Proposal for an Optimal Job Allocation Method for Data-intensive Applications based on Multiple Costs Balancing in a Hybrid Cloud Environment

Yumiko Kasae[1] and Masato Oguchi[1]
[1]*Ochanomizu University, 2-1-1 Ohtsuka, Bunkyo-ku, Tokyo 112-8610, Japan*
yumiko@ogl.is.ocha.ac.jp, oguchi@computer.org

Abstract: Due to the explosive increase in the amount of information in computer systems, we need a system that can process large amounts of data efficiently. Cloud computing system is an effective means to achieve this capacity and has spread throughout the world. In our research, we focus on hybrid cloud environments, and we propose a method for efficiently processing large amounts of data while responding flexibly to needs related to performance and costs. We have developed this method as middleware. For data-intensive jobs using this system, we have created a benchmark that can determine the saturation of the system resources deterministically. Using this benchmark, we can determine the parameters in this middleware. This middleware can provide Pareto optimal cost load balancing based on the needs of the user. The results of the evaluation indicate the success of the system.

Keywords: Hybrid cloud, Load balancing, Data processing, Performance, Cost Balance

1 Introduction

In recent years, large amounts of data, referred to as big data, have become more common with the development of information and communications, creating the need for efficient data processing. As a platform for processing these data, hybrid cloud environments have become a focus of attention. In hybrid cloud environments, users can access public clouds and private clouds; private clouds are secure clouds built using the secure resources of the user company, and public clouds can provide scalable resources if the user pays metered rates. Combining these clouds can address shortcomings related to safety and scalability. For data-intensive jobs, hybrid clouds are appropriate. For increasing amounts of data, hybrid clouds can provide secure and scalable processing.

However, performance and costs must be balanced. When we want to process large amounts of data more rapidly, using many resources that are provided by public clouds, in addition to those provided by private clouds, will increase speed, but the metered cost will also be greater. In contrast, if these jobs are processed using private cloud resources almost exclusively, users will not have to pay metered rates, but the job execution time will be longer. Thus, we need a system that can determine optimal job placement based on cost limitations and necessary performance to ensure efficient processing in hybrid cloud environments.

Therefore, in this research, we proposed a method for providing optimal job placement in hybrid cloud environments in terms of monetary costs and performance. We have developed this system as middleware. In addition, the middleware provides optimal job placement for both CPU-intensive applications and data-intensive applications. In general, unlike in CPU-intensive applications, which can accurately determine the load using the CPU usage, efficient resource use in data-intensive applications is difficult to determine. In the proposed method, we created a benchmark that can be used to change the extent of the load of CPU processing and I/O processing, and we measure the performance of hybrid clouds as an execution environment using this benchmark. Based on the results obtained using this benchmark, we propose a method of determining job execution status based on the status of the I/O resources.

In this paper, we will describe the details of the middleware that can be used to implement the method

proposed in this study. We have evaluated the balance of performance and costs by using this middleware with data-intensive applications. We examine the evaluation axis for performance and monetary costs and show that this middleware can provide optimal job placement for efficient job processing. The monetary cost is the sum of the power consumption cost for private clouds and the metered costs associated with public clouds.

The remainder of this paper is organized as follows. Section 2 introduces cloud computing. Section 3 describes the proposed method of determining the load. Section 4 describes the middleware that we suggest can be used for optimal job allocation. Section 5 introduces the evaluation results for our middleware. Section 6 comments on related research studies, and Section 7 presents concluding remarks and suggestions for future work.

2 Cloud Computing

2.1 Overview of Cloud Computing and Classification

Cloud computing is a service through which users can use necessary software and hardware resources from servers through networks. If a user uses cloud computing services, without having the physical computer resources, the user can receive various services.

The types of services include SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service). Recently, because of the diversity of services that can be provided, these services have been collectively called XaaS (X as a Service).

This study will consider IaaS. The types of platforms for IaaS are private clouds and public clouds. Public clouds can be used through the Internet, and users can use cloud services scalable if they pay metered rates to the cloud provider. However, in public clouds, it is necessary to leave the data with the cloud provider (albeit temporarily) during processing jobs, which generates some security concerns.

Using private clouds can solve these problems. A private cloud is a cloud that is built using resources that users already have. The user can construct the cloud taking security into account. However, private clouds lack scalability relative to public clouds. Hybrid clouds can address the shortcoming each of the cloud types. These clouds can be both secure and scalable. In this research, which is focused on hybrid cloud environments, we proposed a method for ensuring efficient processing.

2.2 The Trade-off Between Cost of the Evaluation Axis for Hybrid Clouds

When we use hybrid cloud environments, there will be a trade-off relationship between performance and necessary costs. When we want to process a large amount of data more rapidly, using many resources that are provided by the public cloud, in addition to the resources of the private cloud, and the associated metered cost will make the job more expensive. In contrast, if these jobs are processed using private cloud resources with little or no use of public cloud resources, users will not have to pay metered costs, but the job execution time will be longer. Thus, we need a system that can determine optimal job placement based on the equilibrium between necessary cost and performance to ensure efficient processing in a hybrid cloud environment.

My research proposes a method of providing optimal job placement in hybrid cloud environments in terms of monetary costs and performance. This method has been operationalized as middleware. This middleware will consider job processing time and monetary costs. Monetary cost is the sum of the charge for power consumption in private clouds and the metered costs associated with public clouds. The recent environmentalism in global affairs makes it especially important to reduce power consumption when processing large amounts of data. It is important that we not waste power. In addition, the monetary cost of private clouds may include fixed costs associated with the system installation. However, such amounts are difficult to define categorically. Thus, we assume that the equipment has already been depreciated, and the fixed costs were evaluated as zero. The monetary cost does not include the fee for the power consumption associated with the public cloud. This is because it is difficult for the user to know the price of the power consumption by each resource in the public cloud. For the public cloud, the power consumption charges are assumed to be included in the metered costs.

2.3 Eucalyptus

In this paper, we have used the cloud-building software Eucalyptus (D.Nurmi, 2010) to build two cloud systems. By connecting with Dummynet to generate an artificial delay between them, we have built an emulated hybrid cloud environment in our laboratory. Eucalyptus is open source software that can create cloud infrastructure. Eucalyptus is compatible with the Amazon EC2 API; the Amazon EC2 (Amazon Elastic Compute Cloud) is a cloud service that is

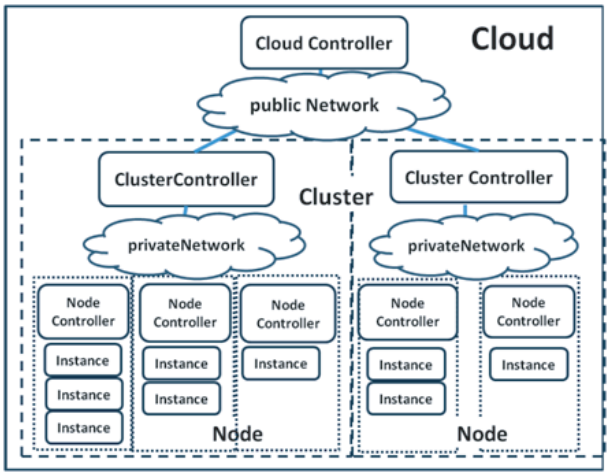

Figure 1: Architecture of Eucalyptus.

provided by the U.S. company Amazon.com. Using a cloud built in Eucalyptus, you can port a service on this cloud as if the service was on Amazon EC2. Figure 1 shows the architecture of Eucalyptus. Eucalyptus is composed of three components. It is treated as a public network to the upper layers of the Cloud Controller (CC) from the Cluster Controller (CLC), and it is treated as a private network to the lower layers of the NC (Node Controller).

Cloud Controller (CLC)
Manages the information in the entire cloud. Equipped with a compatible interface for Amazon EC2; a web management screen provides an API for the user.

Cluster Controller (CC)
Manages the node controller, the state of instances (virtual machines) and the virtual network for the instances.

Node Controller (NC)
Controls the instance. When the program needs to run multiple instances, the virtualization software runs on the node controller.

3 Proposed method for determining the load

The proposed middleware in this paper processes not only CPU-intensive applications but also data-intensive applications. In both these jobs in a hybrid cloud environment, to obtain high-speed and low-cost processing after all of the resources have been used in the private cloud, the next tasks should be processed using in the public cloud. In addition, in the public cloud, after the borrowed resources have all been used, new resources will be needed. Therefore, even when it is important to utilize resources without waste, it is also important to properly determine the load, and all of each resource should be used. Therefore, when used for CPU processing and disk processing, this middleware determines when the resource has been saturated. Based on this information, this middleware will determine the resource load. The methods of determining the load for each type of processing are as follows.

3.1 Method for Determining the Load of CPU Processing

Load balancing for CPU-intensive jobs has been investigated in many past studies. In this research, CPU usage is the focus. This proposed middleware also determines the load based on CPU usage. This method is same as that used in other studies; if the usage reaches 100%, the resource has been saturated, and the middleware does the load balancing. The method of optimally balancing CPU-intensive jobs is not a feature of this proposal because it does not fundamentally change the techniques used in other studies.

3.2 Method of Determining the Disk Processing Load

3.2.1 Disk Performance Measurement

Unlike in CPU intensive-jobs, it is difficult to make a definitive decision about whether the disk load has reached the saturation point during data-intensive jobs. For data-intensive jobs, because the system is often waiting for I/O processing, it is difficult to determine the CPU load. Thus, in the proposed method, we use each cloud resources
/proc/diskstats file to obtain the length of the queue for the current disk. Then, we estimate the number of jobs that are running in these disks. Therefore, in this method, it is also necessary to know the length of the queue, which indicates the saturation of the disk resources.

Therefore, we have created benchmarks that can change the balance of I/O and CPU processing. By using the benchmark Disk Bench, which performs read-only processing, we measured disk performance using the execution environment of the middleware. In Figure 2, as an example of a job by Disk Bench, we can see a state of transition for the CPU load and the number of disk accesses. Disk Bench is a simple benchmark that performs Read processing for the disks in the instance. This figure shows that Disk

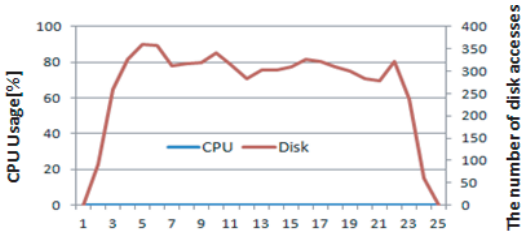

Figure 2: One example of the load transition of Disk Bench.

Table 1: Instance.

OS	Linux 2.6.27.21-0.1-xen / x86_84 GNU / CentOS 5.3
CPU	Intel(R) Xeon(R) CPU @ 3.60GHz 1 core
Memory	1024MByte
Disk	20GByte

Bench processing is not performed when there is little CPU processing and will become I/O bound if many jobs are processed at the same time.

Using Disk Bench, we have measured the performance of the disk. We have made this performance measurement for the instances of performance in Table 1 for the hybrid cloud. In this measurement process, we measured the execution time for the jobs and the queue that is accumulated for the disk during the processing of multiple simultaneous jobs using Disk Bench. We then compare the processing time when these jobs are processed sequentially and the processing time using this measurement.

In general, if there are sufficient disk resources, simultaneously processing the jobs can be more rapid than sequentially processing them. However, when we increase the number of jobs to be processed simultaneously, there is a point at which processing time will be slow as a result. This method determines when there are no more disk resources, and the length of the queue that has accumulated for the disk at that time is defined as the ”conditions in which the disk resources have run out”.

Furthermore, in Disk Bench, there are two parameters. One of parameters specifies the amount of reading at a particular time, and the other specifies the number of times this reading has been repeated. In this performance measurement process, we create a job that we intended to access a variety of patterns using the disk; these parameters were varied. We have measured performance changing these parameters.

Figures 3 and 4 show examples of the comparison results for processing times and the length of the queue at the time of this performance measurement experiments.

The assumed access pattern is as follows: a few read small data blocks, many read small data blocks,

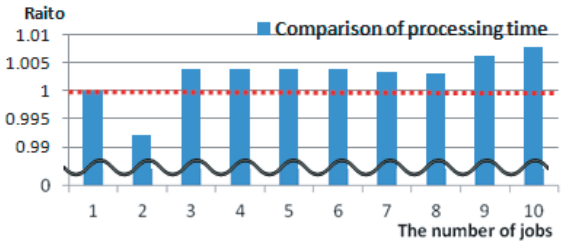

Figure 3: One Example of Comparison of processing times(block size: 64 bytes, repetition rate: 4M times).

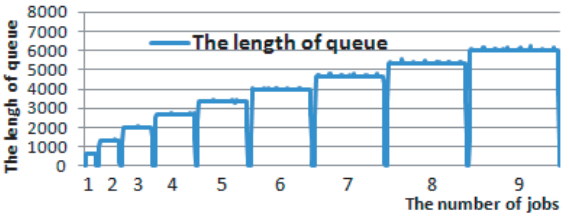

Figure 4: One Example of The length of the queue(block size: 64 bytes, repetition rate: 4M times).

a few read large data blocks and many read large data blocks.

First, in this performance measurement process, we compare the processing time for simultaneous processing and sequential processing. In Figures 3, the vertical axis represents a ratio that indicates the comparison results for the processing time. This ratio was obtained by dividing the sequential processing time into the simultaneous processing time. The vertical axis at the value of 1 is indicated by a red dashed line. If the value is below the dashed line, then simultaneous processing is faster than sequential processing. If the value is above the dashed line, then sequential processing is faster than simultaneous processing. In other words, the disk resource has been exhausted. Thus, in Figure 3, we can see that in job number 4, the disk resource was exhausted.

Figures 4 show the transitions in the length of the queue for each number of concurrent jobs at this measurement. As shown in this figures, if the number of concurrent jobs is increased, the length of the queue is increased. For this state, we can use the following queuing model: The disk access requests from multiple jobs arrive at random, the processing time for the job is nearly constant, block size is constant and the window for each disk is one. Therefore, the degree of congestion of I/O requests from the job, that is, the length of the queue, accurately reflects the degree of saturation of the input and output.

By analyzing the relative processing time and the length of the queue with some parameter setting, in this experimental environment, we found that the lengths of the queues are between 2000 and 2700

when the disk resources run out. However, clearly, this is a range of values. If we analyze the physical disk in detail, these values may be uniquely determined. However, in general, the accuracy of the actual job will not be exact. Therefore, in this method, we determine this range as the saturated disk load. We discuss our preliminary experiments in the next section.

3.2.2 Preliminary Experiments: Experiment in Controlling Load Balancing

In our preliminary experiments, we process data-intensive jobs using Disk Bench in this hybrid cloud environment. In this experiment, as our threshold for load balancing, we use the length of the queue for the disk resource. Using this threshold, by choosing a value in the range of values determined in the performance measurements, we have examined the evaluation of performance and cost.

In this experiment, we received Disk bench jobs every 2 seconds 100 times. These experiments were load balancing experiments intended to determine where to place jobs: whether in private clouds or public clouds. For this experiment in hybrid cloud environments, we ran 8 instances with performance as indicated in Table 1: 4 instances for each cloud. In addition, this range of values for the length of the queue (which was determined by measuring performance) depends on the physical machine. However, because all of the physical servers built as hybrid cloud environments had the same performance, this range is unified at the above-mentioned value.

First, jobs are placed in one instance in a private cloud. If the length of the queue for that instance is equal to or greater than the threshold value, the next jobs will be distributed under the conditions that the length of the queue is less than this threshold or that has not been used within the private cloud. If all of the queue lengths in a private cloud are equal to or greater than the threshold value, the public cloud begins to be used. Then, the next jobs are similarly distributed until the queue length is greater than or equal to the threshold.

In this experiment, as the threshold for load balancing, we varied this value from a small value to large value from 500 to 12500. By using a value within the range of values obtained in the performance measurement process, we verified whether this load distribution could provide an optimal balance between monetary costs and performance as described in Section 2.2. During this experiment, we measured the processing time for the jobs, the power consumption rate for the private cloud and the metered rates for the public cloud. To measure power consumption

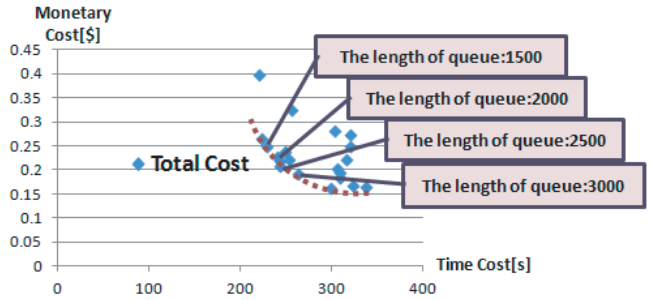

Figure 5: Evaluation results for Experiment in Controlling Load Balancing.

in this environment, we have used a watt-hour meter SHW3A, which is a high-precision power meter produced by the System Artware Company in Japan. After one plugs an electric product into the SHW3A, the power consumption is instantly measured and displayed. In this study, we measure only the private cloud's node power consumption.

Figure5 shows the evaluation results for the experiment.

The horizontal axis is the processing time cost, and the vertical axis is the monetary cost. Monetary costs are calculated using the following equation:
Monetary Cost : $T_R * N_R * C_R + P_L * C_L$

T_R:Execution time for Public Cloud[hour]
N_R:Number of Instances of Use of Public Cloud
C_R:Charges for Public Cloud Use[$/hour]
P_L:Power Consumption in a Private Cloud[kWh]
C_L:Charges for Power Consumption in a Private Cloud[$/kWh]

In this evaluation, the metered unit price is $0.5 based on the price of Amazon EC2 and the unit price of power consumption is set at $0.24 based on the price charged by the Tokyo electric power company.

As shown in Figure 5, there is no configuration that optimally balances both time costs and monetary costs. However, in selecting a threshold for load balancing, if we choose a value from 2000 to 2700 based on the performance measurement that indicates the saturation of the disk resource, we find that load balancing can be provided based on a Pareto optimal cost balance. In other words, if we set the threshold near 2000, although efficiency will be ensured and the load balancing will occur quickly, the monetary costs will increase slightly. In contrast, if we set the threshold near 2700, while efficiency will be ensured, it will take a little time to perform load balancing and ensure a low monetary cost. The balance of time costs and monetary costs should be based on the needs of the user.

Thus, in these preliminary experiments, we could not find a point that best balances time cost and mone-

tary cost because there is a range in which the disk resource is exhausted. However, by setting a threshold value in response to a user request within this range, we found that a processing cost balance can be obtained without wasting resources.

3.2.3 Method of Controlling the Load Distribution in Disk Processing

Based on the discussion in Sections 3.2.1 and 3.2.2, in the proposed method of load determination for disk processing, first, by measuring the performance of the disk, we determine the range for queue length that indicates disk saturation. This phase is regarded as a learning phase. The threshold for load balancing in middleware is the length of the queue for the disk resource, and the user can select a threshold within that range, which is determined by the performance measurement process. This middleware can be used to control the Pareto optimal cost balance load distribution without wasting resources.

4 The Pareto Optimal Job Allocation Middleware

4.1 The Structure of the Middleware

Figure6 shows the behavior of the middleware. This middleware consists of a dispatch unit and monitor unit. The monitor unit in this middleware (for instance, in hybrid clouds) uses the priority of job placements to check the status of the resource on a regular basis. As mentioned in the previous section, to check the status of the resources requires measuring CPU utilization for CPU processing and the length of the queue for disk processing. In addition, the middleware evaluates the load status of these resources, determining CPU utilization and disk processing at the same time, and if the processing becomes saturated, the middleware determines that. The dispatch unit receives and distributes jobs based on the information from the monitor unit.

4.2 An Algorithm for Middleware

This middleware algorithm is as follows. Additionally, when running this middleware in a hybrid environment cloud for the first time, as mentioned in Section 3.2.3, you must determine the range of the queue length to identify disk resource saturation.

1. Based on the range of threshold values determined in the learning phase, the user sets the threshold for load balancing, which can be used to obtain the desired cost balance, and runs the middleware.

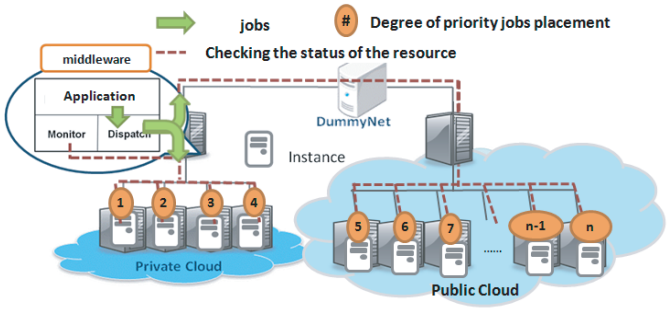

Figure 6: Behavior of this middleware.

2. Middleware receives the submitted job.

3. In order of placement priority in private cloud instances, check the load state of the resource to determine whether it is greater than or equal to the threshold. If the resource is at a value that is less than the threshold value, execute the job using that instance, and then return to (2). If the load states of all resources in the private cloud are equal to or greater than the threshold value, go to (4).

4. In order of placement priority for public cloud instances, check the load states of the resources to determine whether they are greater than or equal to the threshold. If the load state is less than the threshold value, execute the job using that instance; then return to (2). If all of the resource load states are equal to or greater than the threshold value at that time, proceed to (5).

5. In the public cloud, select a new instance and execute the submitted job using that instance; then return to (2).

5 Experiments using this middleware

In this chapter, we describe examples of the results of evaluations conducted using this middleware. Conducting load balancing experiments using this middleware for CPU-intensive applications in hybrid cloud environments is not fundamentally different from the process used in earlier studies of load balancing. Therefore, in these examples, we consider the evaluation results obtained for data-intensive applications using this middleware.

5.1 Overview of experiments

As shown in Figure 6, as the experimental environment for this middleware, we have built a hybrid cloud environment. The node servers that constitute each cloud are single-core CPUs, and all servers have the same performance. For this reason, we will generate one instance of performance from the table 1 for each node server. There are 4 instances in the private cloud. If all instances are saturated, we conduct load balancing using the public cloud resources.

The experiment procedures are as follows. First, in a learning phase, we measured the performance of the disk. However, this experimental environment is the environment in which the performance measurement was carried out in Section 3.2. Therefore, for all instances in this hybrid cloud, the queue length range that indicates disk saturation is between 2000 and 2700. Next, based on this range, we execute this middleware by varying the value of the threshold for load balancing. During these experiments, we measured the processing time for the jobs, the cost of power consumption when the private cloud was used and the metered rates for the public cloud.

In this experiment, we evaluated these three types of costs by varying the threshold for load balancing. In particular, setting the threshold in the range determined by the performance measurement process, we evaluated whether Pareto optimal load balancing is possible without wasting resources.

5.2 Data-Intensive Applications that were used in the Experiment

In these two experiments, we have evaluated middleware used with two different data-intensive applications.

As the first, we used pgbench, which is the PostgreSQL benchmark. Pgbench is a simple tool benchmark that is bundled with PostgreSQL. Tatsuo Ishii created the first version, published in 1999 by the PostgreSQL mailing list in Japan. Pgbench was created based on the TPC-B, which mimics the online transaction process and can measure the number of transactions that can be processed per unit of time. We received pgbench's jobs every two seconds 200 times; the middleware processed these jobs.

For the second, we used queries from DBT-3. DBT-3 is a simplified version of the TPC-H and performs complex select statement queries in large databases. The TPC-H and DBT-3 are decision support benchmarks and consist of ad-hoc queries and concurrent data modifications. In the DBT-3, there are 22 search queries. However, because the processing time may be long because of the number of queries, in this experiment, we select 11 queries for shorter processing times. Then, we submitted these queries repeatedly for a total of 110 jobs, and the middleware processed the jobs. The DBT-3 database was built using MySQL.

The major difference between these two types of data intensive applications is the difference in the CPU processing load. In executing pgbench jobs, we confirmed that CPU processing is generally not performed. In contrast, the search queries for DBT-3 were processed to some extent with the CPU. However, all of the search queries were mainly executed using disk processing; thus, these are data-intensive applications. For each of these two data-intensive applications, using this proposed method, we show that the method does not depend on the nature and type of application.

5.3 Data Placement in Experiments

In a cloud environment, especially for data-intensive jobs, considering data placement is very important. For data placement in a hybrid environment cloud, we can consider using the block storage associated with each cloud or using remote access to local storage from the public cloud. In these experiments, it is assumed that due to remote backup, the necessary data are already located in some instances. In (S.Toyoshima and M.Oguchi, 2011), using middleware, remote access to the local storage is attained using iSCSI from a public cloud. We wish to consider this method of data placement in the future.

5.4 Evaluation of the Results obtained using Middleware

5.4.1 An example of the use of pgbench

Figure 7 shows the results of the cost evaluation obtained using the middleware, which processed pgbench's jobs.

In Figure 7, as in Section 3.2.2, the vertical axis shows the monetary cost, and the horizontal axis shows the time cost. In addition, figure 7 shows details of the representative points for the load balancing threshold.

As we can see from this figure, if we set the correct threshold at the relevant queue length based on the performance measurement process (i.e., between 2000 and 2700), this middleware provides a Pareto-optimal cost balance and uses resources efficiently. Conversely, some points are on a Pareto-optimal curve even though they are out of the range of

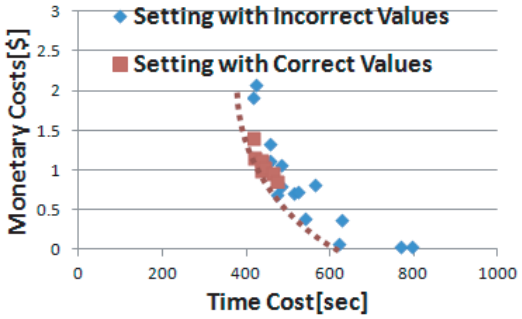

Figure 7: Cost Evaluation of processing pgbench Jobs.

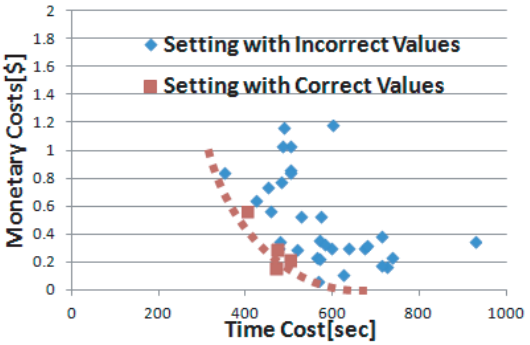

Figure 8: Cost Evaluation of processing DBT-3 queries.

values determined by the performance measurement process. These points are examples that indicate when the load balance is too focused on processing performance and too many resources are used or when a tremendous burden has been placed on the available resources so as not to raise the monetary cost. For points that are not listed on the Pareto optimal curve and that for values other than the threshold value, it is possible that a better cost balance exists.

Based on the above, we found that it is a necessary condition for the point on the Pareto optimal curve to set the threshold for load balancing based on the saturation of the disk resource.

5.4.2 Example using search queries of DBT-3

Figure 8 shows the results of the cost evaluations obtained using the middleware with DBT-3 processing queries.

In Figure 8, as in Section 3.2.2, the vertical axis shows the monetary cost, and the horizontal axis shows the time cost. In addition, Figure 8 shows the details of the representative points only for set values for the load balancing threshold.

From this figure, as well as the DBT-3 processing queries, we can see that if we set the correct threshold at the queue length determined by the performance measurement process (i.e., from 2000 from 2700), this middleware provides a Pareto-optimal cost balance in which resources are used efficiently. Additionally, in other respects, results similar to the ones found using pgbench are obtained.

5.4.3 Observations from these Experiments

Based on the evaluation results for the processing search queries for DBT-3 and pgbench as examples of data-intensive applications, we can conclude that this middleware provides a Pareto-optimal cost balance while using resources efficiently when we set the threshold to the queue length determined by the performance measurement process.

In these experiments, we deliberately added a delay of 20 msec by using Dummynet between the clouds. However, because only certain jobs are transferred to the remote cloud in this middleware, some of the evaluations were barely influenced by the differences in the delay time. In addition, because the unit price of the metered cost for public cloud use was large, the influence of the differences in power consumption in the private cloud was limited. However, for technical and social reasons, these monetary costs may vary significantly. Even when the proposed method is used, when the charge for power consumption is more significant, this factor must be kept in mind during load balancing. However, we can make this modification by simply changing the cost calculation expression.

6 RELATED WORKS

Previous researchers have discussed load balancing in cloud computing (G.JungK. R.Joshi and C.Pu, 2008) and (E.Kalyvianaki and S.Hand, 2009). In these papers, however, CPU-intensive applications were used as the targets of load balancing jobs rather than data-intensive applications. In computing-centric applications, similarly to some scientific calculations, it is possible to perform appropriate load balancing based on the CPU load of each node. In this research, however, we have used a data-intensive application for the jobs. In such cases, because the CPU is often in the I/O waiting state, load balancing is almost impossible based on CPU load. In this research, we have used the disk I/O as a load indicator. In data-intensive applications, load balancing middleware has also been developed that uses the amount of disk access for load decisions (S.Toyoshima and M.Oguchi, 2011). This middleware based on disk access provided dynamic load

balancing between public clouds and a local cluster and ensured optimal job placement. Because we have further developed middleware by introducing user-specified parameters, it will be possible to reduce the monetary costs of load balancing, including the cost of power consumption.

Power saving in cloud computing has also been actively investigated. Unlike in this study, researchers have discussed an approach to power saving that involves the use of CPU-intensive applications in a cloud (Che-Yuan Tu, 2010). Other study (C.; Parr, 2011) examined power saving efforts for a cloud datacenter. Our study aims to save power in all clouds, including private clouds. Researchers (Zhang, 2010) proposed a scheduling algorithm that could be used to evaluate power consumption and job execution time. However, these studies differ from our study, especially because we have focused on total costs in hybrid clouds, including job execution time, public cloud charges at a metered rate, and power consumption charges for private clouds. In addition, we have used data-intensive applications as the target jobs.

7 Conclusions and Future works

We proposed a method of determining the load based on the required cost and performance to ensure efficient processing load balancing in a hybrid environment cloud. We have implemented this procedure using middleware. This middleware uses information about CPU processing and disk processing to provide efficient load balancing if the resources needed to perform a job are scarce. In particular, in the proposed method, we determine the load from CPU usage and the length of the queue for disk processing. First, in a learning phase, by measuring the performance of the disk, we determine the range of queue lengths that indicate disk saturation. The user can select a threshold within that range, which is determined by the performance measurement process. Using this middleware, tone can control the Pareto optimal cost balance load distribution without wasting resources.

Future research should be focused on improving data placement. In the experiments in this paper, data placement was the task of interest. This data placement is not realistic as a model for real situations. A realistic model might aggregate local and remote storage and synchronize these forms of storage. Therefore, we are considering introducing network storage. In our system, iSCSI has already been introduced, and we plan to carry out an experiment using iSCSI in the future. In addition, in the current implementation, based on the values obtained in the learning phase, the threshold should be set for load balancing before running the middleware. In the future, we would like to develop an automating learning phase as a part of this middleware.

ACKNOWLEDGEMENTS

This work is partly supported by the Ministry of Education, Culture, Sports, Science and Technology, under Grant 22240005 of Grant-in-Aid for Scientific Research. The authors would like to thank to Drs. Atsuko Takefusa, Hidemoto Nakada, Ryousei Takano, Tomohiro Kudoh at the National Institute of Advanced Industrial Science and Technology (AIST), Project Associate Professor Miyuki Nakano, Assistant Professor Daisaku Yokoyama, Senior Researcher Norifumi Nishikawa at the Institute of Industrial Science, the University of Tokyo, and Associate Professor Saneyasu Yamaguchi at Kogakuin University for the conscientious advice and help with this work.

REFERENCES

C.; Parr, G.; McClean, S. (2011). Energy-aware data centre management. pages 1–5. Communications (NCC), 2011 National Conference.

Che-Yuan Tu, Wen-Chieh Kuo, W.-H. T. Y.-T. W. S. S. (2010). A power-aware cloud architecture with smart metering. pages 497–503. Parallel Processing Workshops (ICPPW), 2010 39th International Conference.

D.Nurmi, R.Wolski, C. G.-S. L. D. (2010). The eucalyptus open-source cloud-computing system. pages 62–73. Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference.

E.Kalyvianaki, T. and S.Hand (2009). Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In Proc. 6th International Conference on Autonomic Computing and Communications (ICAC2009).

G.JungK. R.Joshi, M.A.Hiltunen, R. and C.Pu (2008). Generating adaptation policies for multi-tier applications in consolidated server environments. pages 23–32. In Proc. 5th IEEE International Conference on Autonomic Computing (ICAC2008).

S.Toyoshima, S. and M.Oguchi (2011). Middleware for load distribution among cloud computing resource and local cluster used in the execution of data-intensive application. *DBSJ Journal, Vol.10, No.1*.

Zhang, L. M. Z. K. L. Y.-Q. (2010). Green task scheduling algorithms with speeds optimization on heterogeneous cloud servers. pages 76–80. Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on and Int'l Conference on Cyber, Physical and Social Computing (CPSCom).