Sensors and Materials, Vol. 29, No. 11 (2017) 1497–1512
MYU Tokyo

S & M 1444

Context-Aware Task Assignment for MapReduce in Heterogeneous Clouds

Wei-Tsung Su, Wei-Fan Pan, and Chao-Chun Chen[1*]

Department of Computer Science and Information Engineering, Aletheia University,
No. 32, Zhenli St., Danshui Dist., New Taipei City 25103, Taiwan, R.O.C.
[1]Institute of Manufacturing Information and Systems, National Cheng Kung University,
No. 1, University Rd., Tainan 701, Taiwan, R.O.C.

(Received April 6, 2017; accepted August 4, 2017)

Keywords: heterogeneous cloud, MapReduce, task assignment, resource-proportional approach

The MapReduce programming model is designed to process large data sets based on parallel computing among multiple computer nodes (CNs). Because the data size is considerably increased (data are collected from sensors in most cases), the optimization problem of task assignment becomes important to improve the performance of MapReduce. Unfortunately, this problem is even more difficult in heterogeneous clouds in which the CNs have different capabilities and available resources. In this paper, the context-aware task assignment (CATA) approach is proposed to improve the performance of MapReduce in a twofold manner. First, CATA takes the resource demands for different types of jobs into account. Second, CATA can assign tasks to CNs according to their capabilities and available resources in a resource-proportional manner. The experimental results show that CATA can efficiently reduce the job execution time by 10 to 40%.

1. Introduction

Cloud computing is a computing paradigm intended to provide resources as services on-demand.[1] For most applications of Internet of Things (IoT), such a paradigm can provide processing of massive data streams from a large number of sensor nodes in cyber-physical environments. A cloud data center consists of a large number of commodity computer nodes (CNs). With parallel processing on these CNs, MapReduce is considered as a suitable model for large data processing. In MapReduce, input data collected from sensor nodes are first divided into a large number of data splits. Then, these data splits are processed by tasks, called mappers and reducers, assigned to CNs in a parallel manner. MapReduce has a built-in fault-tolerance mechanism[2] to ease the development of parallel applications. Thus, a programmer can focus on applications without considering the synchronization among CNs. As a result, many cloud service providers, such as Google, Yahoo, and Facebook, have utilized MapReduce for data analysis.

Apache Hadoop and Spark are the most famous open source frameworks to realize MapReduce. In these frameworks, the default task assignment (DTA) is suggested to refer the number of CPU cores on CNs. In practice, the number of tasks assigned to a CN can be 1 to 2 times the number of CPU cores on this CN. Unfortunately, this simple task assignment becomes inefficient for

*Corresponding author: e-mail: chaochun@mail.ncku.edu.tw
http://dx.doi.org/10.18494/SAM.2017.1659

ISSN 0914-4935

the following reasons.[3] First, many IoT applications (e.g., air quality indexing, ocean current monitoring, and industrial monitoring and control systems) need to deploy numerous sensor nodes for collecting environmental sensor data. Consequently, the size of the data collected from sensor nodes can be enormously increasing over time. Second, the heterogeneity of CNs in the cloud data center increases the difficulty of load balancing. Therefore, a smart task assignment is required to raise the resource utilization of the data center.[4] Although several approaches were proposed,[5,6] the following open issues remain to be addressed for the cloud-side infrastructure to cope with IoT applications consisting of various scales of sensors.

1. Job diversity. Different types of jobs have diverse resource demands. For example, a processing-intensive job can speed up with more computing resources; in contrast, a communication-intensive job requires high-performance I/O resources. Thus, the cloud resources can be optimally utilized only by considering the job diversity.[7]
2. Cloud heterogeneity. The cloud resources are integrated from a large number of CNs with different capabilities and resources. Load unbalancing can be caused by assigning tasks without considering the heterogeneity of CNs.[8] Consequently, it degrades the performance of MapReduce.
3. Network dynamics. Because the CNs are network-connected, network dynamics can significantly affect the performance of MapReduce. For example, the transmission time can increase because of the low network throughput between two CNs. Unfortunately, most task assignment approaches do not take network dynamics into account.[9]

In this paper, the context-aware task assignment (CATA) approach is proposed to address the above issues. The contributions of this paper are described as follows.

1. We prove that the job processing time is shortest if the loads among CNs are perfectly balanced.
2. CATA classifies jobs into two types: (1) processing-intensive or (2) communication-intensive jobs. Then, CATA can allocate the appropriate resources according to the different types of jobs.
3. The objective of CATA is to balance the loads among CNs while taking the heterogeneity of CNs and network dynamics into account.

The rest of this paper is organized as follows. In Sect. 2, the previous work relevant to task assignment in MapReduce is introduced. In Sect. 3, we indicate the challenges and opportunities of task assignment in heterogeneous clouds. In Sect. 4, CATA is introduced in detail. The experiment results for evaluating the performance of CATA are presented in Sect. 5. Finally, the conclusions and future work are summarized in Sect. 6.

2. Related Work

2.1 Execution flow of MapReduce

As shown in Fig. 1, MapReduce consists of three phases called map, shuffle, and reduce.[10] In the map phase, the input data is first divided into several data splits. Then, tasks, called mappers, are assigned to CNs for processing these data splits in a parallel manner. In the reduce phase, tasks, called reducers, are assigned to CNs for integrating the intermediate results generated by mappers. Because mappers and reducers may be assigned to different sets of CNs, the intermediate results must be transmitted among CNs in the shuffle phase.

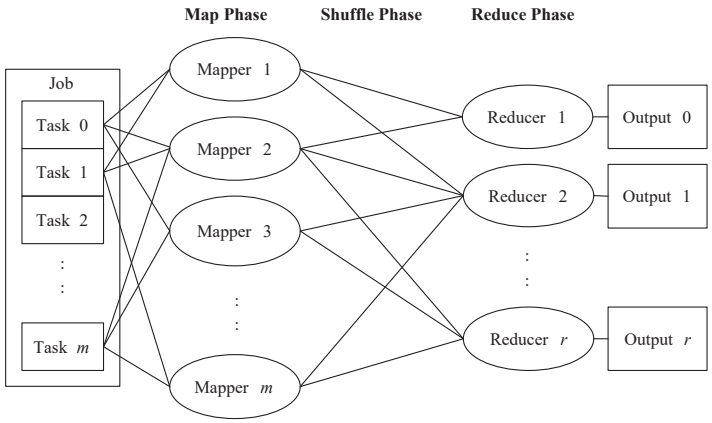

Fig. 1. Execution flow of MapReduce.

In the above execution flow, the job execution time consists of (1) map processing time, (2) shuffle communication time, and (3) reduce processing time. Job execution time can depend on several parameters, such as the number of CNs executing mappers in the map phase, the number of CNs executing reducers in the reduce phase, and the communication load in the shuffle phase. Thus, the performance of MapReduce can be improved by carefully selecting these parameters.[11]

2.2 Performance improvement of MapReduce

Several approaches were proposed to improve the performance of MapReduce. Kambatla *et al.* first proved that the best practices of task assignment are different for different types of jobs.[12] According to the experimental results, the Hadoop parameters, such as the number of CNs, the number of mappers, and the number of reducers, can affect the performance of MapReduce. Moreover, a signature-based approach is proposed to determine the types of jobs based on the signature of this job and then indicates a predefined task assignment strategy. Unfortunately, this approach requires time to create the signature database and cannot adapt to fine-grained job diversity using a predefined signature database.[13,14]

Although Tian *et al.* proposed a fine-grained job classifier,[15] there is still a problem of selecting the appropriate number of CNs, mappers, and reducers in the task assignment strategies for different job types. Under the multicore with a tiling platform, Chen *et al.* proposed the tiled-MapReduce that speeds up at least 19% compared with the original MapReduce.[16] However, a specific hardware platform is required. Moreover, Ahmad *et al.* observed that communication-intensive jobs, which tend to output as much as the input, incur a considerable performance overhead (e.g., 30–40%). The reason for this is that a high data volume is required to be transferred using an affordable disk and network bandwidths in the shuffle phase. Thus, overlapping the shuffle delay with mapper and reducer computations is performed to reduce the job execution time.[17]

In summary, the above approaches tend to reduce the job execution time by finding the best practices for task assignment according to the type of job. Unfortunately, most of them assumed that the CNs are homogeneous and did not take the cloud heterogeneity and network dynamics into account. However, heterogeneous clouds become increasingly common. Thus, a new task assignment approach, which is able to adapt to the capabilities and available resources of CNs, is required.

3. Problem Statement

Prior to the problem statement, Table 1 shows the notations and their descriptions used in this paper. As the task assignment in heterogeneous clouds shown in Fig. 2(a), assume that the processor speed of CN A (denoted by P_A) and that of CN B (denoted by P_B) are 10 and 5 million instructions per second (MIPS), respectively. In addition, assume that a job requires one instruction to process one bit (IPb) of input data. If this job needs to process 300 Mb of data that are equally assigned to A and B, then both A and B need to process 150 Mb of data. Thus, A and B need to spend 15 and 30 s to finish the tasks, respectively. Although A can finish its tasks in 15 s, the map phase can be accomplished only after both A and B finish their tasks. Thus, the time required to accomplish the map phase will be 30 s. However, as the other task assignment shown in Fig. 2(b), if 200 and 100 Mb of the same job are assigned to A and B, respectively, according to their processor speeds, then A and B will both spend 20 s to finish the tasks. Thus, the map phase can be accomplished in 20 s.

As we can see from these two cases, the job execution time can be prolonged if the loads among CNs are unbalanced. Thus, optimizing load balancing among CNs can significantly improve the performance of MapReduce. Unfortunately, it is difficult to optimize load balancing in heterogeneous clouds. The challenge is that CNs have wide varieties of computation and communication capabilities and available resources, such as CPU time, memory, and storage in heterogeneous clouds.

Table 1
Notation list.

Notation	Description
L	Size of data to be processed by current job (bits).
C_T	Complexity of task (e.g., mapper or reducer) of current job (instructions per bit).
N	Set of CNs selected to execute tasks for current job where $\|N\|$ indicates the size of N.
l_i	Size of data to be processed by the tasks assigned to CN i for current job (bits).
P_i	Processor speed of CN i (MIPS).
PN_N	Task processing time by all CNs in N.
c_i	Number of CPU cores of CN i.

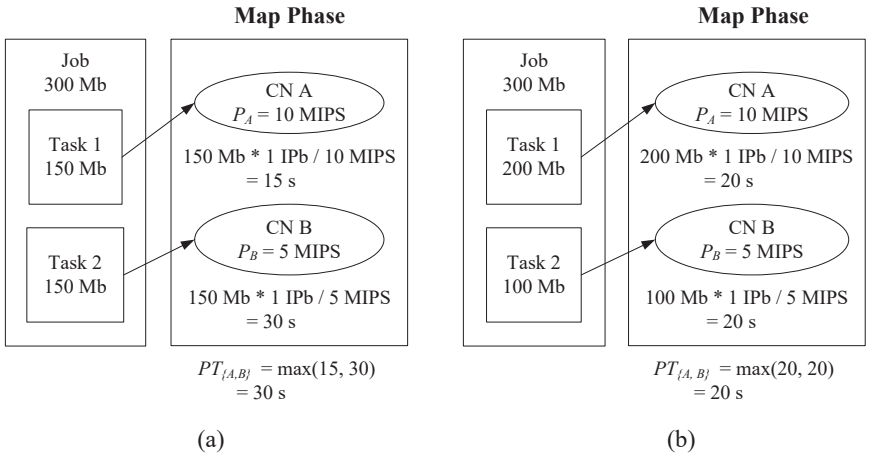

Fig. 2. (Color online) Examples of loads in heterogeneous clouds: (a) load unbalanced and (b) load balanced.

In this paper, CATA is proposed to solve the above problem by balancing the loads of CNs in heterogeneous clouds. The main concept of CATA is to adaptively assign mappers and reducers to CNs according to the type of job and the capabilities and available resources of CNs. In the following paragraphs, we will describe the challenges and opportunities to improve the performance of MapReduce in heterogeneous clouds.

3.1 Computation speedup in heterogeneous clouds

In MapReduce, a job is typically executed in a parallel manner for computation speedup. Intuitively, the computation speedup is higher if more CNs are involved. Unfortunately, it is not exactly true in heterogeneous clouds. For example, in Fig. 3(a), the map phase can be finished in 15 s with two CNs. However, although the third CN C is added to execute this job as shown in Fig. 3(b), the processor speed of C is too low to prolong the map processing time. Therefore, if the loads are not balanced among CNs, the computation speedup cannot be guaranteed even though there are more CNs selected to execute a job. In this paper, the criterion of computation speedup in heterogeneous clouds is defined and described as follows.

Definition 1 (Task Assignment). Assume that there is a job with a data size of L bits, which will be executed by a selected set of CNs denoted by N. A task assignment for this job can be defined as $\{l_1, l_2, ..., l_i, ..., l_{|N|}\}$, where l_i is the size of the data needed to be processed by the tasks assigned to CN $i \in N$ and $\sum_{i=1}^{|N|} l_i = L$.(18)

Different task assignments will have different task processing times defined as follows.

Definition 2 (Task Processing Time). Since the map (or reduce) phase can be accomplished only after all the CNs have finished their tasks,(19) the task processing time is defined as

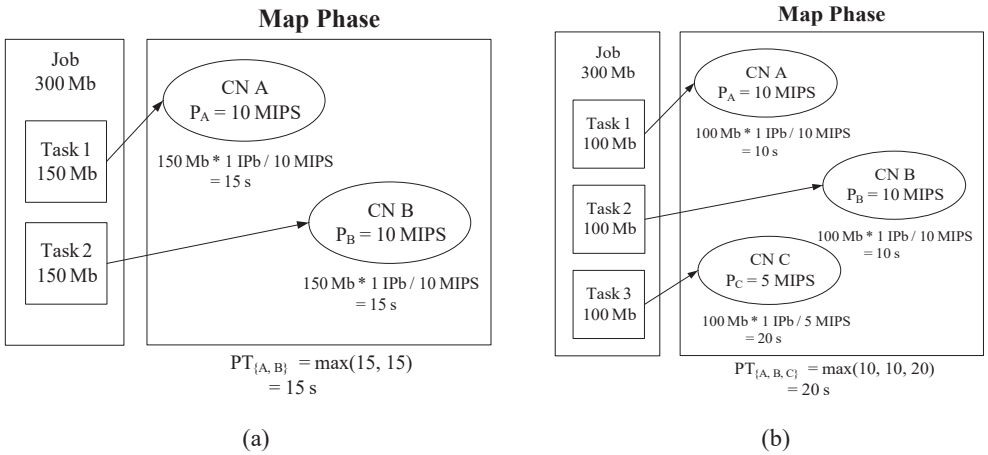

Fig. 3. Challenge of computation speedup in heterogeneous clouds: (a) load balanced among two CNs and (b) load unbalanced among three CNs.

$$PT_N = \max_{\forall i \in N} \left(\frac{l_i \times C_T}{P_i} \right), \tag{1}$$

where P_i is the processor speed of CN $i \in N$ and C_T is the complexity of the task (e.g., mapper or reducer) of the current job.

The objective of a task assignment is to reduce the task processing time while considering the load balance among CNs. In this paper, the perfect load balance (PLB) is defined as follows.

Definition 3 (PLB). A task assignment among all CNs in N is PLB if and only if

$$\frac{l_1}{P_1} = \frac{l_2}{P_2} = ... = \frac{l_i}{P_i} = ... = \frac{l_{|N|}}{P_{|N|}}. \tag{2}$$

In the following paragraphs, we will prove that the task processing time is shortest if the task assignment is PLB.

Theorem 1. If the task assignment among all CNs in N is PLB, then the task processing time is shortest.

Proof. By Definitions 2 and 3, if the task assignment among all CNs in N is PLB, then the task processing time is defined as

$$PT_N = \max_{\forall i \in N} \left(\frac{l_i \times C_T}{P_i} \right) = \frac{\sum_{i=1}^{|N|} \left(\frac{l_i \times C_T}{P_i} \right)}{|N|}. \tag{3}$$

If CN $i \in N$ yields x bits to CN $j \in N$ where $x > 0$ and $i \neq j$, then this task assignment will not be PLB by Definition 3. The new sizes of data that need to be processed by the tasks assigned to CN i and CN j are $l'_i = l_i - x$ and $l'_j = l_j - x$, respectively. Thus, the new task processing time is

$$PT'_N = \max \left(\frac{l_1 \times C_T}{P_1}, \frac{l_2 \times C_T}{P_2}, ..., \frac{(l_i - x) \times C_T}{P_i}, ..., \frac{(l_j + x) \times C_T}{P_j}, ..., \frac{l_{|N|} \times C_T}{P_{|N|}} \right). \tag{4}$$

Since $x > 0$, the following inequality holds:

$$\frac{(l_j + x) \times C_T}{P_j} > \frac{l_i \times C_T}{P_i}. \tag{5}$$

According to Eqs. (3)–(5), it is easily proven that

$$PT'_N = \frac{(l_j + x) \times C_T}{P_j} > PT_N. \square \tag{6}$$

Based on Theorem 1, we can prove that the task processing time is shorter by adding more CNs if the task assignment is still PLB.

Theorem 2. Assume that the task assignments among two sets of CNs denoted by N and N', where $|N| = n$, $|N'| = m$, and $m > n$, for a job are both PLB, then $PT_N > PT_{N'}$.

Proof. According to Eq. (3), if the task assignment is PLB, then the relationship between l_i and l_j can be obtained as $l_j = l_i \frac{P_j}{P_i}$. Thus, $\sum_{j=1}^{|N|} l_j = \sum_{j=1}^{|N|} \left(l_i \frac{P_j}{P_i} \right)$. Since $\sum_{j=1}^{|N|} l_j = L$, the following equation holds:

$$L = \sum_{j=1}^{|N|} \left(l_i \frac{P_j}{P_i} \right) = \frac{l_i}{P_i} \sum_{j=1}^{|N|} P_j. \tag{7}$$

According to Eq. (7), l_i is then given by

$$l_i = \frac{L \times P_i}{\sum_{j=1}^{|N|} P_j}. \tag{8}$$

Without loss of generality, we may assume that $P_i \leq P_j$ when $i < j$. Assume that a new CN k is added to N, where $P_i \leq P_k \leq P_{i+1}$ and the new set of CNs is denoted by N'. To preserve PLB, the new task assignment $\left\{ l'_1, l'_2, ..., l'_i, ..., l'_{|N'|} \right\}$ is required, where l'_i is the new size of data to be processed by the tasks assigned to CN $i \in N'$ and $\sum_{i=1}^{|N'|} l'_i = L$. According to Eq. (8), the following equations hold:

$$l'_i = \frac{L \times P_i}{\sum_{j=1}^{|N'|} P_j}, \tag{9}$$

$$\frac{l_i}{l'_i} = \frac{\frac{L \times P_i}{\sum_{j=1}^{|N|} P_j}}{\frac{L \times P_i}{\sum_{j=1}^{|N'|} P_j}} = \frac{\sum_{j=1}^{|N'|} P_j}{\sum_{j=1}^{|N|} P_j}. \tag{10}$$

Since $\sum_{j=1}^{|N|} P_j < \sum_{j=1}^{|N'|} P_j$, it is obvious that

$$l_i > l'_i. \tag{11}$$

According to Eq. (3),

$$PT_N = \frac{l_i \times C_T}{P_i} > \frac{l'_i \times C_T}{P_i} = PT_{N'}. \square \tag{12}$$

According to Theorem 2, the computation speedup of parallel computing on more CNs can be guaranteed if the task assignment is PLB.

3.2 Communication load in heterogeneous clouds

In addition to the task processing time, the shuffle communication time also influences the performance of MapReduce. Since the communication among CNs is based on networking, the

insufficient bandwidths among these CNs can incur a substantial performance reduction. In a heterogeneous cloud, the bandwidths among these CNs may vary greatly, so the task assignment becomes more complex. For example, as shown in Fig. 4, assume that this is a communication-intensive job where the size of the output data is equal to that of the input data. In addition, assume that the bandwidth between A and C is 1 Mbps, and the bandwidth between B and C is 10 Mbps. Although the load is balanced among A and B in Fig. 4(b), the map processing time (i.e., 20 s) is shorter than that in Fig. 4(a) (i.e., 30 s). However, owing to the communication bottleneck between A and C, the shuffle communication time in Fig. 4(b) (i.e., 210 s) is much longer than that in Fig. 4(a) (i.e., 165 s). By summing the times required to complete the map and shuffle phases, the task assignment in Fig. 4(a) (i.e., 195 s) is better than that in Fig. 4(b) (i.e., 230 s).

In conclusion, the optimization problem of task assignment must consider not only the computation speedup but also the communication load. Unfortunately, it is impractical to find the optimal number of CNs, mappers, and reducers while considering the type of job and capabilities and resources of CNs. Thus, CATA is proposed to find the approximation solution in a reasonable time by exploring the above challenges and opportunities of task assignment in heterogeneous clouds.

4. CATA

CATA has two adaptation levels: (1) node-level adaptation (NLA) and (2) task-level adaptation (TLA). In the first level (CATA-NLA), the appropriate number of CNs is determined according to the type of current job. Then, the determined number of CNs is selected to execute the current job using a resource-proportional approach. In CATA-NLA, the number of tasks assigned to a CN only depends on the number of CPU cores. In the second level (CATA-TLA), these tasks are reassigned to the selected CNs according to the processor speeds of CNs for approximating PLB. The flow chart of CATA is shown in Fig. 5 and described as follows.

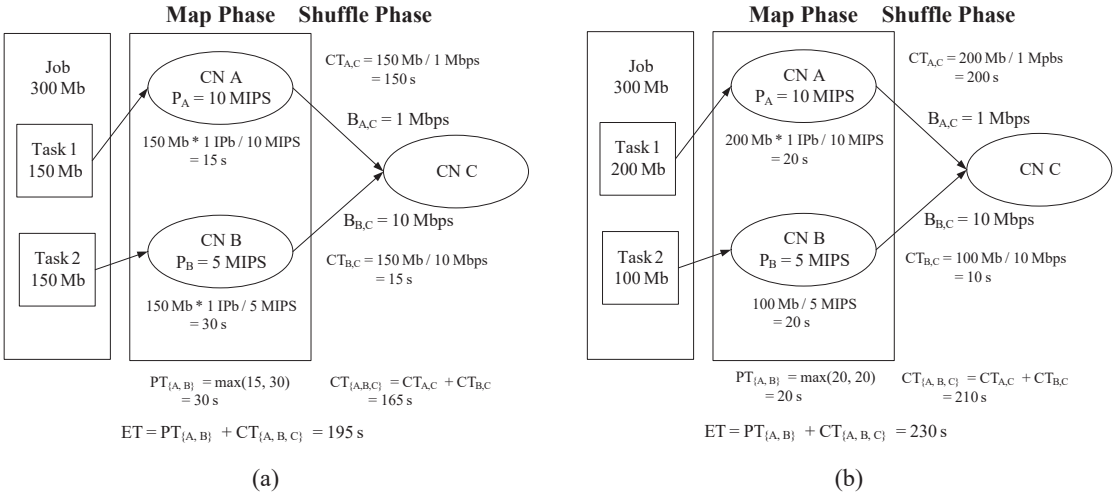

Fig. 4. Challenges of communication load in heterogeneous clouds (a) with and (b) without considering bandwidth.

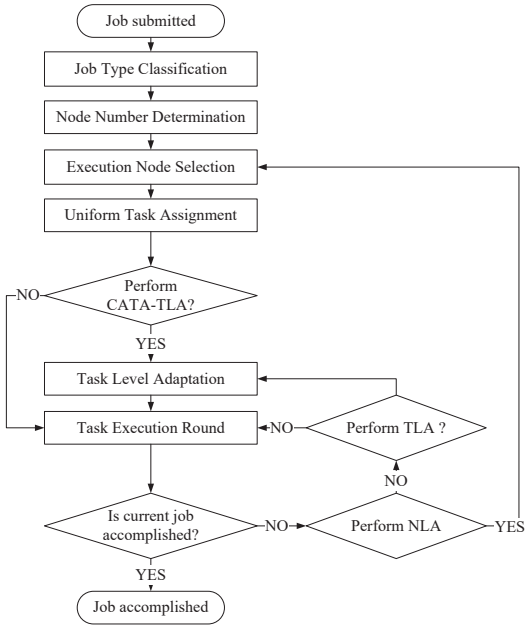

Fig. 5. Flow chart of CATA.

4.1 CATA-NLA

CATA-NLA improves DTA by selecting the appropriate CNs while taking (1) the type of job and (2) the capabilities of CNs into account. CATA-NLA has four steps as described in detail below.

4.1.1 Job type classification

Typically, a processing-intensive job demands more computation resources. In contrast, a communication-intensive job demands more communication resources. If the task assignment does not take the type of job into account, it can prolong the job execution time. Thus, the first step in CATA-NLA is to determine the type of current job.

In CATA-NLA, the job type classification is simplified by determining the communication load,

$$\omega = \min\left(\frac{S_{MOD}}{S_{MID}}, 1\right), \tag{13}$$

where S_{MOD} and S_{MID} are the sizes of mapper input data (MID) and mapper output data (MOD), respectively.[15] A job (e.g., max) with a low communication load (i.e., $\omega \approx 0$) is more likely to be a processing-intensive job. This is because few data are needed to be transmitted from mappers to reducers among CNs. In contrast, a job (e.g., *sort*) with a high communication load (i.e., $\omega \approx 1$) is more likely to be a communication-intensive job.

4.1.2 Node number determination

More CNs can be selected to execute a processing-intensive job for computation speedup. In contrast, a communication-intensive job can be executed by a few CNs to decrease the communication among CNs. Thus, CATA-NLA determines appropriate number of CNs to execute the current job according to its type. Because the shuffle phase is typically the bottleneck in MapReduce,[20,21] we only consider the case of only one CN selected to execute reducers and the reduce processing time is ignored.

The problem is to select $\overline{n}$ CNs among all n CNs in the data center with the objective of minimizing the job execution time. Assume that the current job is executed by x CNs in the map phase and the expected job execution time is

$$EET_x = (1-\omega) \times (MPT_x + RMT_x) + \omega \times ECT_x, \tag{14}$$

where EET_x, MPT_x, and RMT_x are the expected map processing time, reduce processing time, and shuffle communication time, respectively. According to the finding of Chowdhury *et al.*,[19] the map processing time decreases linearly with the number of CNs, but the shuffle communication time increases linearly owing to data transfer among CNs. Thus, in this paper, the expected map processing time and reduce processing time are modeled to be in linear positive correlation with the number of CNs as

$$\begin{aligned} MRT_x &= \frac{L \times C_M}{x \times \overline{P}}, \\ RPT_x &= \frac{L \times C_R}{x \times \overline{P}}, \end{aligned} \tag{15}$$

where $\overline{P}$ is the average processor speed of all CNs in the data center. Moreover, the expected shuffle communication time is modeled to be in linear negative correlation with the number of CNs as

$$ECT_x = x \times \frac{S_{MOD}}{\overline{R}}, \tag{16}$$

where $\overline{R}$ is the average bandwidth among all CNs in the data center. Finally, node number determination is actually the optimization problem to find $\overline{n}$ such that

$$EET_{\overline{n}} = \min_{x=1,\ldots,n} \{EET_x\}. \tag{17}$$

4.1.3 Execution node selection

Although the appropriate number of CNs, denoted by $\overline{n}$, has been determined in Step 2, there is still a problem on how to select the $\overline{n}$ of all CNs in the data center.[22] Since the available resources of CNs are different, the number of $\overline{n}$ CNs must be carefully selected according to the type of current job and the available resources of CNs. Basically, CATA-NLA selects CNs with more computation resources for a processing-intensive job. In contrast, the CNs with more communication resources are selected to execute a communication-intensive job.

CATA-NLA adopts the resource-proportional node selection strategy.[23] Assume that there are K kinds of resources. First, CATA-NLA calculates the selection priority $SP_{i,j}$ of CN i, where $i = 1, 2, ..., n$, for job j by

$$SP_{i,j} = \sum_{k=1}^{K} \left(\omega_{j,k} \times RP_{i,k}\right), \tag{18}$$

where $\omega_{j,k}$ and $RP_{i,k}$ are the weight on resource k for job j and the resource proportion of resource k on CN i, respectively. Since different types of jobs have various resource demands, the node selection strategy, suitable for the type of job j, can be configured as the specific demand weights on resources as $(\omega_{j,1}, \omega_{j,1}, ..., \omega_{j,K})$, where $\sum_{k=1}^{K} \omega_{j,k} = 1$. Moreover, the resource proportion of resource k on CN i is

$$RP_{i,k} = \frac{\frac{r_{i,k}}{R_{i,k}}}{\sum_{k=1}^{K}\left(\frac{r_{i,k}}{R_{i,k}}\right)}, \tag{19}$$

where $r_{i,k}$ and $R_{i,k}$ are the available amount and maximum amount of resource k on CN i, respectively. Then, CATA-NLA can simply sort these CNs by selection priority and select first CNs to execute this job.

4.1.4 Uniform task assignment

In the last step of CATA-NLA, the number of mappers assigned to CN i, denoted by M_i, is determined. Although each mapper can handle one data split at one time, it does not mean that a higher M_i incurs a higher performance, which depends on the capabilities of CN i. On the basis of our experimental results on homogeneous clouds, as shown in Fig. 6, it is found that the job execution time is shortest if the ratio of the numbers of CPU cores to mappers is about 1:2 regardless of the data size and the number of CNs. Thus, in CATA-NLA, M_i is defined as

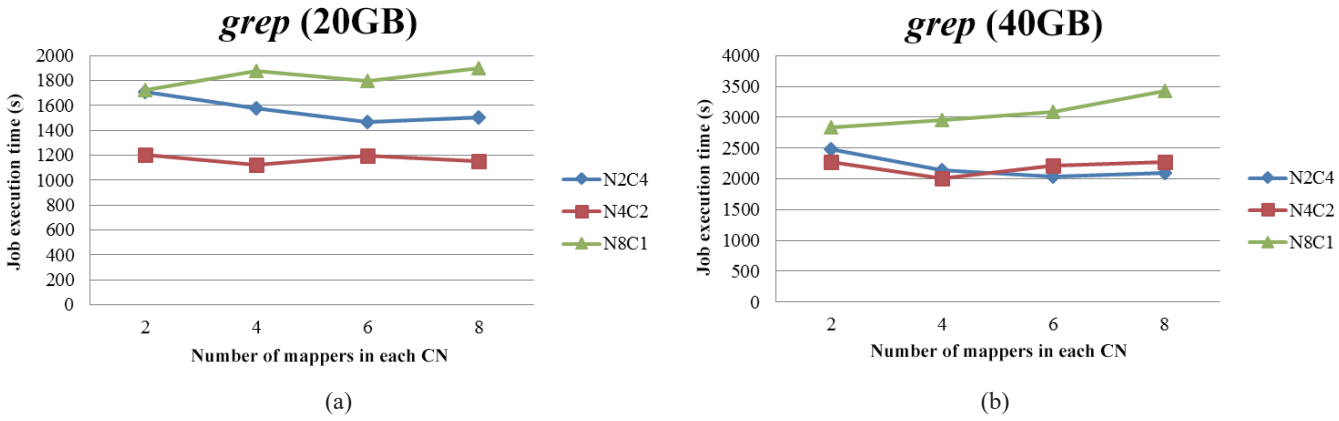

Fig. 6. (Color online) Relationships between the job execution time and the numbers of CNs and CPU cores in each CN. (The notation NnCc means that the experiment is conducted on n CNs and each CN has c CPU cores.) The data sizes are (a) 20 and (b) 40 GB.

$$M_i = 2 \times c_i, \tag{20}$$

where c_i is the number of CPU cores of CN i.

4.2 CATA-TLA

CATA-NLA can improve the performance of MapReduce by selecting the appropriate number of CNs to execute the current job while considering the computation speedup and communication load at the same time. However, the task assignment in CATA-NLA assumes that the cloud is homogeneous. Thus, according to Theorem 1, CATA-NLA can be further improved by reassigning mappers to CNs according to the processor speeds of CNs for approximating PLB.

Thus, the objective of CATA-TLA is to achieve PLB by performing task reassignment. In CATA-NLA, the total number of mappers in each task execution round is the summation of the numbers of mappers in CN i, where $i = 1, ..., \overline{n}$, and is defined as

$$M_R = \sum_{i=1}^{\overline{n}} M_i. \tag{21}$$

To achieve PLB, the number of mappers in CN i must be reassigned according to the processor speed of CN i in CATA-TLA. According to Theorem 1, the reassigned number of mappers to CN i, where $i = 1, ..., \overline{n}$, is defined as

$$m_i = \frac{P_i}{\sum_{j=1}^{\overline{n}} P_j} \times M_R. \tag{22}$$

5. System Evaluation

These experiments are conducted to evaluate the benefits of using CATA. The performances of CATA-NLA, CATA-TLA, and Hadoop DTA are compared in terms of different types of jobs, data sizes, and the numbers of CNs, mappers, and reducers.

5.1 Experimental setup

The jobs selected to represent processing- and communication-intensive jobs are *sort* and *grep*, respectively. The experimental environments are built on two physical computers with a 2.66 GHz quad-core processor, an 8 GB main memory, and a 1 TB hard disk. The virtualization software, XEN, is employed to emulate a homogeneous cloud and a heterogeneous cloud with 8 and 4 CNs, as shown in Tables 2 and 3, respectively.

5.2 Experimental results

In all experimental results, the notation NnMm indicates that the MapReduce configuration for the experiment is composed of m mappers that run on n CNs. Each experimental result is the average of 10 runs of experiment with the same configuration.

Table 2
Homogeneous cloud with 8 CNs.

	CN 1–CN 8
Core	1 core
Memory	1.5 GB
Storage	200 GB

Table 3
Heterogeneous cloud with 4 CNs.

	CN 1	CN 2	CN 3	CN 4
Core	3 cores	1 core	2 cores	2 cores
Memory	3 GB	3 GB	3 GB	3 GB
Storage	200 GB	200 GB	200 GB	200 GB

5.2.1 Experiments in homogeneous cloud

First, CATA-NLA is compared with DTA in the homogeneous cloud as described in Table 2. Since the size of MOD is far less than that of MID in *grep*, a higher computation speedup can thus be expected if more CNs are selected. In contrast, since the size of MOD is equal to that of MID in *sort*, a higher communication load can thus be expected if more CNs are selected.

When the data size is 10 GB, the numbers of CNs for *grep* and *sort* are determined as 8 and 4 by CATA-NLA, respectively. Moreover, since each CN has a single CPU core in this homogeneous cloud, each CN is assigned 2 mappers by Eq. (20) in each task execution round. As shown in Fig. 7, since the size of MOD for *grep* is small, CATA-NLA is intended to execute *grep* on more CNs for a high computation speedup. In contrast, as shown in Fig. 8, CATA-NLA executes *sort* on few CNs for a low communication load. According to the experimental results, the job execution time, under the numbers of CNs and mappers determined by CATA-NLA, is the shortest among all other combinations.

For different data sizes of 10, 20, 40, and 80 GB, the experimental results for *grep* and *sort* are shown in Figs. 9 and 10, respectively. The experimental results show that the advantage of CATA-NLA becomes more obvious with increasing data size.

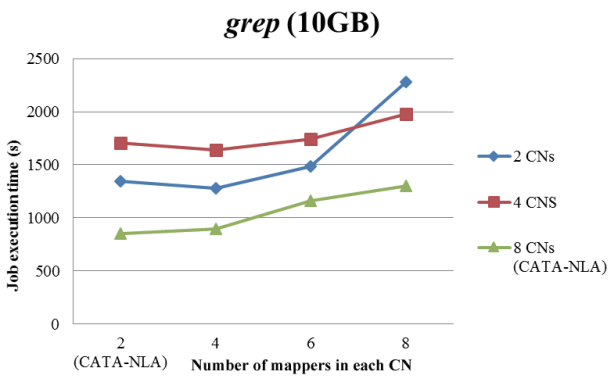

Fig. 7. (Color online) Job execution time of *grep* with different numbers of CNs and mappers in the homogeneous cloud.

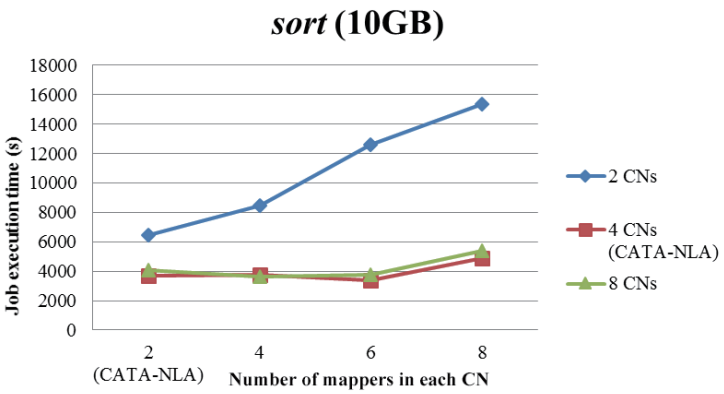

Fig. 8. (Color online) Job execution time of *sort* with different numbers of CNs and mappers in the homogeneous cloud.

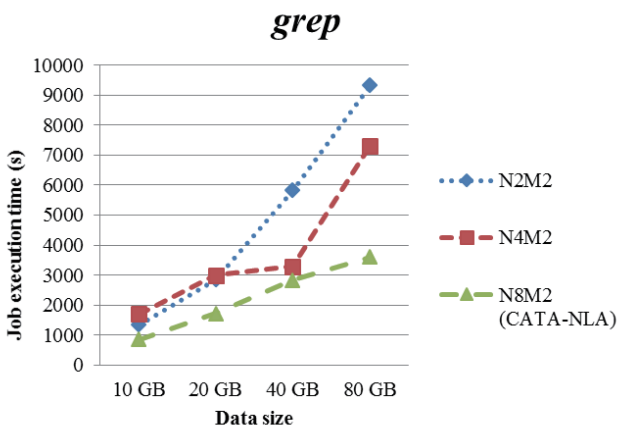

Fig. 9. (Color online) Job execution time of *grep* with different numbers of CNs and different data sizes in the homogeneous cloud.

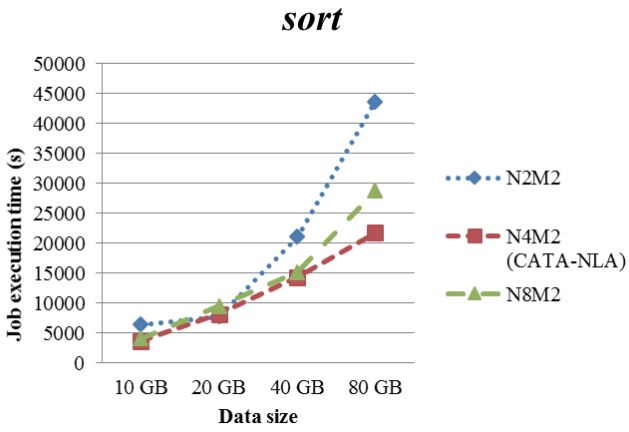

Fig. 10. (Color online) Job execution time of *sort* with different numbers of CNs and different data sizes in the homogeneous cloud.

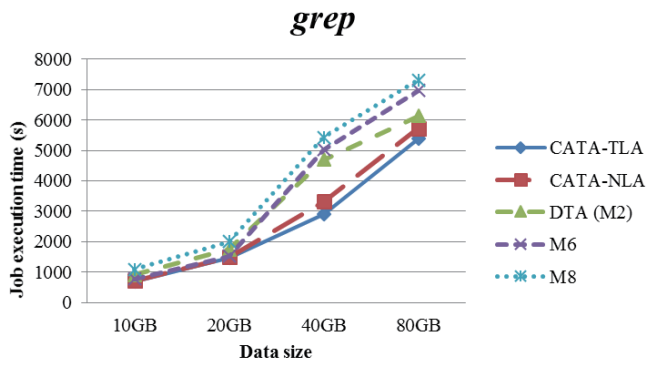

Fig. 11. (Color online) Job execution times of *grep* compared among DTA, CATA-NLA, and CATA-TLA in the heterogeneous cloud.

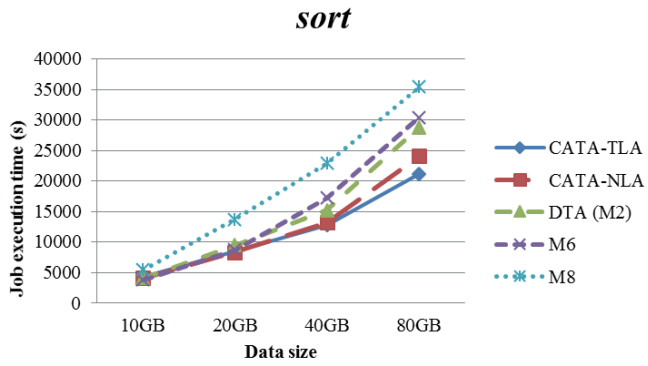

Fig. 12. (Color online) Job execution times of *sort* compared among DTA, CATA-NLA, and CATA-TLA in the heterogeneous cloud.

5.2.2 Experiments in heterogeneous clouds

In this section, CATA-TLA is compared with DTA and CATA-NLA in the heterogeneous cloud as described in Table 3. In this experiment, the numbers of CNs selected for *grep* and *sort* by CATA-TLA are both 4 CNs owing to the heterogeneous cloud, so we only show the experimental results with different numbers of mappers. Because the CNs in heterogeneous clouds have different processor speeds, CATA-TLA reassigns the numbers of mappers to CNs for approximating PLB. As shown in Figs. 11 and 12, CATA-TLA can further improve CATA-NLA by considering the processor speeds of these CNs.

6. Conclusions

For most IoT applications, one of the challenges is to efficiently process massive data streams from a large number of sensor nodes. However, the task assignment approach in the MapReduce-based cloud framework is inefficient to cope with the enormously increasing size of sensor data and the heterogeneity of CNs in the data center. In this paper, the CATA approach is proposed to improve MapReduce performance. CATA can assign tasks while considering different resource demands according to the type of job. Moreover, CATA also takes the capabilities and available resources of CNs into account while performing task assignment. There are two contributions of CATA. First, CATA can adaptively assign tasks to CNs according to the capabilities and available resources of CNs with the objective of achieving PLB. Thus, the job execution time is efficiently reduced because of the balanced load among CNs. Second, CATA does not require any specific hardware platform. The experimental results show that CATA can reduce the job execution time by 10–40% whether the job is processing- or communication-intensive.

Currently, CATA is based on the Hadoop first-in first-out (FIFO) scheduler, so the decision of task assignment is only for a single job at one time. Thus, it is still possible to incur starvation if the high-priority and long-running jobs exhaust cloud resources. In the future, we will modify CATA for other efficient schedulers, such as fair, capacity, and Hadoop on Demand schedulers.[24]

Acknowledgments

This paper is based on the work partially supported by the Ministry of Science and Technology (MOST), Taiwan, R.O.C., under Grant Nos. MOST 106-2221-E-156-001, 106-2221-E-006-005, 105-2221-E-156-004, and 105-2221-E-006-141.

References

1 R. Maggiani: Proc. IEEE Int. Professional Communication Conf. (IEEE, 2009) p. 1.
2 W. Tom: Hadoop: The Definitive Guide (O'Reilly, Sebastopol, 2009) p. 193.
3 M. M. Rafique, B. Rose, A. R. Butt, and D. S. Nikolopoulos: ACM SIGOPS Operating Syst. Rev. **43** (2009) 25.
4 J. Gautam, H. Prajapati, V. Dabhi, and S. Chaudhary: Proc. 2015 IEEE Int. Conf. Electrical, Computer and Communication Technologies (IEEE, 2015) p. 1.
5 M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica: Proc. 8th USENIX Symp. Operating Systems Design and Implementation (USENIX, 2008) p. 29.
6 W. Hu, C. Tain, X. Liu, H. Qi, L. Zha, U. Liao, Y. Zhang, and J. Zhang: Proc. 6th Int. Conf. Semantics Knowledge and Grid (IEEE, 2010) p. 135.
7 Y. Yao, J. Tai, B. Sheng, and N. Mi: IEEE Trans. Cloud Comput. **3** (2014) 411.
8 Z. Guo and G. Fox: Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing (IEEE/ACM, 2012) p. 714.
9 L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica: Proc. 10th ACM Workshop on Hot Topics in Networks (ACM, 2011) Article No. 22.
10 J. Dean and S. Ghemawat: MapReduce: Proc. 4th USENIX Symp. Operating Systems Design and Implementation (USENIX, 2004) p. 137.
11 D. Jiang, B. C. Ooi, L. Shi, and S. Wu: Proc. VLDB Endowment **3** (2010) 472.
12 K. Kambatla, A. Pathak, and H. Pucha: Proc. 1st Workshop on Hot Topics in Cloud Computing (USENIX, 2009) Article No. 22.
13 H. Herodotou and S. Babu: Proc. VLDB Endowment **4** (2011) 1111.
14 G. Wang, A. Khasymski, K. R. Krish, and A. R. Butt: Proc. Int. Conf. Parallel and Distributed Systems (IEEE, 2013) p. 299.

15 C. Tian, H. Zhou, Y. He, and L. Zha: Proc. Int. Conf. Grid and Cooperative Computing (IEEE, 2009) p. 218.
16 R. Chen, H. Chen, and B. Zang: Proc. 19th Int. Conf. Parallel Architectures and Compilation Techniques (IEEE, 2010) p. 523.
17 F. Ahmad, S. Lee, and M. Thottethodi: J. Parallel Distrib. Comput. **73** (2013) 608.
18 B. Ghit and D. Epema: Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing (IEEE/ACM, 2016) p. 11.
19 M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica: ACM SIGCOMM Comput. Commun. Rev. **41** (2011) 98.
20 M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat: Hedera: Proc. 7th USENIX Conf. Networked Systems Design and Implementation (USENIX, 2010) p. 19.
21 A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta: Commun. ACM **54** (2009) 95.
22 G. K. Archana and V. D. Charkravarthy: Proc. 2015 Int. Conf. Computing and Communication Technologies (IEEE, 2015) p. 368.
23 C.-L. Chen, J.-W. Lee, W.-T. Su, M.-F. Horng, and Y.-H. Kuo: Int. J. Ad Hoc Ubiquitous Comput. **3** (2008) 224.
24 C. Wang, Y. Peng, J. Liu, M. Tang, G. Liu, J. Feng, and P. You: Proc. 9th IEEE Int. Conf. Networking, Architecture, and Storage (IEEE, 2014) p. 118.

About the Authors

Wei-Tsung Su received his B.S. degree in Information and Computer Science from Chung Yuan Christian University, Taiwan, in 2000 and his Ph.D. degree in Computer Science and Information Engineering from National Cheng Kung University, Taiwan in 2008. From 2009, he is an assistant professor at Aletheia University, Taiwan. His research interests are in cloud computing and network security.

Wei-Fan Pan received his B.S. degree in Information Application from Aletheia University, Taiwan, in 2009 and his M.S. degree in Computer Science and Information Engineering from Aletheia University, Taiwan, in 2011. He is a system engineer in Chung Hwa Pulp Corporation. His research interests are in distributed systems and parallel computing.

Chao-Chun Chen is an associate professor in the Institute of Manufacturing Information and Systems (IMIS) and the Department of Computer Science and Information Engineering (CSIE), National Cheng Kung University, Taiwan. He received his Ph.D. degree in the Department of Computer Science and Information Engineering at National Cheng Kung University, Taiwan, in 2004. His research interests include distributed data management, cloud manufacturing, artificial intelligence, system integration and optimization, and databases.