Cluster Computing (2021) 24:181–193
https://doi.org/10.1007/s10586-020-03191-2

OMBM-ML: efficient memory bandwidth management for ensuring QoS and improving server utilization

Hanul Sung[1] · Jeesoo Min[1] · Donghun Koo[1] · Hyeonsang Eom[1]

Received: 2 January 2019 / Revised: 29 September 2020 / Accepted: 11 October 2020 / Published online: 1 February 2021

Abstract
As cloud data centers are dramatically growing, various applications are moved to cloud data centers owing to cost benefits for maintenance and hardware resources. However, latency-critical workloads among them suffer from some problems to fully achieve the cost-effectiveness. The latency-critical workloads should show latencies in a stable manner, to be predicted, for strictly meeting QoSs. However, if they are executed with other workloads to save the cost, they experience QoS violation due to the contention for the hardware resources shared with co-location workloads. In order to guarantee QoSs and to improve the hardware resource utilization, we proposed a memory bandwidth management method with an effective prediction model using machine learning. The prediction model estimates the amount of memory bandwidth that will be allocated to the latency-critical workload based on a REP decision tree. To construct this model, we first collect data and train the model with the data. The generated model can estimate the amount of memory bandwidth for meeting the SLO of the latency-critical workload no matter what batch processing workloads are collocated. The use of our approach achieves up to 99% SLO assurance and improves the server utilization up to 6.8× on average.

Keywords Latency-critical big data workload · Batch processing big data workload · Cloud Service · SLO · QoS · High server utilization · Memory bandwidth management · Shared resource isolation

1 Introduction

Cloud data centers have been growing because the cost of computation and storage is lower compared to traditional data centers. Therefore, many big data applications including grid and HPC system ones have been run in the cloud data centers. In addition, service providers for latency-critical workloads such as Web and interactive applications have recently been using data centers. The cloud service providers need to be aware of the different requirements of the applications in their new systems and then manage the resources accordingly [9, 10]. Especially, the latency-critical workloads require strict Quality of Service (QoS) guarantees compared to batch processing ones such as large scientific applications. However, it is difficult to meet Service Level Objectives (SLOs) due to shared resources such as cores, cache, and memory controller. Because the contention for the shared resources causes performance interference, the latency-critical workloads experience unexpected tail latencies when they are executed with co-located ones. Therefore, the cloud service providers allow the latency-critical workloads to exclusively use the hardware resources, resulting in low hardware resource utilization of the data center, which is only 10% to 45% [5].

There have been several studies to improve the server hardware utilization while meeting the SLOs [3, 4, 6, 24, 25, 35]. Most of the studies proposed solutions that isolate only the cores and Last Level Cache (LLC) [29, 31] physically for each workload to eliminate the performance interference for strictly meeting the QoSs.

✉ Hyeonsang Eom
hseom@cse.snu.ac.kr

Hanul Sung
husung@dcslab.snu.ac.kr

Jeesoo Min
jsmin@dcslab.snu.ac.kr

Donghun Koo
dhkoo@dcslab.snu.ac.kr

[1] Department of Computer Science and Engineering, Seoul National University, Seoul, Republic of Korea

However, in one of our prior studies, Optimized Memory Bandwidth Management (OMBM) [28], it was found that the contention for the memory controller as well as the core and LLC is significant. OMBM thus eliminates the contention by allocating the isolated memory bandwidth to the latency-critical workload and the batch processing ones. For strictly meeting the SLOs in every possible situation, OMBM predicts the amount of bandwidth for the latency-critical workload under the assumption that the contention for the memory controller is the worst. Therefore, OMBM always allocates the maximum memory bandwidth to the latency-critical workload for meeting the SLOs without considering the current memory contention. In other words, OMBM also wastes hardware resources.

Figure 1 shows a prediction line made in OMBM and actual tail latencies of a latency-critical workload, silo, by collocating A and B which are different combinations of the batch processing workloads. The prediction line is created under the assumption that the memory contention is the worst, and is used to predict the tail latency corresponding to the amount of memory bandwidth. As the graph shows, the actual tail latency values with both combinations, A and B, are less than those of or near the line. For this reason, OMBM meets the SLOs well, but it has the limitation of saving the memory bandwidth for high server utilization. Due to a lack of hardware-based memory bandwidth isolation techniques, OMBM isolates the memory bandwidth by using a software-based technique. The technique cannot provide physically isolated memory bandwidth, and thus the co-executing batch processing workloads affect the performance of the latency-critical workload depending on their memory usage patterns. For example, because in Combination B, the memory controller is accessed frequently enough to make the memory controller contention nearly as bad as possible, the actual

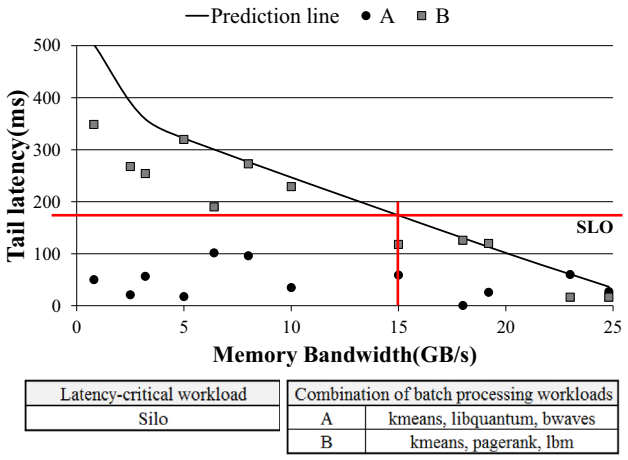


Fig. 1 Prediction line and the actual tail latencies when OMBM is applied

latency values in B are close to the prediction line. In contrast, in Combination A, the contention for the memory controller is not as much as in B, and thus the latency values are far from the prediction line. Although the SLOs can be met with a smaller amount of memory bandwidth, the memory bandwidth is wasted due to the worst contention assumption. Consequently, OMBM cannot consider the amount of stress which the batch-processing workloads put onto the memory controller.

In order to address this problem, we propose a new memory bandwidth management method including a new prediction model, Optimized Memory Bandwidth Management-Machine Learning (OMBM-ML). OMBM-ML is not only meeting the SLO but also improving the server utilization efficiently. OMBM-ML predicts the amount of memory bandwidth for the latency-critical workloads based on a model constructed by using a machine learning algorithm. The prediction model of OMBM-ML reflects the degree of dynamic memory contention stress caused by the batch processing workloads rather than assuming that the memory contention is the worst. By applying the proposed model, the cloud service provider met up to 99% SLOs of the latency-critical workloads and improved the server resource utilization up to 6.8× on average.

In the rest of this paper, Sect. 2 introduces the background and the machine learning model we utilized. Section 3 describes the new proposed model in OMBM-ML. Section 4 shows the experiment results to demonstrate how effective our proposed model is compared to other models. Section 5 discusses related studies. Finally, Sect. 6 concludes the paper.

2 Background and motivation

This section briefly explains the prior work, OMBM, and its limitation. Then, we will discuss the new model to overcome the limitation.

2.1 Prior work

Prior work proposed a memory bandwidth management scheme, OMBM, to ensure the QoS and improve the hardware utilization. OMBM predicts the amount of the memory bandwidth that is needed for meeting the SLOs based on pre-profiling results. OMBM utilizes Memguard [34] which is a memory bandwidth reservation system to isolate the memory bandwidth. However, the system is implemented with a CPU scheduling in the operating system, so the memory bandwidth cannot be isolated physically. Therefore, even if OMBM provides isolated memory bandwidth to the latency-critical workload and the batch processing ones, the latency-critical workloads show

unstable tail latencies depending on the collocated batch processing ones. In order to meet SLOs strictly in all possible situations, the pre-profilings are executed under an assumption that the contention for the memory controller is the worst. Then, OMBM provides the predicted amount of the memory bandwidth to the latency-critical workload and the rest of the memory bandwidth to the batch-processing workloads.

In Fig. 1, the horizontal line is the value of SLO and the value of the x-axis where the SLO overlaps the prediction line indicates the predicted amount of the memory bandwidth. If the predicted amount of the memory bandwidth is allocated to the latency-critical workload, the expected tail latency should be close to the prediction line. However, there is a big difference between the actual tail latencies and the expected one. This is because the collocated batch processing workloads less stress the memory controller than the worst assumption. In this case, the service provider does not need to allocate the whole predicted memory bandwidth to the latency-critical workload. Therefore, we need a more effective prediction model that predicts the amount of memory bandwidth considering the stress degree of the batch processing workloads to the memory controller.

2.2 Machine learning model

To address the limitation of prior work, it is important to find out a method that can predict the memory bandwidth depending on the memory controller contention caused by the collocated batch processing ones.

We propose the memory bandwidth management method by using machine learning, OMBM-ML. The machine learning model has the advantage of identifying the complex relationships of the various factors that affect the attribute we are predicting and filtering out the non-critical factors. In other words, the machine learning model can find out the relationships between the tail latencies and various degrees of the memory controller contention. To make the new model, we collect data and train the model based on collected data. By doing so, OMBM-ML can make an effective and accurate model for managing memory bandwidth efficiently,

3 OMBM-ML

This section describes an overview of OMBM-ML and how OMBM-ML creates the model and predicts the amount of memory bandwidth for ensuring the SLOs of the latency-critical workloads.

An overview of the proposed OMBM-ML is shown in Fig. 2. OMBM-ML consists of four parts, a *offline collector*, a *offline trainer*, a *online predictor* and a *online allocator*. Since OMBM-ML is an independent system for each node, they have their own model and independently predicts the amount of memory bandwidth. Thus, it is good to be used for large-scale workloads in multi-nodes and shows high scalability.

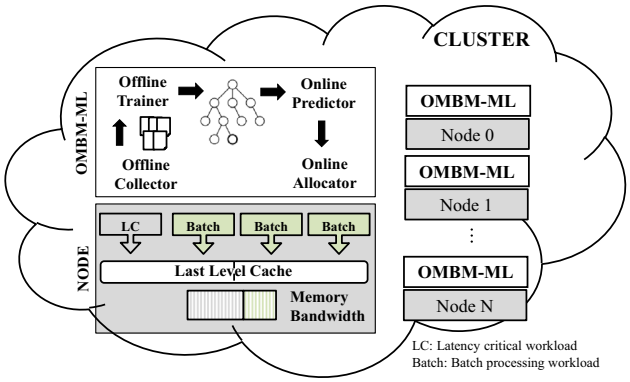


Fig. 2 Overview of the proposed OMBM-ML

Figure 3 shows processing flows of *OMBM-ML*. The *offline collector* collects profiling data in order to construct a model that predicts the amount of memory bandwidth for meeting SLO. It collects the tail latencies by adjusting the

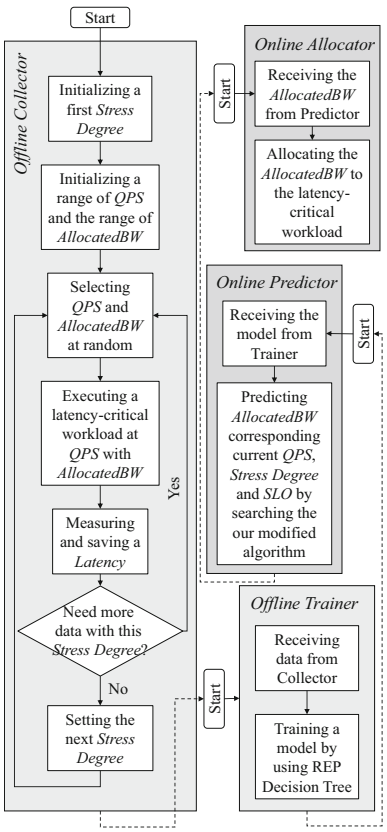


Fig. 3 Flow chart of the proposed OMBM-ML

allocated memory bandwidth for the latency-critical workload, QPS, and the memory controller contention.

And the *offline trainer* receives the data from the *collector* and trains the model by using REP decision tree and makes the tree-shaped model which predicts memory bandwidth for meeting SLO corresponding QPS and current memory controller contention.

Then the *online predictor* predicts allocated memory bandwidth by tracking the model from *Trainer* with our modified algorithm. And it sends the predicted memory bandwidth for the latency-critical workload to the *online allocator*.

Finally, the *online allocator* allocates the predicted memory bandwidth to the latency-critical workload and the rest of it to the batch-processing ones.

3.1 Data collection

The machine learning model is created by training the collected data. Table 1 shows data attributes for collecting the data. To collect the data, we provide the isolated core and cache to the latency-critical workload and the batch processing ones. And we execute STREAM benchmark [2] which is one of the batch processing workloads on each core except for the core on which the latency-critical workload is executed to give continuous stress to the memory controller. We also use Memguard to allocate isolated memory bandwidth to each workload. Because we collect data in this way, we do not need to make the new model even if the batch-processing workloads which are being executed changes. In other words, once the models are created, they can continue to be used with other batch-processing ones.

As described in Table 1, the first attribute, *Stress Degree*, means how stressful the batch processing workloads are to the memory controller. By using Intel Performance Counter Monitor (Intel PCM) [1], we measure how much memory bandwidth is currently used by the workloads. To be able to generate various values of *Stress Degree*, we let the STREAM benchmark to use different sizes of the data. We use the values of *Stress Degree* as 1/4, 2/4, 3/4, and 4/4 of the maximum memory bandwidth by default. And then we utilize a divide and conquer method to choose additional values of *Stress degree* for accurate prediction with minimal profiling overhead. Tail latencies increase on an exponential scale, because they are sensitive to memory bandwidth used. After the execution of profiling with the four default *StressDegree*, the additional values are further selected between two values in which the latency has increased rapidly. As expected, the more additional values, the higher the profiling overhead, but the more accurate the prediction. In the above method, we use 6.8 GB/s, 10.7 GB/s, 14.3 GB/s and 21.9 GB/s as the default values of *Stress Degree* and choose 17.9 GB/s as the additional value. Next, the second attribute is the *QPS* of the latency-critical workload. We set the minimum and maximum of the *QPS* values to those that the service provider can handle. Third, *AllocatedBW* means the amount of memory bandwidth for the latency-critical workload by utilizing Memguard. The *AllocatedBW* value is set within the range from 0.8 to 24.8 GB/s.

Finally, the last attribute described in the table, *Latency*, indicates the measured tail latencies under the described profiling setting.

3.2 The model

The collected data based on the attributes described above are used for model training. In this work, we use Weka [29], an open-source machine learning tool, for training the model. In selecting a machine-learning algorithm to create the prediction model, it is necessary to make some considerations. The first one is how fast the model can be trained, the second one is how accurate the results of prediction can be made based on the model, and finally, how fast the model can predict using the model. Considering these factors, we decide to use a REP Decision Tree among many machine learning algorithms. The model based on this algorithm has been used for making accurate predictions in another study [11]. Furthermore, the REP Decision Tree is known as an accurate yet fast decision tree learning algorithm [14, 18].

The model generated by REP Decision Tree is a tree-shaped one. The decision tree consists of decision nodes and leaf nodes. Each decision node contains one of data attributes, *Stress Degree*, *QPS*, *AllocatedBW*, and *Latency*,

Table 1 The attribute of data and description

Attribute	Description
Stress degree	How much the batch processing workloads give contention to the memory controller of the server
QPS	Queries per second of the latency-critical workload
Allocated BW	The allocated amount of the memory bandwidth to the latency-critical workload
Latency	Actual tail latency of the latency-critical workload

and its threshold values for choosing the next branch. The root node is the topmost one among the decision ones. And the leaf node contains the result of the classification or the real value of decision. Through this model, we obtain the real amount of memory bandwidth to be allocated to the latency-critical workload.

First, after input data comes into the model, each node compares the input data corresponding to the attribute type with the threshold value from the root node. For example, the root node has the attribute *SLO* and its threshold value is set to 10. The input data consists of the value of SLO, the current degree of stress, and the value of QPS. The root node receives the input and compares the *SLO* value of the input with the threshold value of the node, if the *SLO* value is smaller than that of the threshold, then the left branch will be made; otherwise, the right branch will be made. The input data is recursively compared to the threshold values of the decision nodes until a leaf node is reached.

By using the tree-shaped model as described above, we predict the range of tail latencies when the estimated amount of the memory bandwidth is allocated to the latency-critical workload by searching the tree. Figure 4 illustrates the abstract prediction model of a certain latency-critical workload. To give an example, we assume that the input data is as follows: the value of SLO is 1.5 ms, QPS is 3500, and the current stress degree is 10.9 GB/s. In the tree, we search the range like a red line as shown in Fig. 4a. The estimated range of the tail latencies will be updated until the leaf node is reached when the attribute of the node is *SLO(Latency)*. Since the root node has the attribute *SLO*, OMBM goes to the next left branch and the maximum estimated tail latency is updated to the threshold of the root node, $expected\ latency < 2.32$. In the next *SLO* node, the minimum estimated tail latency is updated because the threshold of the node is smaller than the input *SLO*. Finally, when we reach the leaf node, the range is determined as $0.48 < expected\ latency < 2.32$. If the estimated amount of memory bandwidth is allocated to the latency-critical workload, it shows the actual tail latencies between 0.48 and 2.32 ms. However, the range has some problems; the final range includes the *SLO* value of input data obviously. But, if we allocate the estimated amount of memory bandwidth to the latency-critical workload, there is a possibility that the expected latency is measured as one above the input *SLO*, 1.5 ms. Therefore, in order to meet the SLO perfectly, the original searching algorithm needs to be modified.

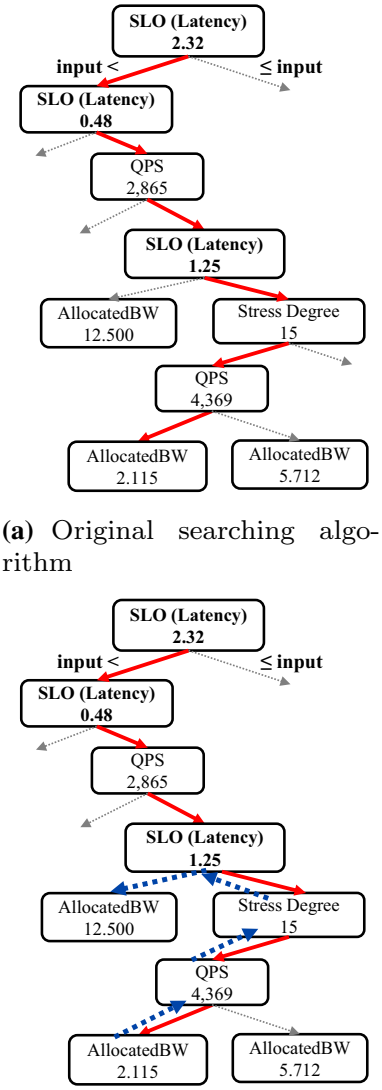


Fig. 4 Abstract model and tracking mark

Algorithm 1 Decision tree Modified Search Algorithm

Input: *Instance* : A set of *QPS* value, *SLO* value, and *Stress Degree*
Output: *PredictedBW* : Predicted amount of memory bandwidth to allocate
A : Minimum expected latency
B : Maximum expected latency should be under the SLO

```
1:  A ← 0
2:  B ← ∞
3:  def modifiedSearch(Instance):
4:    if Tree.Node ≠ LEAF
5:      if Instance.value(ATTR) < SplitPoint
6:        if ATTR = "SLO"
7:          B ← SplitPoint
8:        PredictedBW ←
9:          Tree.Node[Left].modifiedSearch(Instance)
10:       if ATTR = "SLO"
11:         if Change = true
12:           Change ← false
13:     else
14:       if ATTR = "SLO"
15:         oldA ← A
16:         A ← SplitPoint
17:       PredictedBW ←
18:         Tree.Node[Right].modifiedSearch(Instance)
19:       if ATTR = "SLO"
20:         if Change = true
21:           B ← A
22:           A ← oldA
23:           PredictedBW ←
24:             Tree.Node[Left].modifiedSearch(Instance)
25:           Change ← false
26:   if Node = LEAF
27:     if B > Instance.value("SLO")
28:       Change ← true
29:     else
30:       Change ← false
31:     return Tree.Node[output]
```

We show our modified searching algorithm based on the original one in Algorithm 1. Before the execution of the algorithm starts, the variables *A* and *B* are initialized to 0 and infinity, respectively (Lines 1 and 2). This is because *A* and *B* represent the minimum latency of the range and the maximum one to be predicted. Then, an instance is received as the input (Line 3). This original searching algorithm is executed until the leaf node is reached (Line 4). In tracking the decision nodes, the input instance corresponding to the attribute of the current node is compared with the threshold value of the current node. The threshold is represented by *SplitPoint* in Line 5 as a branch point. Normally, if the instance value is smaller than *SplitPoint*, then the left branch will be reached (Line 8); otherwise, the right branch will be reached (Line 16). After the node is selected, the same procedure will be performed to the next node, recursively (Lines 8 and 16). After OMBM-ML reaches the leaf node in this recursion (Line 23), the parent node will be revisited with the predicted value that is on the leaf node (Line 28).

Before Lines 8 and 16 are executed, we check whether the attribute of the current node is *SLO* or not. If it is *SLO*, then the values of the node should be kept. If the left branch is reached, the *SplitPoint* should be saved into *B* as the maximum (Lines 6–7); otherwise, the original value of *A* is saved into a temporary variable, and then A is updated with *SplitPoint* (Lines 13–15). This is the same as what we have previously described.

As mentioned above, the tail latency range from the original searching algorithm is problematic. In order to solve this problem, a variable, *Change*, is used to mark whether the tail latency range is larger than the value of SLO or not. The *Change* variable is set when the leaf node is reached for the first time. When the leaf node is reached, the largest value in the range is checked. In the algorithm, variable *B* has the largest value. If the value of *B* is greater than that of SLO, *Change* is set to true, and if it is smaller than that of SLO, *Change* is set to false. While the parent node is revisited, the algorithm first checks the attribute of the parent node (Lines 9 and 17). Lines 17 to 22 are more important than Lines 9 to 11. This is because we should change the branch of that parent node. Since if the value of *Change* is true (Line 18), this means the maximum value of the range is larger than that of *SLO*, we should make the maximum value one smaller than that of SLO. The first step to do it is to assign the value of *A* to *B* and load the previous value of *A* as the current one of *A*. In the next step, after assigning the new values to *A* and *B*, the left branch is made. The same steps are taken until the variable *Change* is set to false on the leaf node.

To illustrate this more clearly, we describe how the algorithm applies in an example. As described in Fig. 4b, until the leaf node is reached for the first time, the order of the searching sequence is the same as in Fig. 4a. However, since the maximum value of the range is larger than the value of SLO, the algorithm should find a new range to meet the SLO. The parent node is revisited as described in the thick dotted line. During this backtracking, the new range is updated if the right branch is made on the parent node. In other words, we get the predicted amount of the memory bandwidth, 12,500, when the expected tail latency is smaller the input SLO.

By applying the algorithm in the model, it is possible to compute and use the precise amount of the memory bandwidth to meet the SLO. The estimated amount of memory bandwidth is allocated by Memguard. The estimated one is allocated to the latency-critical workload, and the rest is allocated to the batch processing one. Each workload is executed on each isolated core with an isolated cache.

4 Experimental evaluation

In this section, we show accuracies of the prediction model of OMBM-ML compared to other machine learning models. And we show SLO guarantees and resource efficiency in experiments with real workloads.

4.1 Experiment environment and benchmark workload

Our experiments are performed on an Intel Haswell server. It has a quad-core Intel i7-4770k 3.5 GHz processor, 8 MB 16-way set associative cache memory and 25.6 GB/s memory bandwidth. The kernel version is Linux 3.14.15 kernel. For the latency-critical workload, we used Tailbench benchmark [16] developed by MIT. And for the batch-processing workload, we used HiBench benchmark [12] on Hadoop 2.6. Also, we find out the workloads from SPECcpu2006 that have big data processing characteristics [13]. For the following experiments, we chose five memory controller stress, 6.8 GB/s, 10.7 GB/s, 14.3 GB/s, 17.9 GB/s and 21.9 GB/s. And we set the range of QPS the latency-critical workloads from 1000 to 5000 and the range of the amount of memory bandwidth for them from 0.8 to 25.6 GB/s. In this setting, the QPS and memory bandwidth of the latency-critical workload were randomly given and the tail latencies were measured for the training model. We collected approximately 1000 to 2000 data for each workload and it took dozens of milliseconds to create the model by using the REP decision tree algorithm. Since each model had about 15 depths, the online predictor was able to predict the amount of memory bandwidth very quickly.

4.2 Prediction accuracy comparison

We first measure the prediction accuracy of the model generated by OMBM-ML. The accuracy of the experiments proves how well OMBM-ML guarantees the SLO of the latency-critical workload and how efficient the amount of memory bandwidth is predicted. To show the accuracy of OMBM-ML, we compare the accuracy of the OMBM-ML with other methods. Firstly, we use OMBM which predicts the amount of memory bandwidth under the worst memory controller contention of the prior work and some models generated by other machine learning algorithms, Linear Regression algorithm and Multi Layer Perceptron algorithm. We use silo, masstree, and img-dnn in Tailbench benchmark as the latency-critical workloads and STREAM benchmark as the batch processing workload.

Figure 5 is the results of how much the memory bandwidth predicted by each model. Each amount of the

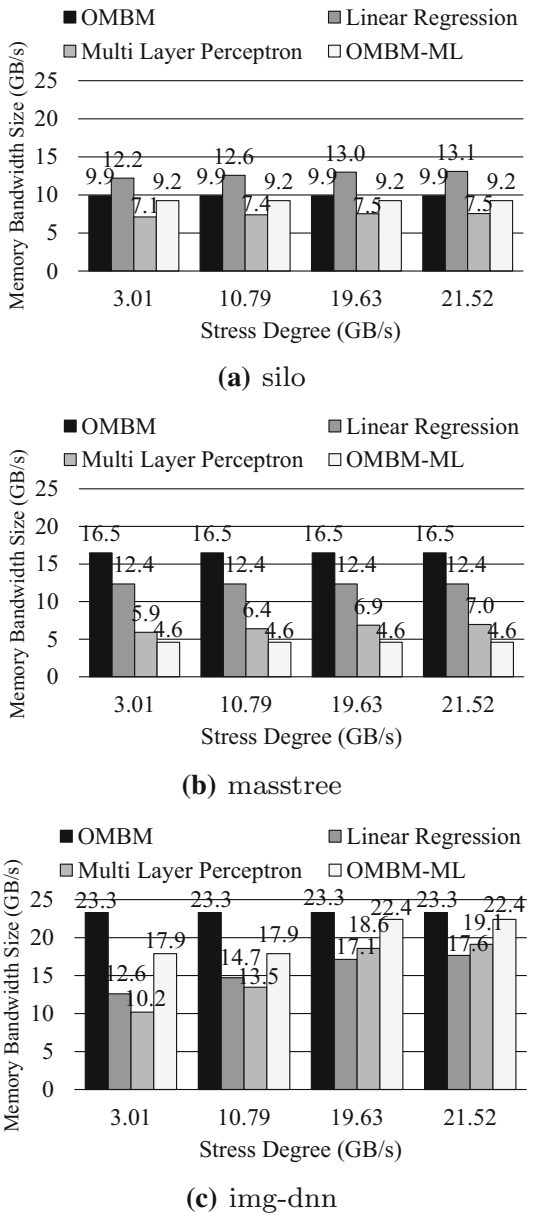
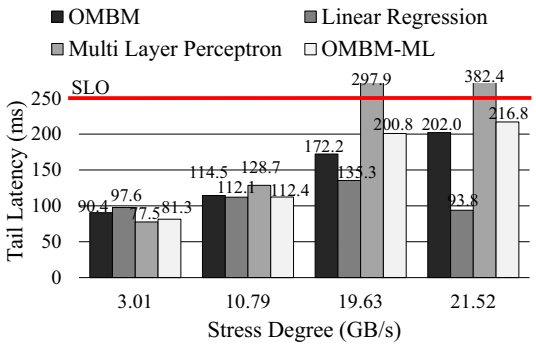
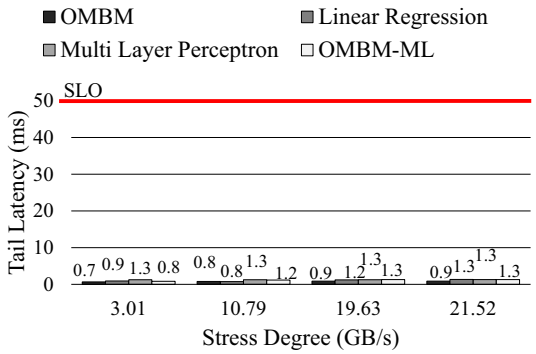


Fig. 5 Predicted amount of the memory bandwidth with each model

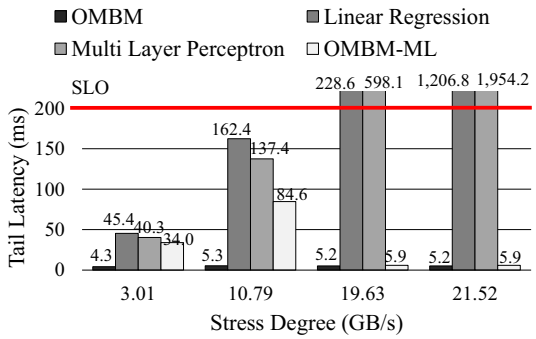
memory bandwidth is predicted to guarantee the SLO described in Fig. 6 in the given stress degree. Because the estimated memory bandwidth determines the server utilization, the model that predicts the smallest memory bandwidth increases the server utilization. As shown in the figure, OMBM-ML allocates less memory bandwidth to the latency-critical workloads than OMBM, since OMBM predicts the memory bandwidth under the assumption that the contention for the memory controller is the worst. In addition, OMBM-ML predicts the smallest memory bandwidth in the Fig. 5b. However, as shown in the Fig. 5a and c, linear regression model or multi-layer perception predict



(a) silo



(b) masstree



(c) img-dnn

Fig. 6 Accuracy comparison

lower memory bandwidth than ours, but as can be seen in the Fig. 6a and c, they do not guarantee SLO.

Figure 6 shows the tail latencies when the predicted amount of memory bandwidth from Fig. 5 is allocated to the latency-critical workload depending on the stress degree, 3.01 GB/s, 10.79 GB/s, 19.63 GB/s, and 21.52 GB/s.

Figure 6a is the result of silo. We set SLO of the silo to 250 ms and QPS to 8000. Each tail latency is measured when the predicted amount of the memory bandwidth by each model is allocated to the latency-critical workload. The tail latencies of the Multi Layer Perceptron are only exceeding the SLO when the stress degrees are 19.63 GB/s and 21.52 GB/s. It is because the Multi Layer Perceptron model cannot consider the current stress degree. Other models predict the amount of the memory bandwidth which is enough to guarantee the SLO. However, the Linear Regression model has the largest predicted amount of memory bandwidth and it is wasteful for guaranteeing the SLO.

Figure 6b shows the result of masstree. We set SLO of the masstree at 50 ms and QPS at 3000. Every predicted amount of the memory bandwidth by each model can meet the SLO in every stress degree. Since the QPS of the workload is so small, it seems not difficult to meet the SLO in any situation. However, the predicted amount of the memory bandwidth by each model are different as shown in Fig. 5b. We can conclude that the smallest one is the best efficient one.

In the case of img-dnn shown in Fig. 6c, the Linear Regression and the Multi Layer Perceptron do not meet the SLO in given stress degree, 19.63 GB/s and 21.52 GB/s. On the other hand, OMBM and OMBM-ML meet the SLO well. However, OMBM predicts the amount of the memory bandwidth in the worst contention, so the amount of the memory bandwidth is bigger than OMBM-ML one. In other words, OMBM-ML predicts the amount of the memory bandwidth according to the different situations.

4.3 QoS guarantee and server utilization improvement

In this section, we show that OMBM-ML can meet the SLO while the latency-critical workload is executing with the actual batch processing workloads. Also, we measure how the server utilization is improved than before. We used the same latency-critical workload in the accuracy comparison experiment, and some combinations of batch processing workload are shown in Table 2. And the maximum stress degrees of the memory controller is also measured.

Table 3 shows the predicted amount of the memory bandwidth by OMBM-ML when the latency-critical workload is executed with each combination of batch processing workloads. Also, it shows tail latency of the latency-critical workload. Table 4 shows how the system improves the utilization in each case.

Table 2 Combination of the batch processing workloads and maximum stress degree

Combi nation	Workloads	Max stress degree
A	Kmeans, Bayesian, Pagerank	3.6 GB/s
B	Kmeans, Libquantum, Bwaves	13.8 GB/s
C	Kmeans, Pagerank, Libquantum	8.7 GB/s

Table 3 Combination of the batch processing workloads and maximum stress degree

	Predicted memory bandwidth	Tail latency
(a) silo		
A	9.235 GB/s	8.14 ms
B	9.235 GB/s	169.45 ms
C	9.235 GB/s	142.838 ms
(b) masstree		
A	4.592 GB/s	0.638 ms
B	4.592 GB/s	0.697 ms
C	4.592 GB/s	0.679 ms
(c) img-dnn		
A	17.874 GB/s	96.924 ms
B	22.390 GB/s	112.218 ms
C	17.874 GB/s	77.082 ms

Table 3a is the result of silo. The QPS of silo is set at 8000 and the SLO at 250 ms. Next, Table 3b shows the result of masstree. Its QPS is set to 3000 and SLO to 50ms. Finally, img-dnn in Table 3c is set to QPS 550, and SLO 200ms. Every tail latencies are meet SLO. However, the silo and masstree show the same predicted amount of the memory bandwidth despite the stress degrees of the memory controller are different. Because the tail latencies tend to change dramatically according to the amount of the given memory bandwidth, we have to collect a lot of data in order to detect the sudden change of the model. However, it is hard to detect the change and it increases the overhead of collecting the data.

For each of these cases, we evaluate the server utilization improvement as shown in Table 4. The hardware resource utilization is improved by about 6.8× on average and up to 8.5×. Moreover, we find out the server utilization of OMBM-ML is improved more than OMBM one up to 1.42×.

To prove our system is effective in practice, we executed the latency-critical workloads with various QPSs. Since the Tailbench has static QPS during execution, we modified the benchmark to have dynamic QPSs. Most tail latencies are smaller or closer than the SLO. But, some tail latencies are not close to the SLO. Our system does not violate the SLO by allocating enough the amount of memory bandwidth to the latency-critical workload for strict QoS. If the model is not trained with sufficient data at some QPS, it shows significantly lower tail latencies than SLO and has also low server utilization. In addition, the tail latencies of the latency-critical workload are sensitive to the allocated memory bandwidth, so the tail latencies may increase sharply even if the allocated memory bandwidth is slightly reduced. In the case, the server utilization should be wasted and the tail latencies are lower than the SLO for strict QoS. In other words, if our model has sufficient data for accurate prediction under the corresponding conditions (current QPS, current memory controller contention), we can get the tail latencies close to the SLO. Otherwise, the tail latencies are far from the SLO since the tail latencies are sensitive corresponding to the allocated bandwidth, current QPS and memory controller contention or our model has not enough data for prediction (Fig. 7).

5 Related works

As the use of cloud services is growing by mobile or IoT users using wireless networks as well as desktop users using wired networks, the usage of data centers is getting bigger. As a web service recommendation design [23] that provides appropriate QoS to service providers has proposed, applications using data centers have regarded QoS guarantee as to the most important factor. But, because

Table 4 Hardware server utilization comparison

	Latency-critical workload alone	With three batch processing workloads
(a) silo		
A	18.30%	97.25%
B	18.52%	96.75%
C	16.82%	97.75%
(b) masstree		
A	12.77%	98.35%
B	11.85%	98.90%
C	12.95%	97.70%
(c) img-dnn		
A	13.62%	97.48%
B	11.38%	97.50%
C	15.62%	97.25%

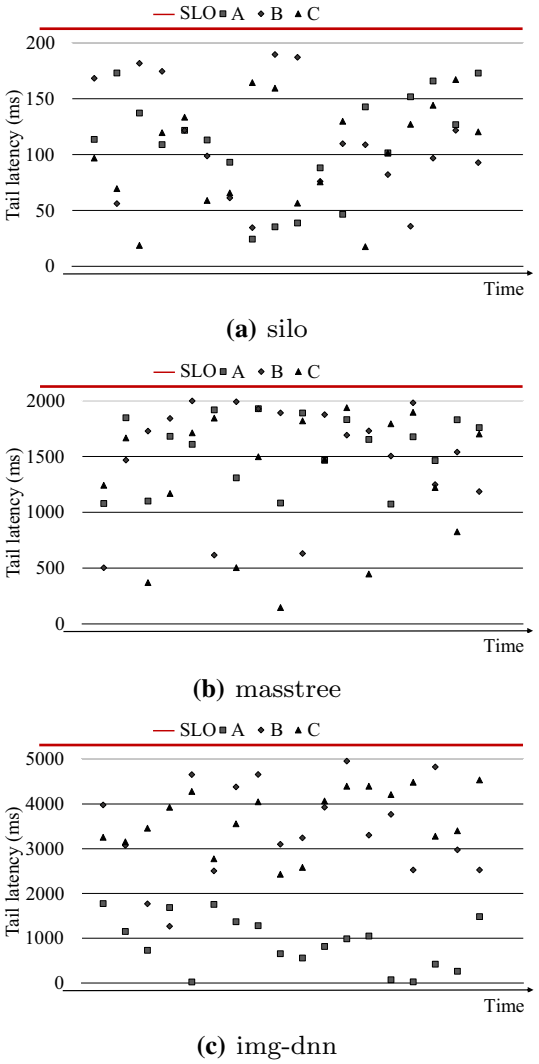


Fig. 7 QoS guarantee of the latency-critical workloads with dynamic QPSs

multiple requests or applications are simultaneously accessed to use the data center, contention in resources of the data center occurs. QoS guarantee is a huge challenge due to the contention.

Service providers for desktop users execute only latency-critical workloads on data centers for strict QoS. The servers in each data center do not be used efficiently due to guaranteeing the QoS. In order to address this, many researchers have been conducted to improve resource utilization while ensuring QoS. Most of the studies manage the shared resources to eliminate the contention situation [8, 15, 20, 30, 32, 36] or to avoid contention by managing between applications [22, 31].

Lo et al. [20] proposed an isolation scheme for the shared resources such as the core, LLC, power, and network. The system with this scheme can improve the utilization of the server. Also, for meeting the SLOs, the system not only applies the scheme but also monitors the QoS violations. If the system seems to have QoS violation, then the system limits the co-executed batch processing workload and allocates the remaining shared resources.

Yang et al. [31] predict the performance interference between the latency-critical workload and the batch processing workloads. Before actual co-executing with the batch processing workloads, the latency-critical workload measures performance interference due to memory subsystem usage. Based on this, the system decides which batch processing workload combination can be performed with the latency-critical workload. After collocation, however, if the system detects the QoS violation, then the system limits the execution of the batch processing workloads.

Because of the advantage of a machine learning model, some studies have been applied to predict the performance. Dwyer et al. [11] is the first study to solve complex problems of predicting performance by using machine learning. This study proposed a model to predict performance degradation of HPC workloads in a multicore environment. The data for training the model is collected by performance counter values while HPC workloads are concurrently executed. This model estimates performance degradation of the workload and by using it, the study proposes a scheduling method that can avoid performance degradation. Dauwe et al. [7] proposed co-location aware performance models in a multicore processor environment. The model is based on a neural network. Unlike methods for single latency-critical workload, Patel et al. [25] colocate multiple latency-critical workloads and multiple batch-processing ones effectively. To satisfy QoS of latency-critical workloads and achieve optimal performance of batch-processing workloads, they proposed a new model of shared resource partitioning configurations based on Bayesian Optimization method.

Zhang et al. [35] address QoS violation problems of spatial multitasking accelerators (Graphics Processing Unit). They predict duration and memory bandwidth usage under various computation resources of latency-critical workloads and allocate optimal resources to meet QoS. In addition, they monitor QoS of latency-critical ones in real-time and adjust the allocation of resources when unexpected QoS is detected due to contention on shared resources.

Since recent user-facing workloads have various behaviors, Chen et al. [6] proposed a shared resource allocation system for latency-critical workload which have diverse behavior unlike others assuming that queries have similar characteristics. They identify features of each query based on Lasso regression and assign shared resources for each query in accordance with identification at runtime.

Gill et al. [27] propose QoS-aware job cloud resource allocation, unlike traditional cloud services which schedule jobs by considering resource availability without SLA violation prevention.

Unlike the service providers for the desktop users, service providers for mobile or IoT users have been struggling to provide high QoS because of high wireless network latencies. In order to meet high QoS, many researchers have studied network communication optimization.

Chung et al. [17] emphasize the importance of medical care systems to be able to send and receive medical data anytime, anywhere between patients and hospitals. To do this, they optimize network communication for improving QoS. Cui et al. [33] propose a channel allocation algorithm which improves chaotic neural network technology for providing high QoS to multimedia mobile services. Felici-Castell et al. [26] also address the dramatic needs of multimedia mobile traffic, which increases by 50% every year. To ensure QoS for video streaming applications, they create a QoE-based mobile media cloud network topology.

Chunlin et al. [19] addresses limitations of the existing cloud service architectures designed for desktop users with fast wired network connectivity. They figure out characteristics of mobile applications, such as video/audio processing, mobile commerce, and mobile healthcare schedule application jobs to make use of appropriate resources based on the characteristics. M. Badawy et al. [21] also propose an efficient job scheduling algorithm depending on the different QoS of IoT applications.

6 Conclusion and future work

We tried to collocate the latency-critical workload and some batch processing workloads so that server utilization can be improved without any QoS violation of the latency-critical workload. In order to achieve this goal, we present OMBM-ML, the memory bandwidth management is based on the machine learning model. The model estimates the amount of the memory bandwidth to be able to ensure the QoS of the latency-critical workload. By this management, OMBM-ML enables the cloud service provider can guarantee QoS of the latency-critical workload accurately and improve the server utilization up to 8.5×.

As you can see in the Fig. 4b, in order to meet SLO using the modified searching algorithm, OMBM-ML allocates sufficient memory bandwidth to the latency-critical workload even though server utilization is low. Thus, if the model has enough data for accurate prediction, server utilization is wasted. In Table 3a, the three combinations allocate the same amount of memory bandwidth to the latency-critical workload, although they make different memory contention. Especially, silo with C shows 8.14 ms tail latency even though SLO is 250 ms. In other words, the smaller memory bandwidth allocation would satisfy the SLO. The machine learning model helps more accurate predictions as the data is larger. Therefore, if there is a large difference between the SLO and the tail latency, it is necessary to train the model by extracting new data by adjusting the memory bandwidth at runtime. To handle this limitation, we will improve our system which automatically finds new data and trains the model.

Acknowledgements This research was supported by (1) Institute for Information & communications Technology Promotion (IITP) Grant funded by the Korea government (MSIP) (R0190-16-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development). It was also partly supported by (2) National Research Foundation of Korea (NRF) Grant funded by the Korea government (MSIP) (NRF-2017R1A2B4004513, Optimizing GPGPU virtualization in multi GPGPU environments through kernels' concurrent execution-aware scheduling), and partly supported by (3) the National Research Foundation (NRF) Grant (NRF-2016M3C4A7952587, PF Class Heterogeneous High Performance Computer Development). In addition, this work was partly supported by (4) BK21 FOUR Intelligence Computing (Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea (NRF) (Grant 4199990214639).

References

1. Intel Performance Counter Monitor. https://software.intel.com/en-us/articles/intel-performance-counter-monitor
2. STREAM Benchmark. http://www.cs.virginia.edu/stream/ref.html
3. Amy Ousterhout, J.B., Joshua Fried, A.B., Hari Balakrishnan, M.C.: Shenango: achieving high CPU efficiency for latency-sensitive datacenter workloads. In: Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (2019)
4. Azimi, R., Kwon, Y., Elnikety, S., Syamala, M., Narasayya, V., Herodotou, H., Microsoft, P.T., Alex, B., Microsoft, C., Jack, B., Microsoft, Z., Wang, B.J., Bing, M.: PerfIso: Performance Isolation for Commercial Latency-Sensitive Services C: alin Iorgulescu* EPFL. Technical report (2018)
5. Barroso, L.A., Clidaras, J., Hoelzle, U.: The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool Publishers, San Rafael (2013)
6. Chen, Q., Wang, Z., Leng, J., Li, C., Zheng, W., Guo Avalon, M.: Towards QoS awareness and improved utilization through multi-resource management in datacenters. In: Proceedings of the International Conference on Supercomputing, pp. 272–283, New York, NY, USA, Jun 2019. Association for Computing Machinery
7. Dauwe, D., Jonardi, E., Friese, R., Pasricha, S., Maciejewski, A.A., Bader, D.A., Siegel, H.J.: A methodology for co-location aware application performance modeling in multicore computing. In: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, pp. 434–443, May 2015
8. Delimitrou, C., Kozyrakis, C.: Quasar: Resource-efficient and qos-aware cluster management. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, pp. 127–144, New York, NY, USA (2014). ACM

9. Desai, N., Cirne, W.: Job Scheduling Strategies for Parallel Processing, pp. 274–278. Springer, Cham (2017)
10. Di, S., Kondo, D., Cirne, W.: Characterization and comparison of cloud versus grid workloads. In: 2012 IEEE International Conference on Cluster Computing, pp. 230–238, Sept 2012
11. Dwyer, T. , Fedorova, A., Blagodurov, S., Roth, M., Gaud, F., Pei, J.: A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pp. 83:1–83:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press
12. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The hibench benchmark suite: characterization of the mapreduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), pp. 41–51, March 2010
13. Hurt, K., John, E.: Analysis of memory sensitive spec cpu2006 integer benchmarks for big data benchmarking. In: Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, PABS '15, pp. 11–16, New York, NY, USA (2015). ACM
14. Kalmegh, S.: Analysis of weka data mining algorithm reptree, simple cart and randomtree for classification of indian news. Int. J. Innov. Sci. Eng. Technol **2**(2), 438–446 (2015)
15. Kasture, H., Sanchez, D.: Ubik: efficient cache sharing with strict qos for latency-critical workloads. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, pp. 729–742, New York, NY, USA (2014). ACM
16. Kasture, H., Sanchez, D.: Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In: 2016 IEEE International Symposium on Workload Characterization (IISWC), pp. 1–10, Sept 2016
17. Kyungyoung, C., Park, R.C.: Cloud based u-healthcare network with QoS guarantee for mobile health service. In: Cluster Computing (2017)
18. Lakshmi Devasena, C.: Comparative analysis of random forest, rep tree and j48 classifiers for credit risk prediction. In: International Journal of Computer Applications (0975-8887), International Conference on Communication, Computing and Information Technology (ICCCMIT-2014) (2014)
19. Li Chunlin, T.J., Youlong, L.: Distributed QoS-aware scheduling optimization for resource-intensive mobile application in hybrid cloud. In: Cluster Computing (2017)
20. Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., Kozyrakis, C.: Heracles: Improving resource efficiency at scale. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA '15, pp. 450–462, New York, NY, USA (2015). ACM
21. Mahmoud, Z.H.A., Badawy, M., Ali, H.A.: QoS provisioning framework for service-oriented internet of things (IoT). In: Cluster Computing (2019)
22. Mars, J., Tang, L., Hundt, R., Skadron, K., Soffa, M.L.: Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-44, pp. 248–259, New York, NY, USA (2011). ACM
23. Anithadevi, N., Sundarambal, M.: A design of intelligent QoS aware web service recommendation system. In: Cluster Computing (2018)
24. Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-clouds: managing performance interference effects for qos-aware clouds. In: Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, pp. 237–250, New York, NY, USA (2010). ACM
25. Patel, T., Tiwari, D.: CLITE: efficient and QoS-aware co-location of multiple latency-critical jobs for warehouse scale computers. In Proceedings—2020 IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, pp. 193–206. Institute of Electrical and Electronics Engineers Inc., Feb 2020
26. Santiago Felici-Castell, J.S.G., Garcia-Pineda, M.: Adaptive QoE-based architecture on cloud mobile media for live streaming. In: Cluster Computing (2018)
27. Sukhpal Singh Gill, M.S., Charana, I., Buyya, R.: CHOPPER: an intelligent QoS-aware autonomic resource management approach for cloud computing. In: Cluster Computing (2017)
28. Sung, H., Min, J., Ha, S., Eom, H.: OMBM: optimized memory bandwidth management for ensuring QoS and high server utilization. In: 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 269–276. IEEE, Sep 2017
29. Witten, I., Frank, E., Hall, M. A., Pal, C. J.: Data Mining: Practical Machine Learning Tools and Techniques (2016)
30. Xu, C., Felter, W., Rajamani, K., Rubio, J., Ferreira, A., Li, Y.: dCat: dynamic cache management for efficient, performance-sensitive infrastructure-as-a-service. In: Proceedings of the 13th EuroSys Conference, EuroSys 2018, volume 2018-January, pp. 1–13, New York, NY, USA, Apr 2018. Association for Computing Machinery, Inc.
31. Yang, H., Breslow, A., Mars, J., Tang, L.: Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In: Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA'13, pp. 607–618, New York, NY, USA (2013). ACM
32. Yang, X., Blackburn, S. M., McKinley, K. S.: Elfen scheduling: Fine-grain principled borrowing from latency-critical workloads using simultaneous multithreading. In: 2016 USENIX Annual Technical Conference (USENIX ATC 16), pp. 309–322, Denver, CO, 2016. USENIX Association
33. Yongfeng Cui, Y. M., Zhongyuan Zhao, Dong, S.: Resource allocation algorithm design of high quality of service based on chaotic neural network in wireless communication technology. In: Cluster Computing (2017)
34. Yun, H., Yao, G., Pellizzoni, R., Caccamo, M., Sha, L.: Memguard: memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 55–64, April, 2013
35. Zhang, W., Cui, W., Fu, K., Chen, Q., Mawhirter, D. E., Wu, B., Li, C., Guo, M.: Laius: towards latency awareness and improved utilization of spatial multitasking accelerators in datacenters. In: Proceedings of the International Conference on Supercomputing, pages 58–68. Association for Computing Machinery, Jun, 2019
36. Zhu, H., Erez, M.: Dirigent: enforcing qos for latency-critical tasks on shared multicore systems. In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, pp. 33–47, New York, NY, USA (2016). ACM

Hanul Sung received BS degree in computer science from Sangmyung University (SMU), Seoul, Korea, in 2012, and PhD degree in computer science and engineering at Seoul National University (SNU), Seoul, Korea, in 2020. She is currently an engineer in SK Hynix. Her main research interests are distributed systems, operating systems, high performance storage systems and cloud computing.

Jeesoo Min is a Master student in computer science and engineering at Seoul National University (SNU), Seoul, Korea. She received BS degree in computer science from Handong University, Pohang, Korea, in 2016. Her main research interests are operating systems, cloud computing and storage.

Donghun Koo is a Master student in computer science and engineering at Seoul National University (SNU), Seoul, Korea. He received BS degree in electronics and information engineering from Electronics and Information Engineering of Korea Aerospace University in Korea, Seoul, Korea. His main research interests are operating systems, distributed file systems and high performance storage systems.

Hyeonsang Eom received the BS degree in computer science and statistics from Seoul National University (SNU), Seoul, Korea, in 1992, and the MS and PhD degrees in computer science from the University of Maryland at College Park, Maryland, USA, in 1996 and 2003, respectively. He is currently an associate professor in the Department of Computer Science and Engineering at SNU, where he has been a faculty member since 2005. He was an intern in the data engineering group at Sun Microsystems, California, USA, in 1997, and a senior engineer in the Telecommunication R&D Center at Samsung Electronics, Korea, from 2003 to 2004. His research interests include distributed systems, cloud computing, operating systems, high performance storage systems, energy efficient systems, fault-tolerant systems, security, and information dynamics.