An Adaptively Speculative Execution Strategy Based on Real-Time Resource Awareness in a Multi-Job Heterogeneous Environment

Qi Liu[1], Weidong Cai[1], Qiang Liu[2], Jian Shen[1], Zhangjie Fu[1], Xiaodong Liu [3] and Nigel Linge[4]
[1] Nanjing University of Information Science and Technology
Nanjing, Ningliu Road 219, China
[e-mail: qi.liu@nuist.edu.cn, caiweidongsuzhou@163.com, s_shenjian@126.com, wwwfzj@126.com]
[2] School of Computer, Hunan University of Technology
Zhuzhou, Hunan, China
[e-mail:liuq1016@126.com]
[3] School of Computing, Edinburgh Napier University
10 Colinton Road, Edinburgh EH10 5DT, UK
[e-mail: x.liu@napier.ac.uk]
[4] School of Computing, Science and Engineering, The University of Salford
Salford, Greater Manchester, M5 4WT, UK
[e-mail: n.linge@salford.ac.uk]
*Corresponding author: Weidong Cai

Received June 18, 2016; revised October 25, 2016; accepted December 23, 2016; published February 28, 2017

Abstract

MapReduce (MRV1), a popular programming model, proposed by Google, has been well used to process large datasets in Hadoop, an open source cloud platform. Its new version MapReduce 2.0 (MRV2) developed along with the emerging of Yarn has achieved obvious improvement over MRV1. However, MRV2 suffers from long finishing time on certain types of jobs. Speculative Execution (SE) has been presented as an approach to the problem above by backing up those delayed jobs from low-performance machines to higher ones. In this paper, an adaptive SE strategy (ASE) is presented in Hadoop-2.6.0. Experiment results have depicted that the ASE duplicates tasks according to real-time resources usage among work nodes in a cloud. In addition, the performance of MRV2 is largely improved using the ASE strategy on job execution time and resource consumption, whether in a multi-job environment.

Keywords: MapReduce, Speculative Execution, Real-Time Resources Awareness, Multi-Job Distribution

This work is supported by the NSFC (61300238, 61300237, 61232016, 1405254, and 61373133);
Marie Curie Fellowship (701697-CAR-MSCA-IFEF-ST); the 2014 Project of six personnel in Jiangsu Province under Grant No. 2014-WLW-013; the 2015 Project of six personnel in Jiangsu Province under Grant No. R2015L06;Basic Research Programs (Natural Science Foundation) of Jiangsu Province (BK20131004) and the PAPD fund.

https://doi.org/10.3837/tiis.2017.02.004 **ISSN : 1976-7277**

1. Introduction

Cloud computing is being widely investigated due to its features of large-scale, virtualization, failure tolerance and control among connected components, and synchronized communication. Requirements of a distributed system with on-demand services, computing abilities and storage resources become increasingly urgent [1], latest report has been concentrated on task allocation in dynamic cloud environments [2]. MapReduce [3], as a popular, distribute programming model in a cloud environment utilizes map and reduce procedures transparently to facilitate datasets processing.

Large-scale data analysis in a cloud is complex due to the diversity of structural complexity, resource requirements, and delivery deadlines. New challenges of job scheduling, workload management, and program design need to be overcome to improve the performance of the framework. To relieve the difficulty in implementing complex data analysis or data mining program, several projects, e.g., Pig [4][5] and Hive [6][7] built on the MapReduce engine in a Hadoop environment, have been launched.

However, poor load-scheduling strategies and speculative execution strategy are still damaging the performance of MapReduce, which can consume much more computing resources and execution time than expected. While theoretically infinite computing resources can be provided in a cloud, the unreasonable increment of mappers/reducers cannot achieve process efficiency, and may waste more storage to complete. Many optimization schemes have been proposed [8-25].

In this paper, a new strategy called Adaptive Speculative Execution (ASE) is proposed to implement speculative execution. Based on the real-time resource condition in Hadoop 2. X, the resource can be represented by the container. Such a strategy can make full use of resources and try to start a backed-up task at a correct timestamp. A practical Hadoop environment containing the ASE has been implemented in our laboratory (Hadoop-ASE). According to our results, the ASE strategy has depicted shorter execution time than MCP.

Our contributions in this paper include:

1) Because of the difference between different nodes, running information of different task is stored separately according to the node that it runs on, while other strategies use cluster average time to estimate the running time.

2) A linear regression algorithm is applied in our strategy to estimate the finishing timestamp of the current phase.

3) Real-time resource usage situation is fully taken into adequate consideration when starting a backup task while other strategies have not considered it in.

The rest sections of the paper are organized as followed. Related work is given in Section II. Section III gives an introduction to previously proposed strategies and their pitfalls and describes the detailed design of our speculative execution strategy. In Section IV, testing environment, and corresponding scenarios are designed for the verification and evaluation of the Hadoop-ASE. Finally, conclusion and future work on our speculation execution strategies in a cloud platform are discussed in Section V.

2. Related Work

Many distributed systems based on Hadoop are also researched. Mars were developed to run on NVIDIA and AMD GPUs as well as multicore CPUs and implemented in Hadoop [8]. Mars shows simple and familiar MapReduce interface to developers rather than the complex GPU computation. However, the implementation is much more complex while in a cluster, the performance of GPUs and CPUs. A distributed computing framework called PrIter was developed, where the prioritized execution of iterative computations is supported in it [9]. But, the PrIter only adapts iterative algorithms, not for all algorithms. GGB and GR were implemented in [10] by Wang et al. They extended the idea of the local greedy algorithm aiming at efficient cluster distribution of the budget and costing minimum scheduling length in GR.

Due to the load deflection of the tasks, it is hard to synchronize the finishing time of all tasks located in the data nodes of MapReduce system. The speed of a job will be largely pulled down by the lately finished task [11]. Various of research efforts have been conducted on the partition of reduce tasks. To facilitate reduce partition processes, historical data [12] and sampling results [13] have gained extensive attention. Though load balancing can be achieved dynamically according to these methods, none of them were verified in a real Hadoop system. Fan et al. proposed a prediction model based on SVM [14]. For the purpose of optimizing MapReduce performance in heterogeneous environments, an adaptive algorithm called HAP is designed to give out the right workload for a certain node. However, inputs of a reduce task need to be divided into smaller splits for finding appropriate input data volume, which causes extra time when re-merging the splits. Liu et al. presented a new method for achieving load balancing in both running time and disk occupation [15].

Virtualized deployment efficiency is also concerned. [16] proposed a general method for estimating resource requirements when some applications are running in a virtualized environment. Various of virtualization overheads have been proposed. Based on a regression model, local system configuration files are mapped to a virtualized one. [17] presented a prediction model for adaptive resource provision in a cloud. This model can predict dynamic demands of resources when VM instances are launched.

In cloud systems, a method for performance evaluation and load efficiency has also been explored. PQR2 was proposed in [18], which can provide an accurate assessment of performance for distributed applications in a cloud environment. [19] presented a model for predicting the resource consumption of MapReduce process. This model can achieve an accurate resource forecasting based on classification and regression trees. Also, the shuffle phase has gained extensive attention. Virtual shuffling was proposed in [20], where a novel virtual shuffling strategy was put forward to improve the efficiency of data movement during shuffling intermediate data.

[21] proposed an alternative technique called Dynamic Hadoop SlotAllocation by modifying the slot-based model. This technique elevates the slot allocation constraint of the native Hadoop, by which slots are permitted to be reallocated to any task freely according to the real-time slot requirement. Straggler is also introduced in the paper as the main reason resulting in the low efficiency of MR. Reasons causing stragglers include hardware faulty and software misconfiguration. Then, the stragglers are divided into two types, called hard straggler and a soft straggler. Speculative execution strategy is also optimized to solve soft straggler. Though their technique in the paper improved the performance a lot, it cannot be applied to MRV2, whereas the resource manager in MRV2 i.e. container can be dynamically

assigned to a map task and reduce task, even to an application master at the very beginning of a job.

Many other researchers are also studying optimizing speculative execution strategies in MapReduce. Zaharia et al. [22] proposed a modified version of speculative execution strategy, called Longest Approximate Time to End (LATE) algorithm. A different metric is proposed in LATE, to estimate the tasks for backing up. The remaining time of the tasks is directly estimated while the progress made by a task is not considered. Through this method, a more clear assessment of the impact that struggling tasks have on the overall job response time can be evaluated more correctly. However, the LATE suffers from unstable execution time at each stage due to the parameter *std* used in LATE not able to represent all cases. MCP [23] was therefore proposed, which used average progress rate to identify slow tasks. According to the results, struggles can be appropriately judged when data skew among the tasks. However, MCP still contains a lot of pitfalls, and it can only be used in MR, not including MRV2 that has adopted a newly resource management framework. Liu et al. proposed a new model by collecting real-time data, which has achieved higher accuracy [24]. A dynamic strategy has been proposed based on an exponential smoothing model for each phase [25].

All research work above is focused on improving the resource usage efficiency and job execution efficiency in a distributed environment, and many achievements have been made. However, in a Heterogeneous environment, reasonable speculative execution is still a hard problem. In LATE, MCP or other latest speculative execution strategies, concurrent situations of real-time resources usage not being well handled is still of an urgent challenge.

3. An Adaptively Speculative Execution Strategy

3.1 Time Prediction

1) Linear Regression and Exponential Smoothing to Estimate Remaining Time of Current Task

Table 1. Symbol Definition

T_{rem}	Remaining time of the current task
T_{cp}	Remaining time of the current phase
T_{fp}	Remaining time of the following phases
T_{back}	Remaining time of a backup task
TaskId	Id the of the current task
NodeId	Node Id of the current node
Avg_{Sort}	Average time of sort phase
Avg_{Reduce}	Average time of reduce phase

In a MRV2 cluster, a name node divides the input files into multiple map tasks after a job is submitted, and then schedules both map and reduce tasks to data nodes. A data node runs tasks in its containers and keeps updating the tasks' progress to the master by periodic heartbeat. **Table 1** shows the symbols will be used in the following paragraphs.

Map phase, as the first stage of MapReduce, its processing speed can directly affect shuffle phase velocity. Shuffle speed depends on the map output. As described in the original MRV2, shuffle phase will begin after 5% of map stage has completed. So, the speed of shuffle phase is always changing. At the beginning of shuffling, because of sufficient output of map phase,

speed is relatively fast. With the continuation of this process, the speed will be significantly slower. If the amount of input data is not enough, shuffle stage data will remain in the waiting state until there is sufficient input.

$$T_{rem} = T_{cp} + T_{fp} \tag{1}$$

Concluding these, we calculate the remaining time of current task according to (1). However, a linear regression algorithm is used in our strategy. When a heartbeat reaches, we firstly detect which phase the task is at. Then, we judge whether the data has existed. If exists, we update the data using current data. Otherwise, data is directly added to a data set. The data set is generated by each task divided by task number. Progress and timestamp are stored as a key-value pair such as $ProgressPair(progress, timestamp)$. When current task is a map task, linear regression is directly applied to estimate the finishing timestamp of the current task according to the latest five groups of *ProgressPair*, as shown in (2).

$$T_{rem} = Regression(map, TaskId) \tag{2}$$

However, when current task is a reduce task, we are required to judge which stage is the current task attempt in. As describe before, if the current task is in shuffle phase, the finishing time is easily influenced by the output of map stage. So an exponential smoothing is necessary when estimating the finishing timestamp of shuffle phase. Moreover, running time of next 2 phases, sort and reduce, is estimated by the historical consuming time of tasks running on the same node in the current job. The method for Estimation of next phases is reasonable because data volume assigned is relatively even and time for the same node to process similar volume would not vary too much. Details are shown in equation (3) and (4).

$$T_{cp} = F(t) = \alpha * F(t-1) + (1-\alpha) * E(t-1) \tag{3}$$

$$T_{fp} = Avg_{sort}(TaskId, NodeId) + Avg_{reduce}(TaskId, NodeId) \tag{4}$$

In (3), $F(t)$ and $E(t)$ are the estimated value and real value are time t while α is an effect factor, in this paper, it is set 0.2 according to [22]. Avg_{sort} and Avg_{reduce} represent the average running time of sort and reduce phase, they would be in detail in the following part.

Similarly, we can get estimated time when current phase is sort or reduce as shown in equation (5) and (6).

$$T_{rem} = Regression(sort, TaskId) + Avg_{reduce}(TaskId, NodeId) \tag{5}$$

$$T_{rem} = Regression(redeuce, TaskId) \tag{6}$$

Equation (5) shows the method of estimating the remaining time when current phase is sort.Moreover, equation (6) demonstrates the method for that current phase is reduce. Detailed algorithm is presented in Algorithm 1.

2) Host Classification and Time Requirement of Backup Task

When a task finishes, our strategy automatically stores task running time on some node. If the current task is a map task, stored data will be its hostname and average duration. Otherwise, while it is a reduce task, the duration of each phase will be recorded as well as the total duration. We have established a storage mechanism to help us store information while having a low space occupation. The detailed storage method is shown in **Fig. 1**.

Algorithm 1. Estimate Remaining Time of Current Task

Input: TaskId Tid **, NodeId** Nid **, TaskType** $Type$ **, Phase** P

Output: Remaining running time T_{rem}

1 Get Tid of current task

2 If Tid equals Map

3 $T_{cp} = Regression(map, Tid), T_{fp} = 0$

4 Else

5 If P equals shuffle

6 $T_{cp} = F(t) = \alpha * F(t-1) + (1-\alpha) * E(t-1)$

7 $T_{fp} = Avg_{sort}(Tid, Nid) + Avg_{reduce}(Tid, Nid)$

8 ElseIf p equals sort

9 $T_{cp} = Regression(p, Tid)$

11 ElseIf P equals reduce

12 $T_{cp} = Regression(p, Tid), T_{fp} = 0$

13 EndIf

14 $T_{rem} = T_{cp} + T_{fp}$

15 Return T_{rem}

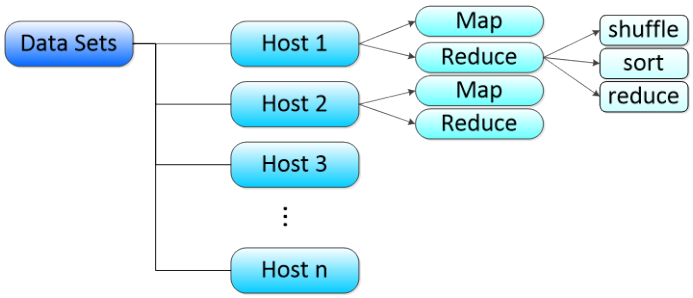


Fig. 1. Storage Mechanism

As shown in **Fig. 1**, a data set is found according to its hostname at the beginning of a storage procedure. Data generated by the tasks that run on the same node will be stored in the same collection. Every storage area is further divided into two sub-storage areas called Map and Reduce. If the finished task is a map task, the running time is directly recorded in Map. while it is a reduce task, detailed running information, containing shuffle time, sort time and reduce time, is respectively written in 3 sub-blocks.

While detailed average running time of a task on some node needs to be known (such as $Avg_{shuffle}$, Avg_{sort} or Avg_{reduce}), hostname is first gotten. If running time of a task is unreachable, overall running time of different hosts can replace the detailed one. Otherwise, the average time of certain phase is directly obtained. Detailed algorithm is shown in Algorithm 2.

Algorithm 2. Get the estimated running time of a backup task

Input: TaskId Tid **, NodeId** Nid **, TaskType** $Type$

Output: Running time of a backup task T_{back}

1 Get hostname according to Tid of current task
2 Get the data set that hostname mapping to
3 If the dataset is empty
4 If $Type$ equals Map
5 Get the average running time of different hosts, Recorded as Avg_{map}
And $T_{back} = Avg_{map}$
6 ElseIf $Type$ equals Reduce
7 Get the per phase average running time of different hosts,
Recorded as $Avg_{shuffle}$ Avg_{sort} and Avg_{reduce}
8 $T_{back} = Avg_{shuffle} + Avg_{sort} + Avg_{reduce}$
9 End If
10 Else
11 If $Type$ equals Map
12 Get the average running time of the same host as Avg_{map} and $T_{back} = Avg_{map}$
13 $T_{back} = Avg_{map}$
14 Else
15 Get sum of per phase average running time of the same host, Recorded as T_{back}
16 End If
17 End If
18 Return T_{back}

3.2 3.2 An Adaptive Method of Selecting a Task to Backup

1) Necessity of a Dynamic Selection Method

When having detected a struggle, most speculative execution strategies in MRV1, e.g., LATE would kill the native task and start a new one, which increases the overall execution time if it incorrectly starts a backup task. So in MRV2, a strategy called Hadoop-Original is applied to take the trade-off between the execution time and space. Due to data locality, the allocation of map tasks, their sizes and overall execution sequence in each node are all determined. Such formation can be depicted in **Fig. 2**, but it can result in execution delay in low-spec nodes under certain circumstances. For instance, as shown in **Fig. 3**, if task A1 is detected to be a struggle, a task A1' is backed up according to the original MRV2. Since backup tasks have a higher priority in MRV2, A1' will be assigned ahead of other waiting tasks, i.e., ahead of B2 in this case. If Node B happens to be a low-spec node, it will consume more time to finish, which can cause the overall execution time to be greatly delayed.

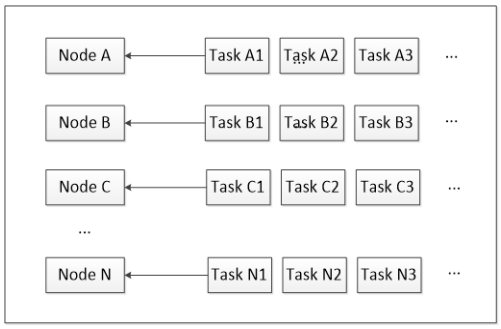


Fig. 2. Original Task Queue

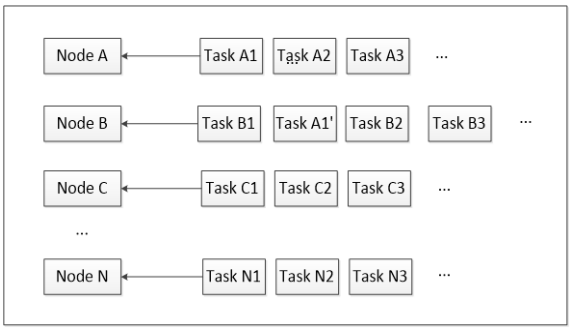


Fig. 3. Task Queue after Backing up

2) An Adaptively Dynamic Method Based on Real-time Resource

In our method, three cases are defined, as follows:

Case A: Tasks to be processed in a node are more than its available resources, as shown in **Fig. 4**.

Case B: Tasks to be processed in a node can right utilize its available resources as shown in **Fig. 5**.

Case C: Tasks to be processed in a node are less than its available resources, as shown in **Fig. 6**.

As a speculator starts on some node, it continuously detects which case the node is at. Corresponding procedures are then adapted. Detailed procedures are shown in Algorithm 3.

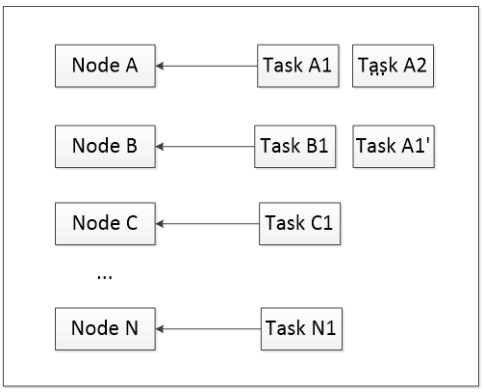


Fig. 4. Case A

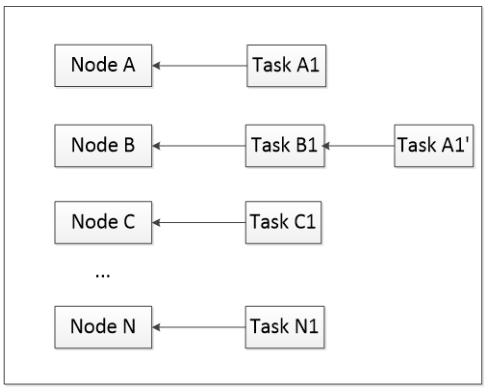


Fig. 5. Case B

When a node is in Case A, the corresponding procedure will kill the struggle and starts a new task when the time saved by new task reaches a half of native remaining time. Because of its higher priority, the backup task will be immediately run at the original node.

As Case B is detected as shown in **Fig. 5**, which represents resource on every node is just enough, corresponding steps are taken as follow:

(1) Estimate remaining time of all running tasks based on the node average consuming time;

(2) Select the task which has the longest remaining time, mark as A1 and remaining time of it T_{rem};

(3) Find the task that has the shortest remaining time T_{rem}' and marks it as B1;

(4) Try to speculate the running time if A1 runs on the node that has shortest remaining time, mark it as T_{new} .When $T_{rem}'+T_{new} < T_{rem}$ is satisfied, then a backup task is directly launched;

Algorithm 3. An adaptive algorithm for speculative execution

Input: TaskId**,** Backup Set ***(Bus)***
Output: ***Bus***

```
Get the current case according to T_id, called Case
If Case matches A
For each running task
If T_back < T_rem and (T_rem - T_back) / T_rem <1
Add T_id to Bus
   End If
  End For
ElseIf Case matches B
  Get the task has the longest and shortest remaining time Tl_rem , Ts_rem
  If Ts_rem + T_back < Tl_rem
    Add the T_id to Bus
  End If
ElseIf Case matches C
 Time_min=0
  For each running task
    If T_back < T_rem and (T_rem - T_back) / T_rem <20%
      Add T_id to Bus
    ElseIf  T_rem <Least Time
      Recording the T_id, Time_min = T_rem
    End If
  End For
  Add T_id to Bus
End If
Return Bus
```

Case C represents the state that there is enough resource in the cluster, as shown in **Fig. 6 (a)**. At this time, because data locality has a higher priority on scheduling, every node has a similar volume of tasks. However, tasks on the low-performance node (e.g. Node A) will not finish on time in a heterogeneous environment, then, the following tasks would be assigned to other nodes that have enough resources, as shown in **Fig. 6 (b)**. A decision to start a backup task is made counting on 20% of the time can be saved compared with no backup. If the condition is satisfied, Task A1 would be backup to Node B as shown in **Fig. 6 (c)**.

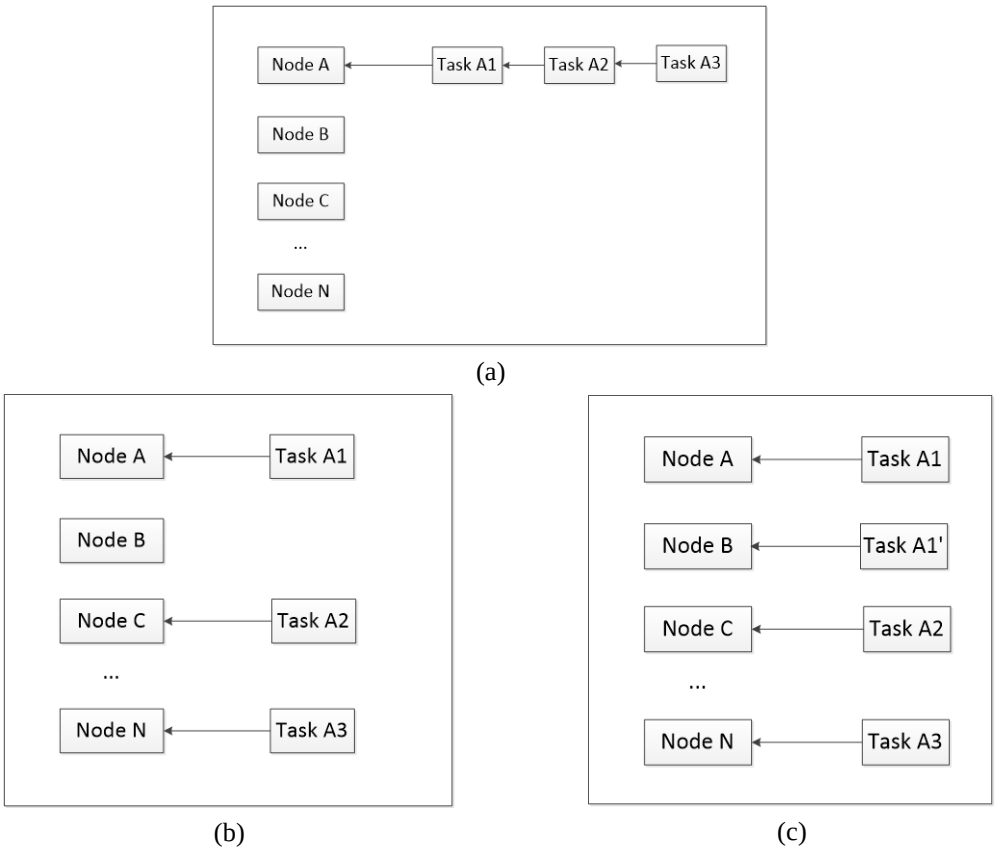


Fig. 6. Case C (a) Detected free resources existing (b) Tasks transferred to other nodes which have free resources (c) Low-task backed up to another free node

4. Experimental Classification Results and Analysis

In order to estimate the performance of the ASE strategy, a practical environment was deployed consisting of a personal computer and a server being equipped with 288 GB of memory and a 10 TB SATA hard drive. The personal computers are equipped with 12GB of memory, a single 500GB disk and four 2.4GHz Intel® Core™ 2 Duo (TM) processors. Eight virtual machines were set up in the server and given different amounts of memory and a different number of processors, as shown in **Table 2**. These machines were connected to the physical network by a switch according to bridge mode.

Table 2. Detailed information of each virtual machine

	Node1	**Node2**	**Node3**	**Node4**	**Node5**	**Node6**	**Node7**	**Node8**
Memory(GB)	10	8	8	8	4	4	18	12
Core processors	8	4	1	8	8	4	4	8

Virtual machines and PC were run as the data nodes and the server as the name node. WordCount, Grep and Sort were employed to evaluate the performance of the ASE strategy.

The Purdue Benchmarks Suite were used for the provision of the test-bed environment with Tarazu datasets [26] as testing input. Detailed configuration information of three testing cases is shown in **Table 3**.

Table 3. Input of these applications

	WordCount	Grep	Sort
Input(GB)	50	50	30
Number of Mappers	200	132	200
Number of Reducers	16	18	15

The ASE strategy is compared with Hadoop-MCP, Hadoop- Original and Hadoop-None (with the speculative execution feature being disabled). All tests were executed for six times for each strategy. Average, worst, and best cases have been calculated for fair comparison.

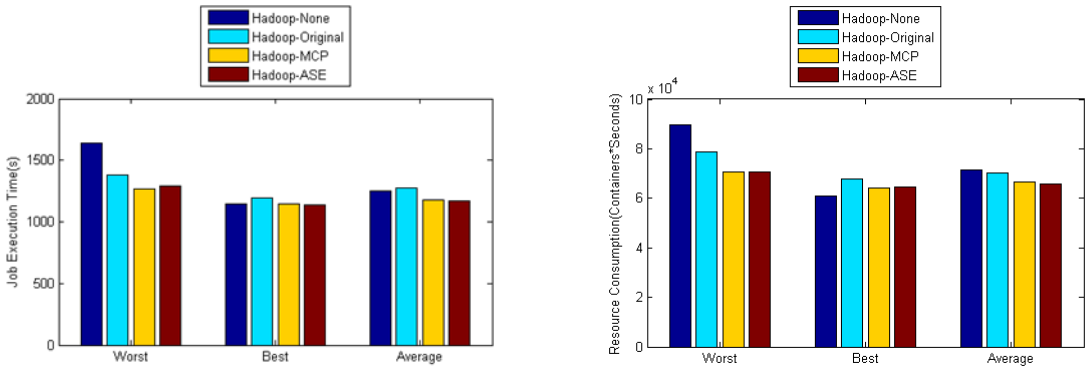
Traditional strategies only evaluate the job execution time. In this paper, resource consumption is also considered, which represents the number of containers and the time of the container occupied. So an equation is gotten as shown in (7).

$$Consumption = Containers * Seconds \tag{7}$$

The improvement is therefore calculated by the job execution time and resource consumption of diffident strategy as follows:

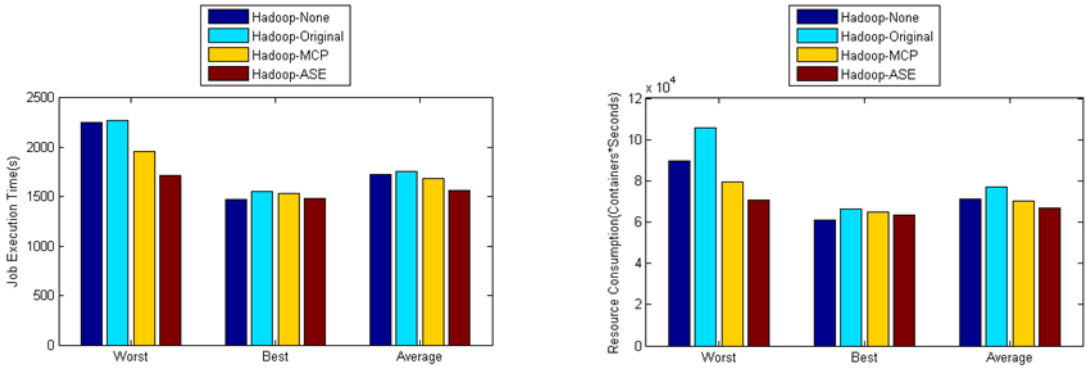
$$Improvement = \frac{OtherStrategy - ASE}{OtherStrategy} \tag{8}$$

Fig. 7, **Fig. 8** and **Fig. 9** show the job execution time and resource consumption under the condition for a single-job scenario. It can be seen that the ASE strategy has an improvement over other strategies. For WordCount, Hadoop-ASE finishes jobs 8% faster than MCP, at the same time resource consumption is also not bigger than MCP. For Sort, Hadoop-ASE has a similar performance with MCP. However, it saves more than 3000 resources. Then, for Grep, Hadoop-ASE saves 4% time and also has a little improvement over resource consumption.



a) Job Execution Time b) Resource Consumption

Fig. 7. WordCount in a single-job heterogeneous environment



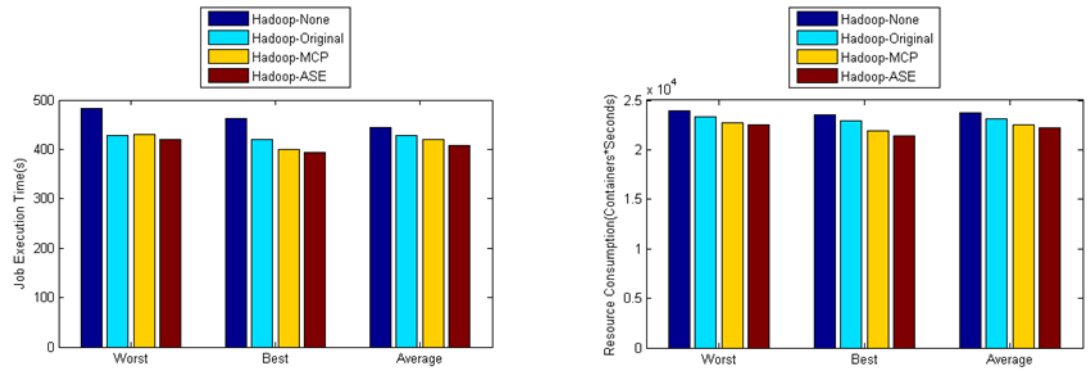
a) Job Execution Time b) Resource Consumption

Fig. 8. Sort in a single-job heterogeneous environment

From the three figures, we can see the performance of our strategy does not have a very excellent performance under single job environment. However, when it comes to a multi-job environment, Hadoop-ASE has a very outstanding performance, no matter in job execution time or resource consumption.

We also operated another experiment that we submitted the same job after that the first one has run for 15 seconds. In this experiment, two jobs are put into a group, and five groups of an experiment for every sample are operated. This experiment simulates a multi-job environment. **Fig. 10**, **Fig. 11** and **Fig. 12** show the performance of Hadoop-ASE under a multi-job environment for different samples.

Fig. 10 depicts the performance of WordCount. Hadoop-ASE finishes jobs 13.5% faster than Hadoop-None, 3.3% than Hadoop-Original and Hadoop-MCP. On resource consumption, Hadoop-ASE achieves 9.9% less resource occupation than Hadoop-None, 6.0% less than Hadoop-Original and Hadoop-MCP (about 10000 resources).



a) Job Execution Time b) Resource Consumption

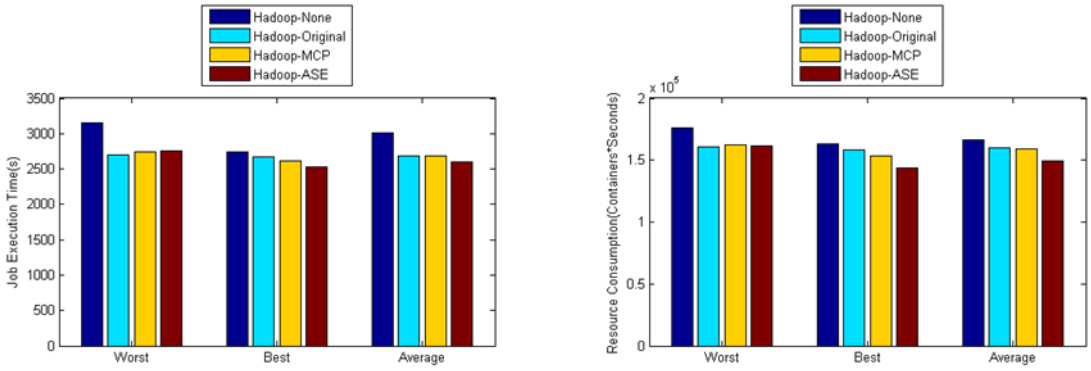
Fig. 9. Grep in a single-job heterogeneous environment

Fig. 11 shows the performance of Sort gotten through different strategies. We can see that on average, Hadoop-Original and Hadoop-MCP has a poor performance in a multi-job environment, both on execution time and resource consumption. Hadoop-ASE shows an excellent performance that it finishes job 16.4% faster than Hadoop-None, 21.9% than

Hadoop-Original, and 24.1% than Hadoop-MCP. Moreover, on resource consumption, it saves 18.0% compared with Hadoop-None, 22.9% compared with Hadoop-Original, and 23.5% compared with Hadoop-MCP.

Fig. 12 describes Grep results gotten through different strategies. Hadoop-ASE finishes job 8.1% faster than Hadoop-None, 10.2% faster than Hadoop-Original, and 14.5% time Hadoop-MCP. Besides, on resource consumption, it saves 10.4% compared with Hadoop-None, 10.3% compared with Hadoop-Original, and 13.2% compared with Hadoop-MCP.

To explore the reason why Hadoop-ASE has better performance than others, we take WordCount as an example and analyze the running logs of these four strategies. Detailed information is shown in **Table 4**.

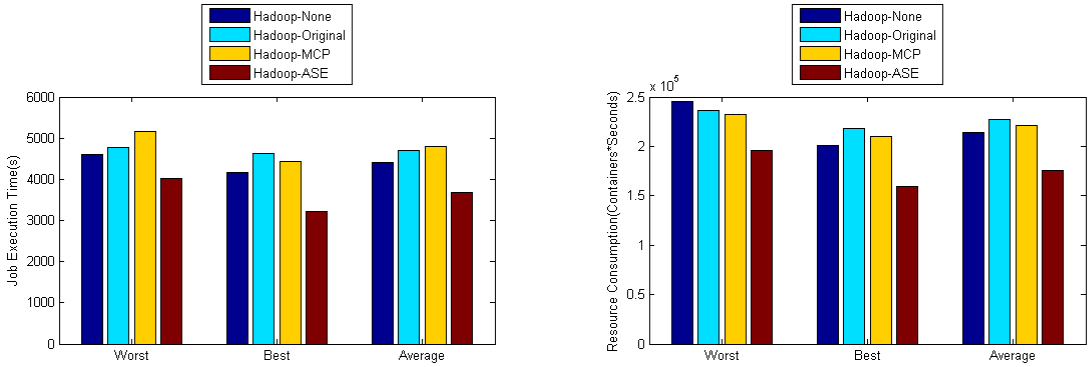


a) Job Execution Time b) Resource Consumption

Fig. 10. WordCount in a multi-job heterogeneous environment

Table 4. Backup task information

Strategy	Sum of Backup		Success of Backup		Backup success rate (%)	
	Map	Reduce	Map	Reduce	Map	Reduce
Hadoop-Original	90	0	64	0	71.1%	-
Hadoop-MCP	108	0	60	0	55.6	-
Hadoop-ASE	36	24	36	16	69.4%	67%

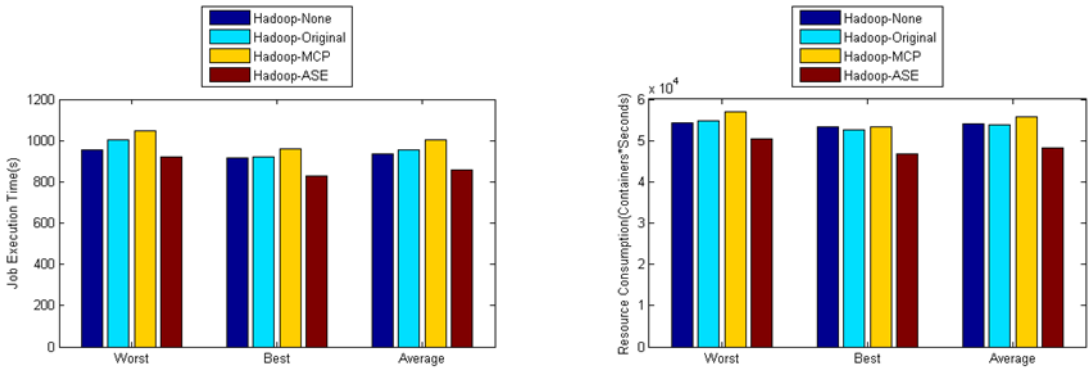


a) Job Execution Time b) Resource Consumption

Fig. 11. Sort in a multi-job heterogeneous environment

Through **Table 4**, we can find that Hadoop-Original and Hadoop-MCP launch much more backup tasks for map tasks, while Hadoop-ASE starts only about one-third of it. Moreover, none of the backed-up reduce tasks are launched by Hadoop-Original and Hadoop-MCP. By analyzing the reason, we discovered that these two strategies started the backup task only relying on historical finishing information. As there are no reduce tasks completed, the remaining time of the current task and running time of a backup task cannot be estimated. When enough tasks are completed, the remaining time of the current task and running time of a backup can be gotten. However, a backup task cannot complete faster than the current task. All of these determine that Hadoop-Original and Hadoop-MCP launched no backup tasks. Hadoop-ASE tries to get historical task information while estimating. If there is no information, Hadoop-ASE selects the slowest task that has lagged behind the average speed for 20%. This method helps us back up a reduce task, though no reduce tasks have finished.

Table 4 shows that MCP and Original Strategy have not backed up any reduce tasks. Actually, MCP and Original Strategy start backup tasks while there are some tasks that have finished. Because it is based on the average running time of a cluster, while there are no tasks(reduce tasks) have finished, backup tasks would not be launched though there are enough resources. Further, analyzing **Table 4**, we can find that the backup success rate of Hadoop-ASE is even little lower than Hadoop-Original, but Hadoop-ASE can execute job more quickly and consume fewer resources, which tells us that the backup moment selection plays a more important role rather than the backup success rate. Launching a backup task while there are no resources in the cluster would just cause some tasks to be delayed especially in a heterogeneous multi-job environment. The reason that Hadoop-MCP has a low backup success rate may be their method of estimating the remaining time of the current task. Therefore, the volume of the success rate is not the only key factor affecting the speed in a heterogeneous environment, but the moment when starting backing up the struggling tasks.



a) Job Execution Time b) Resource Consumption

Fig. 12. Grep in a multi-job heterogeneous environment

5. Conclusion

In this paper, we take different situations of resources into consideration. We also have implemented an adaptively speculative execution strategy in Hadoop-2.6.0, and we call it Hadoop-ASE. Experiment results show that our method can smartly backup tasks based on real-time resource. The performance of MRV2 is improved based on our strategy both on job execution time and resource consumption, no matter in a single-job environment or a multi-job

environment compared with other strategies. In the future, we would try to study cloud storage as shown in [27][28] which seems to be applied to optimize the mechanism of HDFS.

References

[1] Z. Fu, X. Sun, Q. Liu, L. Zhou and J. Shu, "Achieving Efficient Cloud Search Services: Multi-keyword Ranked Search over Encrypted Cloud Data Supporting Parallel Computing," *IEICE Transactions on Communications*, vol. E98B, no. 1, pp. 190-200, 2015. Article (CrossRef Link)

[2] Y. Kong, M. Zhang, and D. Ye, "A belief propagation-based method for task allocation in open and dynamic cloud environments," *Knowledge-Based ystems*, vol. 115, pp. 123 – 132, 2017. Article (CrossRef Link)

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010. Article (CrossRef Link)

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008. Article (CrossRef Link)

[5] K. Anyanwu, H. S. Kim, P. Ravindra, "Algebraic Optimization for Processing Graph Pattern Queries in the Cloud," *IEEE Internet Computing*, vol. 99, no. 2, pp. 52-61, 2013. Article (CrossRef Link)

[6] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *Proc. of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 1099-1110, 2008. Article (CrossRef Link)

[7] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu and R. Murthy, "Hive-a petabyte scale data warehouse using Hadoop," in *Proc. of 2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pp. 996-1005, 2010. Article (CrossRef Link)

[8] P. J. Tai and J. Yan, "Computing resource prediction for mapreduce applications using decision tree," *Web Technologies and Applications*, pp. 570-577, 2012. Article (CrossRef Link)

[9] W. Fang, B. He, Q. Luo and N. K. Govindaraju, "Mars: accelerating mapreduce with graphics processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 608-620, 2010. Article (CrossRef Link)

[10] Y. Zhang, Q. Gao, L. Gao and C. Wang, "Priter: a distributed framework for prioritizing iterative computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1884-1893, 2014. Article (CrossRef Link)

[11] B. Palanisamy, A. Singh and L. Liu, "Cost-effective resource provisioning for mapreduce in a cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1265-1279, 2015. Article (CrossRef Link)

[12] Y. Kwon, M. Balazinska, B. Howe and J. Rolia, "Skewtune: mitigating skew in mapreduce applications," in *Proc. of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 25-36, 2012. Article (CrossRef Link)

[13] B. Gufler, N. Augsten, A. Reiser and A. Kemper, "Handling data skew in MapReduce," in *Proc. of the 1st International Conference on Cloud Computing and Services Science (CLOSER)*, pp. 574-583, 2011. Article (CrossRef Link)

[14] Y. Fan, W. Wu, Y. Xu and H. Chen, "Improving MapReduce Performance by Balancing Skewed Loads," *Communications, China*, vol. 11, no. 8, pp. 85-108, 2014. Article (CrossRef Link)

[15] Q. Liu, W. Cai, J. Shen, Z. Fu, X. Liu, and N. Linge, "A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment," *Security and Communication Networks*, preprint, 2016. Article (CrossRef Link)

[16] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Proc. of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pp. 366-387, 2008. Article (CrossRef Link)

[17] S. Islam, J. Keung, K. Lee and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155-162, 2012. Article (CrossRef Link)
[18] A. Matsunaga and J. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proc. of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 495-504, 2010. Article (CrossRef Link)
[19] Y. Wang and W. Shi, "Budget-Driven Scheduling Algorithms for Batches of MapReduce Jobs in Heterogeneous Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 306-319, 2014. Article (CrossRef Link)
[20] W. Yu, Y. Wang, X. Que and C. Xu, "Virtual shuffling for efficient data movement in mapreduce," *IEEE Transactions on Computers*, vol. 6, no. 1, pp. 556-568, 2015. Article (CrossRef Link)
[21] S. Tang, B. S. Lee and B. He, "DynamicMR: A Dynamic Slot Allocation Optimization Framework for MapReduce Clusters," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 333-347, 2014. Article (CrossRef Link)
[22] M. Zaharia, A. Konwinski, A. Joseph, R. Katz and I. Stoica, "Improving Mapreduce Performance in Heterogeneous Environments," *OSDI*, vol. 8, no. 4, 2008. Article (CrossRef Link)
[23] C. Qi, C. Liu and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy, " *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 954-967, 2014. Article (CrossRef Link)
[24] Q. Liu, W. Cai, D. Jin, J. Shen, F. Zhang, X. Liu, and N. Linge, "Estimation Accuracy on Execution Time of Run-Time Tasks in a Heterogeneous Distributed Environment," *Sensors*, vol. 16, no. 9, pp. 1-15, 2016. Article (CrossRef Link)
[25] Q. Liu, W. Cai, J. Shen, X. Liu, and N. Linge, "An Adaptive Approach to Better Load Balancing in a Consumer-centric Cloud Environment," *IEEE Transaction on Consumer Electronics*, vol. 62, no. 3, pp. 243-250, 2016. Article (CrossRef Link)
[26] F. Ahmad, S. Chakradhar, A. Raghunathan and T. Vijaykumar, "Tarazu: optimizing MapReduce on heterogeneous clusters," *ACM SIGARCH Computer Architecture News*, pp. 61-74, 2012. Article (CrossRef Link)
[27] M. Dai, Z. Lu, D. Shen, H. Wang, B. Chen, X. Lin, S. Zhang, L. Zhang, and H. Liu, "Design of (4, 8) binary code with MDS and zigzag-decodable property," *Wireless Personal Communications*, vol. 89, no. 1, pp. 1-13, Jul. 2016. Article (CrossRef Link)
[28] M. Dai, C. W. Sung, H. Wang, X. Gong, and Z. Lu, "A new zigzag-decodable code with efficient repair in wireless distributed storage," *IEEE Transaction on Mobile Computing*, preprint, 2016. Article (CrossRef Link)

Qi Liu (M'11) received his BSc degree in Computer Science and Technology from Zhuzhou Institute of Technology, China in 2003, and his MSc and PhD in Data Telecommunications and Networks from the University of Salford, UK in 2006 and 2010. His research interests include context awareness, data communication in MANET and WSN, and smart grid. His recent research work focuses on intelligent agriculture and meteorological observation systems based on WSN.

Weidong Cai received his bachelor's degree in Software Engineering from Nanjing University of Information Science and Technology in 2014, and he is pursuing a master's degree in software engineering at the Nanjing University of Information Science and Technology. His research interests include Cloud Computing, Distributed Computing and Data Mining.

Qiang Liu is a lecturer of Hunan University of Technology. He is the information systems project management engineer and database system engineer. His research interest is majorly at the application of network technology and data mining.

Jian Shen received his bachelor's degree in Electronic Science and Technology Specialty from Nanjing University of Information, Science and Technology in 2007, and he received his masters and PhD in Information and communication from CHOSUN University, South Korean in 2009 and 2012. His research interests includes Computer network security, information security, mobile computing and network, wireless ad hoc network.

Zhangjie Fu received his BS in education technology from Xinyang Normal University, China, in 2006; received his MS in education technology from the College of Physics and Microelectronics Science, Hunan University, China, in 2008; obtained his PhD in computer science from the College of Computer, Hunan University, China, in 2012. Currently, he works as an assistant professor in College of Computer and Software, Nanjing University of Information Science and Technology, China. His research interests include cloud computing, digital forensics, network and information security.

Xiaodong Liu received his PhD in Computer Science from De Montfort University and joined Napier in 1999. He is a Reader and is currently leading the Software Systems research group in the IIDI, Edinburgh Napier University. He was the director of Centre for Information & Software Systems. He is an active researcher in software engineering with internationally excellent reputation and leading expertise in context-aware adaptive services, service evolution, mobile clouds, pervasive computing, software reuse, and green software engineering. He has meanwhile a successful track record of teaching in a number of software engineering courses which are widely informed by his research activities.

Nigel Linge received his BSc degree in Electronics from the University of Salford, UK in 1983, and his PhD in Computer Networks from the University of Salford, UK, in 1987. He was promoted to Professor of Telecommunications at the University of Salford, UK in 1997. His research interests include location based and context aware information systems, protocols, mobile systems and applications of networking technology in areas such as energy and building monitoring.