Cloud Instance Selection Using Parallel K-Means and AHP

Taiyang Guo, Rami Bahsoon
guotycn@163.com
r.bahsoon@cs.bham.ac.uk
School of Computer Science
University of Birmingham, UK

Tao Chen
t.t.chen@lboro.ac.uk
Department of Computer Science
Loughborough University, UK

Abdessalam Elhabbash,
Faiza Samreen, Yehia Elkhatib
i.lastname@lancaster.ac.uk
School of Computing and Comm.
Lancaster University, UK

ABSTRACT

Managing cloud spend and qualities when selecting cloud instances is cited as one of the timely research challenges in cloud computing. Cloud service consumers are often confronted by too many options and selection is challenging. This is because instance provision can be difficult to comprehend for an average technical user and tactics of cloud provider are far from being transparent biasing the selection. This paper proposes a novel cloud instance selection framework for finding the optimal IaaS purchase strategy for a VARD application in Amazon EC2. Analytical Hierarchy Process (AHP) and parallel K-Means Clustering algorithm are used and combined in Cloud Instance Selection environments. It allows cloud users to get the recommendation about cloud instance types and job submission periods based on requirements such as CPU, RAM, and resource utilisation. The system leverages AHP to select cloud instance type. Besides, AHP results are used by the parallel K-Means clustering model to find the best execution time for a given day according to the user's requirements. Finally, we provide an example to demonstrate the applicability of the approach. Experiments indicate that our approach achieves better results than ad-hoc and cost-driven approaches.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Domain specific languages**.

KEYWORDS

Cloud Service Selection, Analytical Hierarchy Process, Parallel K-means, Clustering, Hadoop, MapReduce

ACM Reference Format:
Taiyang Guo, Rami Bahsoon, Tao Chen, and Abdessalam Elhabbash, Faiza Samreen, Yehia Elkhatib. 2019. Cloud Instance Selection Using Parallel K-Means and AHP. In *IEEE/ACM 12th International Conference on Utility and Cloud Computing Companion (UCC '19 Companion), December 2–5, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3368235.3368845

1 INTRODUCTION

In contemporary society, cloud computing has attained considerable attention and popularity as a computing paradigm. As a result of such success, the number of available public cloud services is continuously growing, including the variety of Infrastructure as a Service (IaaS) instance types. However, cloud instances offered by a number of cloud service vendors differ from one another in specification, performance, pricing policies and several other attributes. Figure 1 depicts the quantity and purpose of instance types offered by the major Cloud providers in 2015 and 2017. As can be seen in the figure, Amazon EC2 provided 57 different types of cloud instances in 2017. The figure also shows the increase of the offers in the market from 2017 compared to 2015. With such scale, the decision of selecting the optimal or near-optimal instance is a challenge for cloud customers [5, 6].

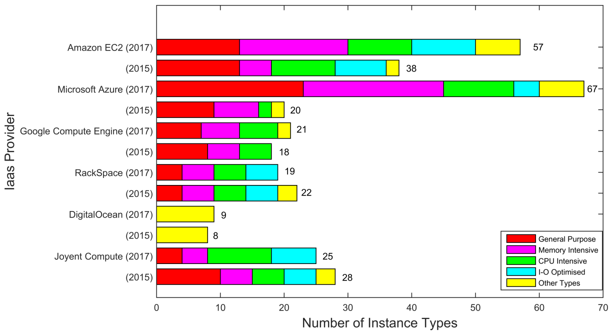

Figure 1: The number of on-demand Linux cloud instance types from major CSPs in August 2015 and July 2017.

RightScale conducted its sixth annual 'State of the Cloud Survey' in January 2017 [19], detailing the latest trends in the use of cloud infrastructure by users. Among other things, the survey asked 1,002 IT professionals about their adoption of cloud infrastructure and related technologies. According to the reported results, central IT teams have expressed a stake in selecting public clouds (65%) and private clouds (63%). It is thus evident that selecting the right cloud instance should strike a balance in meeting the needs of both customers and professional IT teams. Importantly, though, the problem is not trivial due to the diversity of candidate solutions and vast variations in provision. This calls for a systematic method to assist cloud customers in making the right decision when selecting cloud instances.

The problem of cloud service selection has been addressed in the relevant literature [9, 15, 17]. The majority of research leverage multi-criteria decision-making method (MCDM) or the optimisation method to measure quality and prioritise cloud services. Meanwhile, some of the work focuses on using machine learning to predict cloud service performance. Despite the progress made to assist customers decision making, there is still a lack of support for dynamic decision making that can effectively optimise deployment on a time-aware basis. Specifically, *'which resource type is most cost effective and at which time interval?'* remains an unsolved question. This is the potential domain for combining machine learning and MCDM methods to develop a framework of a comprehensive recommendation of both cloud service types and predicted performance. Therefore, this paper presents an attempt to tackle this problem by combining data clustering technique with a MCDM technique to systematically assist in addressing the problem of selecting cloud instances and saving costs.

The core contributions of this paper are:

- Analytical Hierarchy Process and parallel K-means [13] clustering algorithm are used and combined for solving Cloud Instance Selection problem (see Section 3).
- A Cloud Instance Selection framework for helping customers in the selection process, where Amazon EC2 cloud instance type and job submission period was taken as an example. The selection aims to provide an optimal match between customer requirements and the available Cloud service products (see Section 4).

2 RELATED WORK

Aside from manual and arbitrary methods, cloud service selection methods can be divided into two categories, MCDM-based and optimisation-based methods.

For MCDM, Saaty [16] proposed the analytic hierarchy process (AHP) to determine the best choice, which establishes a hierarchic model using special rating and comparison method to describe the relationship between decision alternatives, decision criteria, and goals. Garg et al. [7] propose a framework and an AHP-based ranking mechanism to measure the quality and prioritise cloud services. In addition, there are some studies which are conducted on specific types of cloud services like IaaS, PaaS and SaaS. For example, Godse and Mulik [8] present an approach that makes use of the AHP technique for prioritising SaaS product features and scoring products. Menzel et al. [14] propose Multi-Criteria Comparison Method for Cloud Computing, a framework based on Analytic Network Process (ANP), offering mechanisms to differentiate IaaS not only by costs but also by benefits, opportunities and risks. Limam and Boutaba [12] describe a framework for reputation-aware software service selection and rating using multi-attribute utility theory (MAUT). Zhao et al. [21], inspired by the mode of a Web search engine, propose how to use Simple Additive Weighting to find the appropriate services that satisfy the users' multiple Quality of Service (QoS) requirements, such as service response time, trust level and monetary cost.

The second category consists of optimisation-based methods. These aim to find a service that maximises or minimises one or more criteria whilst constraints are satisfied. They use techniques such as dynamic programming, greedy algorithms and integer programming. Chang CW et al. [3] design algorithms based on dynamic programming to select cloud service providers in order to maximise the benefits with a given budget. Sundareswaran et al. [18] design a B+-tree like indexing technique for managing the information of a large number of cloud service providers, and then leverage a greedy algorithm to rank and aggregate potential service providers.

Table 1: The specifications of the six EC2 instances covered in the dataset. (GP: General Purpose, CO: Compute Optimised)

Series	Instance type	vCPU	ECU	RAM (GB)	Storage (GB)	Price $/h
T2	t2.small	1	*variable*	2	20	0.026
(GP)	t2.medium	2	*variable*	4	20	0.052
M3	m3.medium	1	3	3.75	(SSD) 4	0.070
(GP)	m3.large	2	6.5	7.5	(SSD) 32	0.140
C4	c4.large	2	8	3.75	20	0.116
(CO)	c4.xlarge	4	16	7.5	20	0.232

Cloud service performance prediction for QoS by using machine learning has been explored in various aspects in cloud service selection. Chen and Bahsoon [4] propose a self-adaptive and online QoS modelling approach, to predict the QoS value as output over time by using the information on environmental conditions, control knobs and interference as inputs. Samreen et al. [17] employ machine learning techniques to develop an adaptive deployment policy, providing an optimal match between the customer demands and the available cloud service offerings. Bankole and Ajila [1], used three machine learning techniques: Support Vector Machine, Neural Networks and Linear Regression. However, all of the work has not considered predicting optimal job submission time, and more importantly, they have not provided systematic guide to select the optimal cloud service that better comply one's requirements. Thus, combing AHP and clustering algorithm can assist cloud service selection by partitioning cloud service data into groups, each of which contains different aspects of information, and taking appropriate decisions to cater customer requirements as well as predicting optimal job submission time.

3 METHODOLOGY

We now present our approach. We start by discussing the dataset upon which our work is based, then present our architectural design, and finally detail the algorithms that underpin our work.

3.1 Data preprocessing

This work utilises the dataset of Samreen et al. [17], which aims at exploring the potential use of machine learning in cloud instance selection. The dataset includes fine-grained snapshots of instance performance over all times of the day and days of the week for six different Amazon Web Services (AWS) instances.

We report on an experiment, which was conducted on Amazon Elastic Cloud Compute (EC2) due to its wide variety of instance types and high market share. In total, there are six types of instances with different configurations for the experiment, which used 64-bit Ubuntu Linux operating system. The detail can be seen in Table 1. 'vCPU' refers to the number of virtual cores, The number of 'ECU' relates to the amount of EC2 Compute Unit, 'RAM' and 'Storage' indicate the size of memory and hard disk, 'Price' indicates the hourly charge for using corresponding instance type. Our work also focuses on these six types of instances. The use case application executing on different instances is VARD [2], a tool designed to detect and tag spelling variations in historical texts. The output is aimed at improving the accuracy of other corpus analysis solutions. VARD is a single threaded and highly memory intensive application. It holds in memory a representation of the full text, as well as various dictionaries that are used for normalising spelling variations.

Data collection was continuously repeated using a fixed set of input texts over a period of seven days with a delay of ten minutes between each pair of runs. The Linux tools `vmstat` and `sysstat` were used to continuously monitor resource utilisation. The dataset consists of three parts. The first part contains the actual output of VARD running on different EC2 instances. This information is related to the configuration specification of AWS virtual machine along with the submission and completion time of VARD. The second part is related to `vmstat` monitoring information of the EC2 instances to capture the resource utilisation every three seconds whilst the VARD application is running. The final part is the `sysstat` output, which is a collection of performance tools. It logs the resource utilisation details at a fine grain level with a 60 seconds interval between each successive log entries. The logs of each experimental day are contained in a separate file. We extract CPU and memory utilisation from the last two partial datasets.

After standardisation and normalisation, we created labels referring to time in a day. We use the 1 to represent 24 hours, so 0.25 will represent 6 hours, which will help us to better divide the day into a time period. It can be noticed that the data were split into four equally groups according to the time of a day, the first one forth of samples were assigned the label "early morning" the second forth were assigned the label "morning", and then, the next forth were assigned the label "afternoon", the last forth were assigned the label "night". Detailed information can be seen in Table 1.

Table 2: Labels for the time of day.

Label	Portion of time	Portion of value	Value
Early morning	0:00:00 – 5:59:59	$0 \leq p < 0.25$	1
Morning	6:00:00 – 11:59:59	$0.25 \leq p < 0.5$	2
Afternoon	12:00:00 – 17:59:59	$0.5 \leq p < 0.75$	3
Night	18:00:00 – 23:59:59	$0.75 \leq p < 1$	4

3.2 Architecture

Our Cloud Instance Selection framework consists of three main modules (depicted in Figure 2): *Constraints*, *Decision Support*, and *Recommendation Outcome*. The most crucial part of the architecture is the Decision Support module which consists of two phases: *Instance Type Selection Phase* and *Execution Period Selection Phase*. The *Instance Type Selection phase* realises the AHP-based method. The system sends the result of the first phase as the input of *Execution Period Selection phase*, which leverages the Parallel K-Means Algorithm. There are two elements in the *Constraints* module, one is customer requirements, another is the dataset generated by running the cloud application. The *Recommendation Outcome* consists of the output of two methods. We first describe the *Constraints* module and how the two phases of Decision Core module work. The *Recommendation Outcome* will be introduced in Section IV.

3.3 Constraints

There are two modules of the *Constraints*, one is cloud user requirements and another is the dataset generated by running the case application. For cloud user requirements, our framework is different from some existing approaches which used by other cloud vendors that typically need configuration details from customers. Customer requirements can be presented by the simple input box where users do not have to input or search for comprehensive cloud attributes but provide preferred requirements on simple cloud specification, performance preference and workload. We now introduce specific

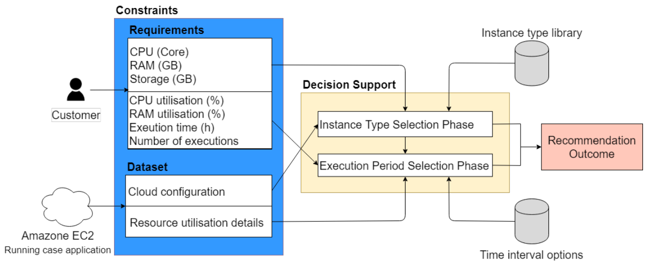

Figure 2: The architecture of our Cloud Instance Selection framework.

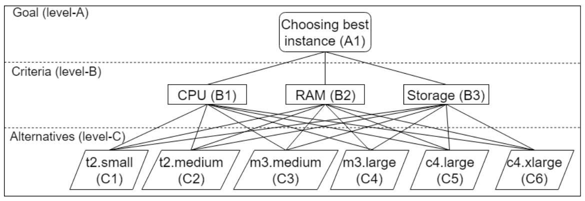

Figure 3: The Cloud Instance Selection framework Architecture.

details of the requirements. CPU: The number of CPU cores which required by the users. RAM and Storage: The volume of memory and hard disk which required by the users, the unit of measurement is GB. CPU utilisation and RAM utilisation: The percentage of CPU and memory which required by the users when they execute their cloud job. Length of execution: The users enter the length of execution time they need for the whole job, the unit of measurement is the hour. Number of executions: The customers enter how many times they want to execute their job. We can calculate the result of Length of execution divided by the Number of executions to get the user preferences of the execution time for a single job. We show an example of a user's requirement in Table 3.

3.4 Instance type selection phase

We leverage AHP for choosing optimal instance in the *Instance Type Selection Phase*. The AHP proposed by Saaty [16] is a classical MCDM method. The AHP hierarchy consists of an overall goal, a set of criteria for comparing and a group of tactics for achieving the goal. The criteria are pairwise compared according to the importance of criteria. The alternatives are compared against each of the criteria according to user preference. Figure 3 shows the hierarchy we build for solving a cloud instance selection problem. To help customer choose the optimal type of instance is the Goal as the Level-A. Criteria(Level-B) consists of CPU(B1), RAM(B2) and Storage(B3). We have six Alternatives corresponding to six instances: T2-Small as C1, T2-Medium as C2, M3-Medium as C3, M3-Large as C4, C4-Large as C5 and C4X-Large as C6. We will leverage this hierarchy to help users make decision.

There is an important question about how to define the importance of each activity. Saaty proposed a 9-grade value scale to compare different choices. The scale ranges from 1 (equal importance) to 9 (extremely importance). We can prioritise the alternatives by pairwise comparisons and insert the value in a pairwise matrix. The diagonal of the matrix consists of one, because an element is equally important when compared to itself. For example, for each pair of activities like C1 and C2, we insert their determined relative priority in the position (C1, C2) where the row of C1 meets the column of C2. In position (C2, C1) insert the reciprocal value. Continue to perform pairwise comparisons at other position in the matrix. Firstly, criteria are prioritised based on their relative significance to the goal. We consider the CPU

Table 3: An example of a customer's requirement.

User Requirements	Value
CPU (Core)	2
RAM (GB)	3.75
Storage (GB)	20
CPU Utilisation (%)	15
RAM Utilisation (%)	30
Length of execution (h)	2
Number of executions	90
Average job execution time (s)	2/90*3600=80

and RAM as equally important and moderate important than the storage of instance, because customers always prioritise CPU and memory when selecting virtual machine types. The result gives the following matrix of pairwise comparisons, as illustrated in Table 4.

Secondly, alternatives are assessed based on the difference between their configuration and the users' preference. For example, customers want to choose a cloud instance with 2 cores, 3.75 GB memory and 20 GB storage. When we insert the value into the matrix of B2, M3.medium and C4.large have the highest weight because the variance is zero. M3.large and C4.xlarge have the lowest weight due to the biggest difference. The larger the weight represents the larger the importance, so that we can perform pairwise comparisons of C1-C2, C1-C3, and so on. After the artificial judgement, we can get three matrices of pairwise comparisons for B1, B2 and B3, as illustrated in Tables 5–7.

After checking the consistency of each pairwise comparison matrix, we compare all the six alternatives with three criteria for selecting the optimal instance type. Table 8 shows the results of AHP in a matrix. The overall priority vector was calculated by using weight multiply each alternative's priority value. Because C5 has the highest value in the overall priority vector, the optimal alternative is C5 which represents C4-large when the requirement of user is an instance with 2 cores, 3.75 GB memory and 20 GB storage.

Table 4: A decision matrix of the goals.

A	B1	B2	B3
B1	1	1	3
B2	1	1	3
B3	0.33	0.33	1

Table 5: The CPU decision matrix.

B1	C1	C2	C3	C4	C5	C6
C1	1	0.333	1	0.333	0.333	1
C2	3	1	3	1	1	2
C3	1	0.333	1	0.333	0.333	1
C4	3	1	3	1	1	2
C5	3	1	3	1	1	2
C6	1	0.500	1	0.500	0.500	1

Table 6: The RAM decision matrix.

B2	C1	C2	C3	C4	C5	C6
C1	1	0.500	0.333	3	0.333	3
C2	2	1	0.500	4	0.500	4
C3	3	2	1	5	1	5
C4	0.333	0.250	0.200	1	0.200	1
C5	3	2	1	5	1	5
C6	0.333	0.250	0.200	1	0.200	1

Table 7: The storage decision matrix.

B3	C1	C2	C3	C4	C5	C6
C1	1	1	5	0.333	1	1
C2	1	1	5	1	1	1
C3	0.200	0.200	1	0.333	0.200	0.200
C4	0.333	0.333	3	1	0.333	0.333
C5	1	1	5	1	1	1
C6	1	1	1	0.500	1	1

Table 8: The priority vectors in a matrix for every alternative.

	w	C1	C2	C3	C4	C5	C6
B1	0.4286	0.0854	0.2369	0.0854	0.2369	0.2369	0.1185
B2	0.4286	0.1168	0.2994	0.2994	0.0502	0.2994	0.0502
B3	0.1429	0.2194	0.2194	0.0403	0.0820	0.2194	0.2194
Overall	–	0.1180	0.2118	0.1706	0.1348	0.2612	0.1036

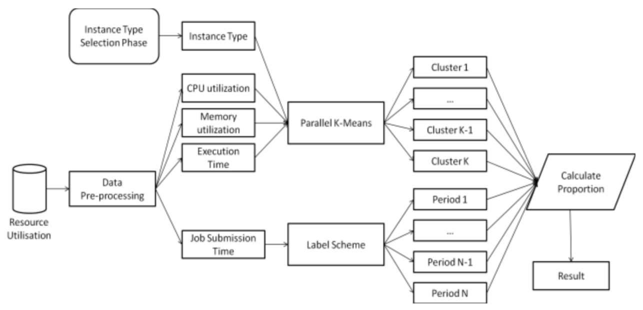

Figure 4: The architecture of the Execution Period Selection Phase.

3.5 Execution Period Selection Phase

We used k-means clustering algorithm to build the model. Execution time, CPU utilisation and memory utilisation were used as variables to find the clusters. Euclidean distance was used as the rule of distance measure. Then, the job submission time was marked according to different times of the day as described in Section III.A. In the end, we calculate the proportion between time periods and each cluster so that we can find the job submission time in which users are interested. The architecture is depicted in Figure 4 Meanwhile, we use Hadoop to implement the parallelisation of K-means. Apache Hadoop is an open-source framework for reliable, scalable, distributed processing of large data sets across clusters of computers using simple programming models. There are two reasons for using Hadoop. Firstly, we want to enhance the scalability of the framework. The traditional K-means is inefficient and unstable when it is used to calculate large-scale data sets. In the Future, we plan to incorporate more service providers, instance types and case applications into our system, Hadoop offers a new opportunity to solve these problems through algorithm parallelisation. These have been verified in detail [10, 11, 20]. Secondly, Hadoop is a free and open-source distributed computing framework which can effectively reduce the price of deploying our framework.

In our experimental platform (see Figure 5), three virtual machines were used to build the cluster structure, which contains one master node and two slave nodes. They connect to each other via the local area network and share one file system (Hadoop distributed file system). When we submit a MapReduce job via the client, the master node is used to distribute the job, and the slave node is responsible for performing the job. For the experimental software environment is, we run Hadoop v2.6.4 on top of Linux CentOS v6.5.

Figure 6 is the MapReduce flow chart of parallel K-means algorithm. As seen above, the main work to achieve parallelisation K-means is to write Mapper and Reducer. Firstly, we copy the dataset to HDFS and select the initial set of centres. HDFS will divide the data set into several blocks with equal size. The format of Mapper and Reducer input is the key-value pair. For the input of Mapper, the

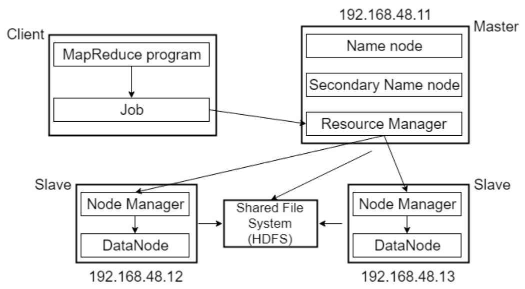

Figure 5: The experimental Hadoop platform architecture.

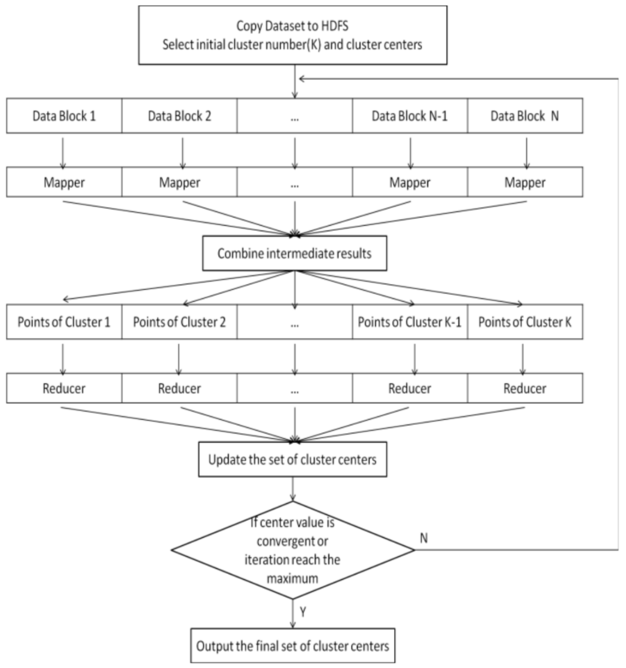
Copy Dataset to HDFS
Select initial cluster number(K) and cluster centers
Data Block 1
Data Block 2
...
Data Block N-1
Data Block N
Mapper
Combine intermediate results
Points of Cluster 1
Points of Cluster 2
Points of Cluster K-1
Points of Cluster K
Reducer
Update the set of cluster centers
If center value is convergent or iteration reach the maximum
N
Y
Output the final set of cluster centers

Figure 6: MapReduce flowchart of parallel K-means algorithm.

key is starting offset of each record, the value is the coordinate of the point. The role of Mapper is to arrange each data into the nearest cluster. After the function, the key of output is the index of the centre which is closest to this points, the value of output is the vector of this point. Its output is also the input of Reducer. The role of Reducer is to calculate the average of each cluster which is chosen as the new centre. Reduce function's output is the result of MapReduce job, its key is the index of the cluster and key is the new coordinate of new centre. This process will be iterated until the result is convergent or iteration reaches the threshold.

4 EVALUATION

We now compare our approach with the ad-hoc and cost-driven ad-hoc selection. We use an ad-hoc selection process as the baseline, where customers randomly select the instance type and time. An example is provided to demonstrate how the system is evaluated. In the cost-driven ad-hoc selection, users input the purpose of the instance and select the lowest price instance among this series and then they randomly select their job submission time. As can be seen in Table 1, T2 and M3 series are designed for general purpose, C4 is a series of

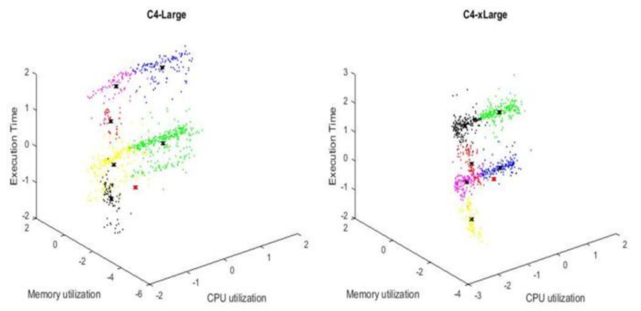

Figure 7: A 3D plot of the K-means result of C4-Large and C4-xLarge, The red x represents the user's requirement.

computing purpose. We simulated the requirements of a user as illustrated in Table 3. Through the observation of the data, we assume that the user who demands job completion within 100 seconds will buy a general series instance, otherwise they will purchase a computing series instance. We notice that the optimal instance type is C4-Large, because there is the smallest difference between C4-Large characteristic and user requirements as illustrated in Table 10.

4.1 CIS framework selection results

As an example, when the CIS framework receives the requirements of customers, decision matrices are generated by human judgement as illustrated in Tables 4–7, the output of *Instance Type Selection Phase* is C5 (i.e. C4-Large), which is indeed the best. Then the *Execution Period Selection Phase* built a clustering model for C4-Large dataset (as illustrated in Figure 7). Through calculating the distance from the centre, user requirements are assigned into the black cluster. Consequently, it recommends period 1 (0:00-5:59:59) as the optimal job submission time. The statistical data is shown in Figure 8. Note that the framework selects the optimal job submission time as the one which has the highest ratio of the number of jobs with the target performance. In conclusion, the result is optimal in both instance type and job submission period based on user requirements.

4.2 Comparison to ad-hoc selection

For the first scenario, the output of random instance type selection is C6 (i.e. C4-xLarge), which has a big difference from user preference as illustrated in Table 10. Through calculating the distance from the centre, the user requirements are assigned into the blue cluster. Execution period was selected randomly at period 4 (18:00-23:59:59) (see Figure 7). The full results are shown in Figure 8. Clearly, the result is not optimal in both instance type and job submission period.

For the cost-driven ad-hoc scenario, because the customer intends to finish their each job at about 80 seconds, we assume that they would opt for an instance with considerable computing power, therefore choosing C4-Large as the lowest costing instance type among the compute optimised series. We leverage Java program to randomly select the job submission time, the output of ad-hoc is period 4 which means customers should execute their work at 18:00-23:59:59. As can be seen in Figure 8, it is not the optimal decision. Thus the result of cost-driven ad-hoc is optimal in instance type selection but not optimal in job submission period selection.

4.3 Discussion

The methods and results of the example in different selection approaches in are shown in Table 9. We repeated this process 10 times. Experiments indicate that our approach consistently achieves better

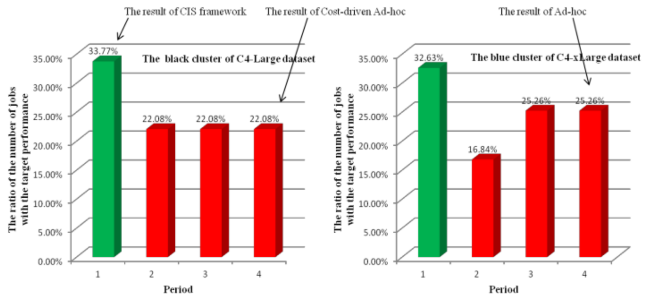

Figure 8: The statistical data of optimal job submission time for C4-Large and C4-xLarge based on target performance. Number of period reference in Table 1.

Table 9: The methods and results of example in different selection approaches

	CIS Framwork	**Ad-hoc**	**Cost-driven Ad-hoc**
Type Selection Method	ITS Phase	Random Selection	Cost-driven Selection
Type Selection Result	C5 (C4-Large)	C6 (C4-xLarge)	C5 (C4-Large)
Period Selection Method	EPS Phase	Random Selection	Random Selection
Period Selection Result	Period 1 0:00-5:59:59	Period 4 18:00-23:59:59	Period 4 18:00-23:59:59

Table 10: User requirements and the characteristic of the instance selected by different approaches

	Requirements	**Characteristics**	
CPU (Core)	2 cores	2	4
RAM (GB)	3.75	3.75	7.5
Storage (GB)	20	20	20

requirement compliance than ad-hoc approaches. We hence conclude that our framework can be used for effective cloud service selection. However, there are some limitations in our evaluation. Firstly, the approach may not scale well in cases where there are thousands of instance types as the decision matrix of AHP needs to be generated by human judgement. Secondly, we only focus on the dataset of six types of the cloud instances from Amazon EC2. Investigating a wider range of instance types and those from other cloud vendors is a plan in future work for more comprehensive evaluation. Finally, K-Mean clustering is a non-deterministic machine learning algorithm, thus it is difficult to guarantee its stability. However, this again would be better determined in a wider study as already discussed.

5 CONCLUSION AND FUTURE WORK

Cloud service customers are often faced with plenty of trade-offs when selecting and deploying cloud instances Research into customer cloud instance selection is an under-researched area. We have designed a Cloud Instance Selection framework to help customers select cloud instances and manage cloud spends. We explored the role of clustering algorithms to support cloud decision making. An overall architecture is proposed and Analytical Hierarchy Process and parallel K-Means Clustering algorithm are combined to address the problem of Cloud Instance Selection. Specifically, our study focused on recommendations about the selection of instance types in Amazon EC2 and job submission period for the execution of the VARD application. Our evaluation results show that: our framework is indeed effective and can be used for cloud service selection; CIS framework which combines parallel K-Means and AHP outperforms the ad-hoc and cost-driven ad-hoc approaches. Our future work will investigating more selection criteria and constraints at a larger scale to test the scalability of our framework, where we will include more service providers, more instance types, more period options and applications.

REFERENCES

[1] Akindele A. Bankole and Samuel A. Ajila. 2013. Predicting cloud resource provisioning using machine learning techniques. In *Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 1–4. https://doi.org/10.1109/CCECE.2013.6567848

[2] Alistair Baron and Paul Rayson. 2008. VARD2: A tool for dealing with spelling variation in historical corpora. In *PG Conference in Corpus Linguistics*.

[3] Chia-Wei Chang, Pangfeng Liu, and Jan-Jan Wu. 2012. Probability-based cloud storage providers selection algorithms with maximum availability. In *41st International Conference on Parallel Processing*. IEEE, 199–208.

[4] Tao Chen and Rami Bahsoon. 2016. Self-adaptive and online qos modeling for cloud-based software services. *IEEE Transactions on Software Engineering* 43, 5 (2016), 453–475.

[5] Cloud Standards Coordination (Phase 2). 2016. *Cloud Computing Users Needs - Analysis, conclusions and recommendations from a public survey.* Special Report 003 381 V2.1.1. ETSI. 12–19 pages. http://csc.etsi.org/phase2/UserNeeds.html

[6] Abdessalam Elhabbash, Faiza Samreen, James Hadley, and Yehia Elkhatib. 2019. Cloud Brokerage: A Systematic Survey. *ACM Computing Surveys, to Appear* (2019).

[7] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. 2011. Smicloud: A framework for comparing and ranking cloud services. In *International Conference on Utility and Cloud Computing*. IEEE, 210–218.

[8] Manish Godse and Shrikant Mulik. 2009. An approach for selecting software-as-a-service (SaaS) product. In *International Conference on Cloud Computing*. IEEE, 155–158.

[9] S. Gupta, V. Muntes-Mulero, P. Matthews, J. Dominiak, A. Omerovic, J. Aranda, and S. Seycek. 2015. Risk-Driven Framework for Decision Support in Cloud Service Selection. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 545–554. https://doi.org/10.1109/CCGrid.2015.111

[10] Amresh Kumar, M Kiran, and BR Prathap. 2013. Verification and validation of MapReduce program model for parallel k-means algorithm on hadoop cluster. In *International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 1–8.

[11] Qing Liao, Fan Yang, and Jingming Zhao. 2013. An improved parallel K-means clustering algorithm with MapReduce. In *International Conference on Communication Technology (ICCT)*. IEEE, 764–768.

[12] Noura Limam and Raouf Boutaba. 2010. Assessing software service quality and trustworthiness at selection time. *IEEE Transactions on Software Engineering* 36, 4 (2010), 559–574.

[13] David JC MacKay and David JC Mac Kay. 2003. *Information theory, inference and learning algorithms.* Cambridge university press.

[14] Michael Menzel, Marten Schönherr, and Stefan Tai. 2013. $(MC^2)^2$: criteria, requirements and a software prototype for Cloud infrastructure decisions. *Software: Practice and Experience* 43, 11 (2013), 1283–1297. https://doi.org/10.1002/spe.1110

[15] Ioannis Patiniotiakis, Yiannis Verginadis, and Gregoris Mentzas. 2014. Preference-based Cloud Service Recommendation As a Brokerage Service. In *Proceedings of the 2Nd International Workshop on CrossCloud Systems (CCB '14)*. ACM, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/2676662.2676677

[16] Thomas L. Saaty. 1990. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research* 48, 1 (1990), 9 – 26. https://doi.org/10.1016/0377-2217(90)90057-I Desicion making by the analytic hierarchy process: Theory and applications.

[17] Faiza Samreen, Yehia Elkhatib, Matthew Rowe, and Gordon S. Blair. 2016. Daleel: Simplifying cloud instance selection using machine learning. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 557–563. https://doi.org/10.1109/NOMS.2016.7502858

[18] Smitha Sundareswaran, Anna Squicciarini, and Dan Lin. 2012. A brokerage-based approach for cloud service selection. In *International Conference on Cloud Computing (CLOUD)*. IEEE, 558–565.

[19] Kim Weins. 2018. Cloud computing trends: 2018 state of the cloud survey. RightScale.

[20] Kehe Wu, Wenjing Zeng, Tingting Wu, and Yanwen An. 2015. Research and improve on K-means algorithm based on hadoop. In *International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 334–337.

[21] Laiping Zhao, Yizhi Ren, Mingchu Li, and Kouichi Sakurai. 2012. Flexible service selection with user-specific QoS support in service-oriented architecture. *Journal of Network and Computer Applications* 35, 3 (2012), 962 – 973. https://doi.org/10.1016/j.jnca.2011.03.013 Special Issue on Trusted Computing and Communications.