ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 3, No. 6, 67-73 (2018)

www.astesj.com

Special Issue on Multidisciplinary Sciences and Engineering

ASTESJ

ISSN: 2415-6698

Similarity-based Resource Selection for Scientific Workflows in Cloud Computing

Takahiro Koita[1,*], Yu Manabe[2]

[1]*Doshisha University, Faculty of Science and Engineering, Japan*

[2]*NAIST, Department, Division of Information Science, Japan*

ARTICLE INFO

Article history:
Received: 15 August, 2018
Accepted: 22 October, 2018
Online: 01 November, 2018

Keywords:
Cloud computing
Amazon web services
Scientific workflows
Resource selection

ABSTRACT

There are high expectations for commercial cloud services as an economical computation resource when executing scientific computing workflows, for which the computation is increasing on a daily basis. However, no method has been developed for determining whether a scientific computing workflow can be executed at a low usage cost, and thus scientists have difficulty in selecting from the diverse range of computational resources. The aim of this study is to provide clear criteria for selecting a computational resource while executing a scientific computing workflow. This study focuses on the performance of application execution for one such commercial cloud service, Amazon EC2, and proposes a method for selecting the optimal resource showing high similarity to a target application in execution time and usage cost. The novelty of this study is its approach of employing application similarity in resource selection, which enables us to apply our method to unknown applications. The contributions of this work include (1) formularizing performance values of computational resources, as well as similarity values of applications, and (2) demonstrating the effectiveness of using these values for resource selection.

1. Introduction

This paper is an extension of work originally presented in ICBDA2018 [1]. Scientific computing workflows [2] are applications that performs a sequence of processes by dividing applications handling scientific computing into small tasks and, by executing these tasks in stages. Figure 1 shows an overview of scientific computing workflows of Epigenomics and Motage. Characteristically, scientific computing workflows can deal with large amount of data, and the work quantity differs for each task. Here, as examples, we introduce three types of scientific computing workflows. Montage is an application developed by NASA that processes celestial images. A feature of Motage is that, since it handles large-sized images, it requires high levels of I/O performance. Broadband is an application that generates a vibration record diagram from multiple earthquake simulations, and requires high levels of memory performance. Epigenomics is an application dealing with DNA, and requires high levels of processing ability based on CPU performance. Thus far, scientific research has mainly consisted of experiments and theories. However, with developments in hardware, advanced calculation

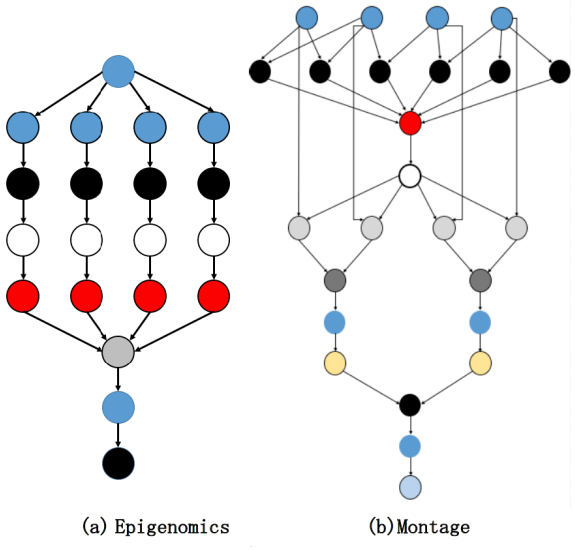

Figure 1. Scientific Workflow

*Takahiro KOITA, Email: tkoita@gmail.com

Table 1. List of Instances

(a) Role list

role	instance name	
General purpose	t2	m4
CPU optimized	c4	
Memory optimized	r3	x1
Storage optimized	i2	d2
GPU instance	g2	
GPU computing	p2	

(b) Performance list

performance			
nano	micro	small	medium
large	xlarge	2xlarge	4xlarge
10xlarge	16xlarge	32xlarge	

Table 2. Instance Performance

	vCPU	ECU	memory (GB)	storage (GB)	cost ($/h)
t2.micro	1	variable	1	8	0.013
c4.large	2	8	3.75	8	0.105
m4.large	2	6.5	8	8	0.12
m4.xlarge	4	13	16	8	0.239
r3.large	2	6.5	15.25	32	0.166
i2.xlarge	4	14	30.5	800	0.853

has become possible, and computer-based simulations have become an essential new research method. The importance of scientific computing workflows are only expected to grow for future science. Many scientific computing workflows have been developed based on distributed processing using high performance computers (HPC) with grids, PC clusters, and supercomputers, and scientists have used their own PC clusters and grid computing, such as Open Science Grid [3], when executing scientific computing workflows. However, through the development of hardware, the data quantity that can be operated is increasing on a daily basis, and the processing capabilities of computation sources and the storage capacity required are expanding in the same way.

When executing scientific computing workflows, for which the computational data is increasing on a daily basis, the use of commercial sloud services as a computation resource, in place of PC clusters and Open Science Grid, has attracted attention. The commercial cloud service is a service in which scientists can use servers on the network by paying usage fees. Features of such services are that computational sources and storage can be swiftly added, and computational sources of various performance (instances) are prepared. Scientific computing workflows are designed based on distributed processing, and it is possible to execute these in the cloud in which distributed processing is performed through distributed computing. Additionally, as a wide variety of performance instances are prepared, tasks with different processing can be performed in respectively optimized environments. It also has the characteristics of being a measured rate system in which you only pay for the time you use, the fact that maintenance costs are not required, and initial investment for constructing facilities is not necessary. It promises to be applicable to scientific computing workflows, and is to be used as a highly-economical computation source. In this study, we use the commercial cloud service Amazon Elastic Compute Cloud (EC2) [4] used in the preceding research [5]. EC2 is a web service provided by Amazon. The users can select virtual machines, called instances, according to various purposes. With EC2, five types of roles and multiple respective processing resources are prepared. A list of the instances is shown in Table 1.

One of the important problems with using EC2 is that it is difficult to select the instance and the application to execute with the instance performance table. If the instance performance does not satisfy the performance requirements of the application, execution will be impossible, or the execution time will increase, leading to an increase in usage costs. On the other hand, if the instance performance is higher than necessary, the cost per unit time will be higher, and even if the execution time is shorter, the costs would increase. Currently, when selecting the instance to execute the application, specialized knowledge about applications and cloud or user experience are required. This stiuation makes it difficult to select a suitable computational resource from a large number of computation resources when considering execution time and usage costs, and this is the problem for scientists using commercial cloud services.

To solve the problem, this study aims to provide a clear criterion for selecting instances for executing scientific computing workflows. Using the provided selection criteria, it will be possible for scientists to casually engage in cloud services, and to perform experiments using advanced computing resources for low research fees. The novelty of this study is its approach of employing application similarity in resource selection, which enables us to apply our method to unknown applications. The contributions of this work include (1) formularizing performance values of computational resources, as well as similarity values of applications, and (2) demonstrating the effectiveness of using these values for resource selection.

2. Current Issues

There are four main issues in executing scientific computation workflow using the cloud, as follows:

1) Virtualization overhead

2) Low throughput in shared/parallel file systems

3) Low network performance

4) Unclear usage costs

Issue 1 appears in a significant way when CPU performance is required. Additionally, 2 and 3 clearly have an impact on applications requiring I/O performance [6]. Based on the above features, the current commercial cloud services cannot achieve HPC-equivalent performance. It is expected that issues 1 to 3 shall be resolved on the hardware front, through the development of commercial cloud services. However, this will not solve 4, which occurs when scientists use commercial cloud services. Scientists need to select the computation resources satisfying the processing ability required for the application based on uncertain factors such as their own knowledge and experience. A cause of issue 4 not being satisfied is that there are no criteria for selecting computational resources that consider the necessity of applications [7].

This study focuses on the issue of usage cost. The issue is how to select instances that will satisfy computing performance requirements and have the lowest user cost when executing a scientific workflow on a commercial cloud service. In EC2, the performance of each instance is published, and users can determine the computation resources based on the performance table values. Table 2 shows part of the published performance table. vCPU expresses the number of virtual Server cores, and ECU (EC2 Computing Unit) is a numerical representation of the total processing performance for the instants. In case of an instance where ECU is 8 and vCPU is 2, the CPU processing ability per core is 4. EBS (Elastic Block Storage) is the block unit storage provided by Amazon, and a total of four types are prepared, comprising two types of SSD and two types of HDD [8]. For all instances in this study, the versatile SSD type found in the default settings is used.

Currently, the user selects the instance using their experience, based on the performance capability required for the executed applications and the values of the instance performance table, and it is possible that the instance with the shortest execution time or the lowest usage costs may not be selected.

Table 3. Instance Performance Values

	ECU	EMU	EFU
m4.large	6.50	9.30	9.50
m4.xlarge	13.0	9.62	9.25
c4.large	8.00	8.00	8.00
c4.xlarge	16.0	8.04	9.08
r3.large	6.50	8.48	9.05
r3.xlarge	13.0	8.45	9.05

We explain this situation using the example of a prime number calculation application. Prime number calculation applications are applications that mainly require CPU processing ability. For this reason, it is predicted that the user will select the c4 interface, which has enhanced CPU performance. However, at that time, they need to decide whether to choose the c4.large with 8 ECU, or the c4.xlarge with 16 ECU. The result of actually executing this was that the execution time was shorter for c4.xlarge, but the usage costs were lower with c4.large. Due to this, until we actually execute the application, it is unclear which instance has the shortest execution time or which has a lower usage costs. Additionally, the processing ability used for prime number calculation examples is virtually CPU only, but with the actual application, memory and I/O processing ability are required at the same time. When selecting the instance, it is important to have a proper understanding of the processing ability required by the application.

To execute the application with a short execution time or a low usage cost, it is necessary to quantitatively grasp the performance ability required by the application and the instance performance and clarify these relationships. Therefore, in the next section, we will quantitatively demonstrate the instance performance and the processing ability required by the application and perform preliminary experiments to provide clear selection criteria.

Very few studies have been made to quantitatively grasp the performance required by the application or the instance. Tovar et al. [9] classified tasks in scientific workflows and proposed an estimation method for the tasks. They showed that the execution time can be estimated and that CPU, memory and I/O performance indexes are important for this estimation. Sfiligoi et al. [10] showed the characteristics of scientific workflows statistically, and the results were effective for their experiment's applications. However, these studies are useful only for known applications whose behavior information can be given well in advance of instance selection. Consequently, if such information is insufficient, these studies cannot be applied. Thus, the current study employs several values to achieve resource selection for unknown applications. Furthermore, previous studies assumed that their target instance was a single type and thus did not consider the various types of instances in commercial cloud services.

3. Preliminary Experiment

We perform preliminary experiments to quantitatively show the instance performance and processing ability required by the application.

3.1. Instance Performance Value

We describe the instance performance as numerical values. We focus on instance performance in terms of CPU, memory, and I/O. This is because CPU, memory, and storage are enhanced respectively in EC2, and because the instances are mainly prepared in relation to these, it is assumed that these will have the greatest impact on execution time and usage cost. The CPU processing ability uses ECU, published by Amazon. In this study, memory and I/O processing ability are defined respectively as EMU and EFU, and these are measured and expressed numerically in these preliminary experiments. In these preliminary experiments, the versatile instances m4.large and m4.xlarge，the CPU optimization instances c4.large and c4.xlarge，and the memory optimization instances r3.large and r3.xlarge are used. The performance of each of these is shown, respectively, in Table 3.

Table 4. Execution Performance Values
(partial result only for two applications in [11])

	execution time [sec]	CPU	memory	I/O
Memory bound apl.	291.6	44.9	31.4	30.7
I/O bound apl.	25.6	3.94	2.75	2.70

Measurements are performed using a program prepared for the purpose of measuring performance evaluations. Memory performance evaluations involve reading and writing memory multiple times, whereas I/O performance is assessed by reading and writing a text file multiple times. The respective execution times are measured, with the ratio with the c4.large value and 8, which is the same as ECU, to discover the EMU and EFU of each instance. The respective instance performance is shown in Table 4.

3.2. Execution Performance Value

In this experiment, the processing ability required by the application expressed as a numerical value is used as the execution performance value. The execution performance value shows the effect of the CPU, memory, and IO required for the application on the execution time. As an example, we shall show a formula for obtaining the execution performance value based on CPU performance. The performance values based on memory or I/O performance can be calculated by making the respective ECU values the EMU and EFU values according to the following formula.

Here, as an example, we shall seek the two application execution performance values of memory performance evaluation and I/O performance evaluation used when evaluating instance performance. The execution performance values need to actually be executed with the instances. In this preliminary experiment, measurement was performed using the versatile instance m4.large. The execution time and execution performance values based on the CPU, memory, and I/O performance are as follows.

From the results, we can see that the size of the execution performance values changes depending on the execution time, and that differences appear in the execution performance value ratios based on the processing content. The two applications used in this preliminary inspection have high execution performance values based on CPU performance and, as with the instance performance evaluation, the execution time for both applications was shorter with the c4 instance optimized for CPU performance, by referencing the execution performance values, it is possible to grasp the processing capability required by the application.

4. Proposed Method

We propose a method for selecting resources that uses an application with similar execution performance to select the instance that can run an application in the lowest time or with lowest usage cost. In the past, the run time and usage cost were unknown before actually running an application, and there was a risk of costs increasing when the application was run several times. The proposed method enables a resource to be selected, running the application a minimum number of times, by considering the application execution performance values and instance performance.

We expect that if the performance required by two applications is the same, the computing resources required for the shortest execution time or lowest usage cost will be the same for them as well. The proposed method selects an application with similar execution performance values as the application in question from among several that have been run in the past, and selects the computing resource able to execute the application in question in the shortest time or at lowest cost. The method is comprised of the following four steps.

1) Standardization Step: Measure the execution performance of the sample applications

2) Measurement Step: Measure the execution performance of the application in question

3) Comparison Step: Select a sample application with similar execution performance values

4) Selection Step: Select the instance with shortest execution time or with lowest usage cost

Details of each step are described below.

Standardization Step: The sample applications are executed on each instance, and the instances producing the shortest run time and lowest usage cost are selected. The execution performance is also computed using an arbitrarily selected instance. Several applications performing different processes are used as sample applications for the proposed method.

Measurement Step: This step deals with the application for which a computing resource is being selected. The application is executed on an instance selected in the standardization step to measure its execution performance values.

Comparison Step: In this step, the execution performance values of all sample applications measured in the standardization step are compared with the execution performance values of the application in question, as measured in the measurement step. For this method, a similarity level is used for this comparison. The similarity level is expressed as a distance between the execution performance values of the two applications. The normalized execution performance values of CPU, memory and I/O of application A are denoted $E_{Ae}, E_{Am},$ and E_{Ai}, respectively. These execution peformance values, are obitained by the preliminary experiment described in Section 3. Similarly, the execution performance values for application B are denoted $E_{Be}, E_{Bm},$ and E_{Bi}. The similarity, D_{AB}, is given by the following equation (1). This equation uses Euclidean Distance between applications A and B. If the similarity value is sufficiently high, they are considered similar applications. The highest similarity value is thus used to select the most similar application. To select a resource, the target application A is fixed while application B varies. That is, the distances to application A from all other applications are calculated. The distance is determined by the values of execution time, memory, and I/O described in the previous section. More details can be seen in an earlier work [11].

$$D_{AB} = \sqrt{(E_{Ae} - E_{Be})^2 + (E_{Am} - E_{Bm})^2 + (E_{Ai} - E_{Bi})^2} \quad (1)$$

Selection Step: In the selection step, the similarity of the application with each of the sample applications is computed. The sample application with the smallest similarity level is selected as the one that is most similar to the application in question. The instance able to execute this most-similar sample application with the shortest execution time or lowest cost is then selected as the instance to run the application in question.

5. Evaluation

To show that instances can be selected based on similarity of execution performance values, we conducted experiments to evaluate the effect of similarity on the execution time and usage cost.

5.1. Experiment Overview

Using the same instances as in the preliminary experiments, the execution time, usage cost and similarity were measured for multiple applications. If the instances running the applications with the lowest execution time and usage cost are the same for applications that are similar, the proposed method will select resources correctly. The instances used were the same as those used in the preliminary experiments. The test procedure was as follows:

1) Run applications

2) Calculate execution performance values

3) Calculate similarities

4) Select similar applications.

Each of these steps is described in more detail below.

Run applications: Multiple applications were run on each instance, and the execution time was measured. From the execution times for each instance, the instances producing the shortest execution time and lowest usage cost were selected for each.

Calculate execution performance values: Each application was run on an arbitrarily selected instance and the execution performance values were measured. For these tests, we used the m4.large general-purpose instance.

Calculate similarities: Here, the computed execution performance values were normalized, and the similarity to all of the other applications was computed for each application.

Select similar applications: For each application, the one with the lowest similarity level was selected as the most similar application. Each application was compared with the other application most similar to it, and we checked whether the instances producing the shortest execution time and usage cost were the same.

5.2. Applications

The applications used here include Sysbench [12] and UnixBench [13], which are comprehensive benchmark applications, and Hadoop [14], which is a distributed framework. These are described in more detail below.

Sysbench is a general benchmark application for Linux/Unix operating systems. Sysbench has six types of evaluation (e.g. CPU or memory) and enables each of them to execute with adjusting application parameters such as the number of CPUs or the file size. For our experiments, we performed CPU, memory and I/O tests. Prime numbers are computed for the CPU test, reading and writing to memory is done for the memory test, and reading and writing files to storage is done for the I/O test. Each test was done repeatedly and the execution times were measured. The term of test means a specific execution to perform one type of evaluation.

UnixBench is a benchmark application used with Unix-type operating systems. The test covers a variety of tasks from integer arithmetic through to OS system calls. In these experiments, benchmarks for integer computation (Dhrystone), floating-point computation (Whetstone), and file copying (fsdisk) were performed. Results are given in terms of processing capability per unit time. These were converted to results in terms of a time required to complete a fixed-size process for these experiments.

Hadoop is a distributed framework that enables multiple computers to be treated as a single computer with improved performance. Hadoop can be used in any of three modes: stand-alone mode, which runs on a single CPU, pseudo-distributed mode, which virtualizes use of two machines on a single machine, and fully-distributed mode, which uses multiple computers. For these experiments, we used pseudo-distributed mode, measuring execution time for standard sample processes including computing pi, counting words, and sorting files.

5.3. Results

Each application was executed on each of the instances. For some of the applications, such as Dhrystone in UnixBench, the results are given in number of loops per second rather than total execution time. In such cases, the results were converted to a time required to perform a set number of loops. Execution times and usage costs are given in Tables 4 and 5, and execution performance values, as discussed previously, are given in Table 6.

Instance c4.xlarge had the shortest execution time, and instance c4.large had the lowest usage cost in most cases. Execution performance values were calculated using the execution times and the ECU, EMU, and EFU performance values for m4.large, and these were then normalized.

Similarities were then computed using these values. Below, we give an example of computing the similarity, D_{pw}, is given by equation (2) using the execution performance values from the prime number computation in Sysbench and the word count process on Hadoop.

$$D_{pw} = \sqrt{(0.038 - 0.020)^2 + (0.047 - 0.025)^2 + (0.047 - 0.025)^2} \quad (2)$$

Similarities were computed for all process pairs, and that with the smallest similarity value was designated as the similar application for each application. Table 8 shows whether the instances producing the shortest execution time and lowest usage cost were the same for these similar applications.

The process most similar to the prime number computation on Sysbench was word count on Hadoop. The instance with the shortest execution time for the prime number process in Sysbench was c4.xlarge, which was the same as for the Hadoop word count,

Table 5. Execution Times [sec]

application		execution time					
		m4.large	m4.xlarge	c4.large	c4.xlarge	r3.large	r3.xlarge
Sysbench	prime number	157.51	78.99	128.68	64.32	153.31	76.47
	memory read	42.55	38.61	35.79	33.72	40.62	36.39
	memory write	52.32	41.82	43.63	36.96	49.90	40.09
	random read/write	2.07	0.42	1.71	0.78	1.51	1.00
	sequential read/write	34.42	22.90	34.36	22.90	38.14	24.60
hadoop	pi	3328.29	1692.45	2828.35	1438.95	3339.85	1698.05
	word count	83.14	54.62	77.71	50.90	88.76	56.86
	file sort	40.86	36.68	40.00	36.36	42.82	36.55
UnixBench	Dhrystone	0.76	0.85	0.64	0.32	0.74	0.37
	Whetstone	0.34	0.73	0.35	0.17	0.41	0.20
	fsdisk	0.71	0.76	0.68	0.83	0.95	1.14

Table 6. Usage Costs [$]

application		usage cost					
		m4.large	m4.xlarge	c4.large	c4.xlarge	r3.large	r3.xlarge
Sysbench	prime number	21.89	21.96	16.21	16.21	30.66	30.51
	memory read	5.91	10.73	4.51	8.50	8.12	14.52
	memory write	7.27	11.63	5.50	9.31	9.98	15.99
	random read/write	0.29	0.12	0.22	0.20	0.30	0.40
	sequential read/write	4.78	6.37	4.33	5.77	7.63	9.81
Hadoop	pi	462.63	470.50	356.37	362.62	667.97	677.52
	word count	11.56	15.18	9.79	12.83	17.75	22.69
	file sort	5.68	10.20	5.04	9.16	8.56	14.58
UnixBench	Dhrystone	0.11	0.24	0.08	0.08	0.15	0.15
	Whetstone	0.05	0.20	0.04	0.04	0.08	0.08
	fsdisk	0.10	0.21	0.09	0.21	0.19	0.46

Table 7. Execution Performance Value

application		performance value		
		CPU	memory	I/O
Sysbench	prime number	0.03837	0.04725	0.04725
	memory read	0.01030	0.01270	0.01270
	memory write	0.01269	0.01564	0.01564
	random read/write	0.00042	0.00054	0.00054
	sequential read/write	0.00832	0.01026	0.01026
Hadoop	pi	0.81248	1.00000	1.00000
	word count	0.02021	0.02490	0.02490
	file sort	0.00989	0.01220	0.01220
UnixBench	Dhrystone	0.00010	0.00015	0.00015
	Whetstone	0.00000	0.00002	0.00002
	fsdisk	0.00009	0.00013	0.00013

Table 8. Matching Instance and Result of Similarity-based method

application		best time	best cost	similarity application	best time instance	best cost instance
Sysbench	prime number	c4.xlarge	c4.xlarge	word count	Same	Different
	memory read	c4.xlarge	c4.large	file sort	Same	Same
	memory write	c4.xlarge	c4.large	memory read	Same	Same
	random read/write	m4.xlarge	m4.xlarge	Dhrystone	Different	Different
	sequential read/write	m4.xlarge	c4.large	file sort	Different	Same
Hadoop	pi	c4.xlarge	c4.large	prime number	Same	Different
	word count	c4.xlarge	c4.large	memory write	Same	Same
	file sort	c4.xlarge	c4.large	memory read	Same	Same
UnixBench	Dhrystone	c4.xlarge	c4.xlarge	fsdisk	Different	Different
	Whetstone	c4.xlarge	c4.large	fsdisk	Different	Same
	fsdisk	c4.large	c4.large	Dhrystone	Different	Different

so the instances with the shortest execution time matched. On the other hand, the instance able to run the Sysbench prime number process at lowest cost was c4.xlarge, while for Hadoop word count, it was c4.large, so the lowest cost instances did not match. This result appears on the first row of data in Table 7. From left to right, it indicates that for the prime number process in Sysbench, the similar application was Hadoop word count, that the instances with shortest execution time matched, and that the instances with lowest usage cost did not match.

6. Discussion

We now discuss the results of these evaluation experiments. Instances for which execution with short execution time and low usage charges are possible tended to be c4.xlarge and c4.large, respectively. The cause of this is that many of the applications used in this experiment were CPU-bound, and this is considered to have had a major impact on the match rate. In particular, the prime number calculation by Sysbench and the performance required for Dhrystone in UnixBench is biased toward the CPU. As the processing for these had the shortest execution time and lowest usage costs in c4.xlarge, which is optimized for the CPU, this is a result compatible with the published ECU values. For the scientific computing workflow, processing differs depending on the task, and there are tasks that require a lot of non-CPU processing. For that reason, in the reading and writing of memory for Sysbench, which is an application that requires not only CPU, but also memory and I/O processing performance, and file sorting by Hadoop, execution time became shorter due to CPU performance. CPU performance is important even for applications requiring memory and I/O processing performance; therefore, creating a calculation formula that is weighted in consideration of the impact of each on execution time and usage cost is effective when selecting resources.

To apply this method to the scientific computing workflows carrying out a variety of processing, it is necessary to increase the number of sample applications handled and support a more diverse range of processing. Additionally, as the sample applications used in this test have a low computational volume, there is a concern that it cannot support scientific computing workflows handling huge volumes of processing. A greater diversity of sample applications is required to realize this method and enable the selection of computational resources in scientific computing workflows.

7. Conclusion

The objective of this experiment is to achieve a method of selecting resources based on execution time and usage costs when using commercial cloud services for scientific computing workflows. By using instance performance and the execution performance values required for the application, we measured the features of the application for a certain instance. The aim is to propose a method for selecting instances that can be executed in the shortest execution time with the lowest usage costs, by referencing similar applications for the measured execution performance values. In the evaluation experiment, we verified the effectiveness of selecting resources based on similarity. The match rate of the results was approximately 55%, and a large impact was present in CPU-bound applications. As this considers the impact on execution time when making resource selections more than for memory or I/O, it is necessary to focus on CPU performance.

Our experiment showed that the proposed method based on similarity can usually select the best instance for Hadoop or Sysbench-type applications. Furthermore, scientific computing workflow applications are mainly executed using few system functions, like Hadoop and Sysbench-type applications. Thus, our method would be effective in selecting resources for many types of scientific computing workflows. On the other hand, if the application requires many system functions, like UnixBench, the similarity calculation requires weighting factors to handle complicated behavior.

References

[1] T. Koita Performance Evaluation of Memory Usage Costs for Commercial Cloud Services, Proc. of the IEEE 3rd Int'l Conf. on Big Data Analysis (ICBDA2018), pp.307-311, 2018.
[2] Ewa Deelman, Pegasus and DAGMan From Concept to Execution: Mapping Scientific Workflows onto Today's Cyberinfrastructure, Proc. of the Advances in Parallel Computing, vol.16, pp.56-74, 2008.
[3] Open Science Grid, https://www.opensciencegrid.org/.
[4] Amazon Elastic Compute Cloud (EC2), http://www.amazon.com/ec2/.
[5] Y. Manabe, Performance comparison of scientific workflows on EC2, IPSJ technical report, 2016.
[6] S. Ostermann, A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing, Proc. of the Cloud Computing, pp.115–131, 2010.
[7] G. Juve, Scientific workflows and clouds, Crossroads, vol.16, pp.14-18, 2010.
[8] G. Juve, Scientific workflow applications on Amazon EC2, Proc. of the 5th IEEE International Conference on e-Science Workshops, pp.59-66, 2009.
[9] B. Tovar, A Job Sizing Strategy for High-Throughput Scientific Workflows, IEEE Trans. On Parallel and Distributed Systems, vol.29, no.2, pp.240-253, 2018.
[10] I. Sfiligoi, Estimating job runtime for CMS analysis jobs, Proc. of J. Physics: Conf. Series, vol. 513, no. 3, 2014.
[11] Y. Manabe, Resource provisioning method for scientific workflows on commercial cloud services, graduation thesis, Doshisha University, 2017.
[12] Sysbench,http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf, 2009.
[13] UnixBench, http://code.google.com/p/byte-unixbench/.
[14] Hadoop, http://hadoop.apache.org.