing providers of 2011.
[19] SearchCloudComputing-TechTarget. Newservers: 2011 top cloud computing provider, 2011. http://searchcloudcomputing.techtarget.com/feature/Top-10-cloud-computing-providers.
[20] C. Security Alliance. Security guidance for critical areas of focus in cloud computing.
[21] J. Stickeleather. Cloud service brokerage. http://en.community.dell.com/dell-blogs/enterprise/b/it-executive/archive/2011/02/22/cloud-service-brokerage.aspx.
[22] S. Taylor, A. Young, and J. Macaulay. Small businesses ride the cloud: Smb cloud watch - U.S. survey results.
[23] D. A. White and R. Jain. Similarity indexing with the ss-tree. In *Proc. of the International Conference on Data Engineering (ICDE)*, pages 516–523, 1996.
[24] L. Xin and A. Datta. On trust guided collaboration among cloud service providers. In *Proc. of. Collaborative Computing: Networking, Applications and Worksharing (Collaborate- Com)* pages 1–8. IEEE, 2010.
[25] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the distance: an efficient method to knn processing. In *Proc. of the international conference on Very large databases (VLDB)*, pages 421–430, 2001.
[26] 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.