Computers and Electrical Engineering 50 (2016) 166–179

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Novel heuristic speculative execution strategies in heterogeneous distributed environments☆

Xin Huang[a], Longxin Zhang[a,b,*], Renfa Li[a], Lanjun Wan[a], Keqin Li[a,b,c]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[b] National Supercomputing Center in Changsha, Hunan, Changsha 410082, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 24 November 2014
Revised 7 June 2015
Accepted 8 June 2015
Available online 10 March 2016

Keywords:
Cloud computing
Dynamic loading
Hadoop
MapReduce
Speculative execution

ABSTRACT

MapReduce is a promising distributed computing platform for large-scale data processing applications. Hadoop MapReduce has been considered as one of the most extensively used open-source implementations of MapReduce frameworks for its flexible customization and convenient usage. Despite these advantages, a relatively slow running task called straggler task impedes job progress. In this study, two novel speculative strategies, namely, Estimate Remaining time Using Linear relationship model (ERUL) and extensional Maximum Cost Performance (exMCP), are developed to improve the estimation of the remaining time of a task. ERUL is a dynamic system load-aware strategy; using this strategy, we can overcome some drawbacks of the Longest Approximate Time to End (LATE) that misleads speculative execution in some cases. In exMCP, different slot values are considered. Extensive experiments show that ERUL and exMCP are applied to accurately estimate the remaining execution times of running tasks and reduce the running time of a job.

1. Introduction

Distributed computing platforms, such as MapReduce [1] and Dryad [2], have been considered as the mainstream computing platforms of data processing, data mining, web indexing, and e-business. Interconnected by commodity computers through networks, these platforms can implement tasks in parallel with high reliability and at low prices, and can also easily add or remove nodes. Hadoop is an open-source implementation of the MapReduce framework and is fully implemented with Java language, which provides program interfaces, such as C++, Python, Perl, and Shell. Large companies, Yahoo!, Aliyun, Facebook and so on replace expensive computers with Hadoop to conduct large-scale computing because Hadoop can be easily customized and used. Hadoop MapReduce includes computing nodes (called TaskTracker) and storage nodes (called DataNodes). In general, a computing node can function as a storage node at the same time. The data blocks of MapReduce are stored in a *Hadoop Distributed File System* (HDFS). The HDFS is a distributed file system designed to run on commodity hardware [3]. A MapReduce job is divided into multiple map tasks and reduce tasks by JobTracker. JobTracker then assigns these tasks to TaskTracker to execute. A map task processes a data block by using a user-customized mapper operator and delivers the corresponding output to reduce tasks. Reduce tasks fetch input data from

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. Hui Tian.

* Corresponding author at: College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China.
E-mail addresses: huangxin@hnu.edu.cn (X. Huang), longxinzhang@hnu.edu.cn (L. Zhang), lirenfa@hnu.edu.cn (R. Li), wancanjun2008@163.com (L. Wan), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.compeleceng.2015.06.013
0045-7906/

the map output through networks and process data by using a user-customized reducer operator. Since a computing node can also be a storage node, map tasks and data blocks on which share the same node, it is known as data locality. Apparently, data locality can reduce the execution time of map tasks. Map tasks and reduce tasks can be executed in parallel because the data of these two types of tasks are independent.

If a task of a job requires an abnormally long execution time, then the total completion time of the job is affected. Such a task is called a straggler. A speculative copy of this task (also called a backup task) is run on another faster node to ensure that this task is finished earlier than the original task [4]. This mechanism is called speculative execution. In heterogeneous distributed environments, computing nodes differ in terms of computing capability and network bandwidth. In addition, a specific job may cause bugs. These problems can cause stragglers. Thus, the completion time of a job is affected. As such, schedulers cannot acquire accurate execution information of nodes and tasks during task assignment. Consequently, the performance of a scheduling strategy is affected. As a fault-tolerant technology, speculative execution can correct the wrong decisions of schedulers to some extent, thereby improving computing efficiency.

An original implementation of speculative execution (as Hadoop-Naive) [5] is examined in Hadoop-0.20 to enhance performance. However, this strategy cannot work well in heterogeneous MapReduce systems. To schedule tasks in heterogeneous systems, Li [6] analyzed the performance of heuristic power allocation and scheduling algorithms of parallel tasks with constrained precedence. Zhang et al. [7] developed algorithms to enhance task reliability when a dynamic voltage scaling (DVS) technique is applied to achieve low power consumption in heterogeneous computing systems. Xu et al. [8] proposed a genetic algorithm to perform parallel task scheduling on heterogeneous distributed environments by utilizing multiple priority queues. Shen et al. [9] presented several methods to schedule necessary computations to trace the vasculature in retinal images. Tian et al. [10] explored an adaptive data collection strategy by using different communication radii for nodes distributed in different locations to balance the energy consumption in heterogeneous wireless sensor networks. In terms of task management in Hadoop systems, Longest Approximate Time to End (LATE) is proposed as a strategy (Hadoop-LATE) to adapt to heterogeneous environments [11,12]. The LATE strategy can yield some improvements but may cause misjudgment when stragglers are determined. The strategy fails when the misjudgement occurs. Some drawbacks, including inaccurate estimated time and system resource wastage, also exist.

Several other components involving MapReduce and HDFS, such as HBase [13], ZooKeeper [14], Pig [15], and Hive [16], constitute the Hadoop system. The nodes should process the tasks of MapReduce and the tasks of other components at the same time. This process results in an unstable system load of computing nodes, thereby affecting the remaining execution time. Hadoop-LATE and *Maximum Cost Performance* (MCP; as Hadoop-MCP), which was described in a previous study [17], does not consider the effect of system load when the remaining execution time is estimated. As a result, the estimated remaining time becomes inaccurate, thereby affecting the speculation effect. In addition, MCP does not consider the different values in the slots when the values of cluster resources are evaluated.

To solve these problems, we devise a heuristic speculative execution strategy called *Estimate Remaining time Using Linear relationship model* (Hadoop-ERUL). Based on heterogeneous computing systems, the system load used to estimate the remaining time of tasks is considered in Hadoop-ERUL. Hadoop-ERUL can overcome some deficiencies of Hadoop-LATE; the former is also more concise and efficient than Hadoop-MCP. Hadoop-ERUL can estimate the remaining time more accurately and detect stragglers more rapidly and more accurately. Moreover, backup tasks can run on suitable nodes with this strategy. Compared with Hadoop-LATE, Hadoop-ERUL can reduce the execution time of a job by 26%. Hence, we presented the paper entitled “A Heuristic Speculative Execution Strategy in Heterogeneous Distributed Environments” in the Proceedings of the Sixth International Symposium on Parallel Architectures, Algorithms, and Programming 2014 (PAAP-2014) [18]. In this work, a heuristic speculative execution strategy was implemented with ERUL on heterogeneous environments. Another novel strategy called extensional Maximum Cost Performance (exMCP) is devised in this work by fully using the values of different slots that were ignored in MCP.

In our further study, the contents affecting the efficiency of the algorithm are extended. In particular, the conference paper is significantly extended and composed of more than 30% new contents, including new algorithms, discussions, and solid experimental results that are not shown in the conference version. The three major contributions of this further study are listed as follows:

- We propose a heuristic speculative execution strategy with Hadoop-ERUL on heterogeneous environments.
- We implement Hadoop-exMCP to overcome the drawback of ignoring the different values of the slots in heterogeneous computing systems.
- We consider the system load to estimate the remaining time of tasks. This strategy can overcome some defects of Hadoop-LATE; this strategy is also more concise and efficient than Hadoop-MCP.
- We demonstrate that our proposed Hadoop-ERUL can be used not only to estimate the remaining time more precisely but also to detect stragglers more rapidly and more accurately, as revealed by the experimental results of a set of randomly generated task graphs and graphs of real-world problems with various characteristics.

The remainder of this paper is organized as follows: Several related works on speculative execution in the MapReduce framework and the relationship between system load and execution time of tasks are discussed in Section 2. Several drawbacks of previous studies are analyzed in Section 2.3. Our new strategies, ERUL and exMCP, are presented in detail

in Section 3. The performance of ERUL and exMCP is evaluated in Section 4. The conclusions and future works are presented in Section 5.

2. Related studies

In this section, related previous studies on speculative execution strategies and their drawbacks, along with the effect of system load on the execution time of a task, are discussed. At the end of this section, a novel motivation strategy to enhance the performance in Hadoop is presented.

2.1. Previous studies

In the original Hadoop-NAIVE speculation, the progress rates of tasks is calculated and these progress rates are compared with the average rate. If a large gap exists among the progress rates of tasks, then the slowest task is regarded as a straggler and backed up on a faster TaskTracker, which requires tasks to execute. A major drawback of this strategy is that it assumes that the nodes in the entire cluster system are homogeneous; the data to be processed by map tasks are uniform. In fact, different nodes exhibit distinct computing capabilities in realistic MapReduce systems. Moreover, the data processing capability of the same node is even different over time. Accordingly, the strategy may fail during execution of the tasks.

To solve this issue, Zaharia et al. [11] developed the Hadoop-LATE strategy. In Hadoop-LATE, $specPriority = \frac{1-progress}{progressRate}$ is used as the speculative execution priority of a task. A standard deviation of progress rate is utilized as the threshold value to determine stragglers. The top 25% nodes in terms of computing capability are selected as candidate nodes to execute backup tasks, and the number of the tasks undergoing speculative execution is limited to save cluster resources. Hadoop-ERUL detects stragglers more accurately than Hadoop-NAIVE. Backup tasks are allocated to the nodes with strong processing capability; as a consequence, these tasks become more suitable to heterogeneous environments. Nevertheless, there are still several problems in LATE, which are discussed in Section 2.3.

The Hadoop-MCP strategy determines the remaining execution time of the task by estimating the remaining time in different stages. The MCP model is used to ensure that backup tasks finish executing ahead of stragglers. Furthermore, implementing speculative execution on a task can benefit the cluster. This model can be implemented based on a previous study [17]. The execution load of the cluster is reduced to a significant extent with this type of speculative execution strategy. However, the implementation of the model is complex and its staged estimation may result in a large error.

2.2. Effect of system load on the execution time of a task

System load is defined as the ratio of the summation of the total number of the tasks to be processed and the tasks being processed over the number of the tasks that can be processed at the same time:

$$R_{systemload} = \frac{T+P}{C}, \tag{1}$$

where T is the total number of the tasks to be processed, P is the number of the tasks being processed, and C is the number of the tasks that can be processed.

Each node is running MapReduce tasks because of a large amount of processes running on nodes. Contention for computing resources among the nodes naturally exists. With more processes running, more tasks are handled, thereby affecting the execution time of running tasks. The relationship between the measured load during execution and the running time is almost perfectly linear ($R^2 > 0.99$), which was proposed in a previous study [19]. This result is useful and can help us consider system loads in estimating the remaining time of a running task.

2.3. Drawbacks in previous studies

The LATE strategy exhibits many drawbacks, as discussed in a previous study [17]. The LATE strategy fails when data skew occurs in map tasks. The progress score of the Hadoop map is calculated using the total processed $\frac{Data_{total}}{Data_{processed}}$. As such, the priority formula is transformed into the following expression:

$$specPriority = (execution\_time) \times \frac{Data_{total}}{Data_{processed} - 1}. \tag{2}$$

Small total data correspond to low priority. However, this phenomenon is not due to the processing capability of nodes. As such, the task is not a real slow task. Thus, misjudgment occurs. As reported in previous studies [20,21], map tasks can be divided into three classes based on the distribution of tasks and data. In particular, data and tasks on the same node can be classified as *Data-Local* type, data and tasks on the same rack can be classified as *Rack-Local* type, and data and tasks on different racks in the same data center can be classified as *Off-Rack* type.

The execution of tasks is distinct among the three classes because of the difference among their data transmission rates. The LATE strategy does not consider this difference. Thus, misjudgment easily occurs when priority and fast or slow nodes

are determined. Heavy system loads lead to low response rates and execution rates, resulting in a long execution time. The Hadoop-LATE and Hadoop-MCP strategies do not consider the system load when the remaining execution time is speculated. As a result, task speculation fails, thereby affecting the determination of slow and fast tasks.

The Hadoop-ERUL strategy is proposed to overcome the aforementioned problems and to yield improvements in this work; the points of these improvements can be described as follows:

- Map tasks are classified into three types based on the distribution of map data. As such, obtaining the statistics and analyzing the execution information of tasks become easier and more efficient.
- A solution is provided to determine priority when data skew occurs in map tasks.
- A more accurate solution is provided to determine reduced priority.
- The MCP model is used to guarantee the benefits of speculative execution.
- The fixed threshold is replaced with a dynamic average threshold to determine slow and fast nodes with more accuracy.
- The system load is considered when tasks are executed.

3. Model and algorithm

The model and algorithms of this study are presented in detail in this section.

3.1. Task classification

As discussed in Section 2.3, map tasks can be classified into three types, namely, *Data-Local*, *Rack-Local*, and *Off-Rack*. This classification can facilitate the priority ranking of the tasks of one type and prevent incorrect priority queues caused by differences in the transmission rates, which will lead to speculation failures. The capability of nodes to process map tasks is also classified into three types similarly, namely, Data-Local, Rack-Local, and Off-Rack. With regard to reduce tasks, their input data are assumed to be evenly distributed on data nodes.

3.2. Speculative execution priority strategy

In this sub-section, a new speculative execution priority strategy is proposed. Data skew and dynamic system loads are considered when priority ranking is performed. Therefore, the priority ranking is more accurate and reasonable.

3.2.1. Handling input data skew of map tasks

In Section 2.3, the effect of input data skew on the execution time of map tasks was analyzed. Based on Eq. (2), $priority = (execution\_time) \times \frac{Data_{rem}}{Data_{processed}}$. As such, $\frac{priority}{Data_{rem}}$ can be used as the priority. This method can weaken the effect of $Data_{rem}$ on the determination of fast and slow tasks.

3.2.2. Estimation of the remaining time using the linear relationship model

As reported in a previous study [19], the relationship between system loads and execution time of tasks ($R^2 > 0.99$) is linear. Based on extensive statistical analyses, the following relationship can be obtained when the load is regarded as a continuous signal:

$$\frac{t_{exec}}{1 + \frac{1}{t_{exec}} \int_{t_{start}}^{t_{start}+t_{exec}} z(t)dt} = t_{nom}, \tag{3}$$

where t_{exec} is the execution time of the task under the load $z(t)$, t_{nom} is the running time of the task on a complete unloaded machine, and t_{start} is the start time of the execution. However, the system load cannot be expressed with an exact continuous function and the obtained history load information is discrete. The phase in which a task remains is called the current phase. The remaining time of the current phase is calculated using the remaining data and the bandwidth size, as shown in (4):

$$t_{rem} = t_{rem,cp} + t_{rem,np} = \frac{data_{rem,cp}}{bandwidth_{cp}} + \sum_{p \in cp} est\_time_p * factor_d, \tag{4}$$

where $t_{rem,cp}$ is the remaining time in the current phase, $t_{rem,np}$ is the resting time in the next phase, $data_{rem,cp}$ is the size of the remaining data, $bandwidth_{cp}$ is the size of the bandwidth in the current phase, and $factor_d$ is a constant factor.

In this work, *delta* of 5 s is used as the period to determine the load value. A series of load values are obtained as $\langle Z \rangle = \cdots z_{t-1}, z_t, z_{t+1}, \cdots (t \in Z^*)$. As such, Eq. (3) can be rewritten as follows:

$$\frac{t_{exec}}{1 + \overline{Z}} = t_{nom}, \tag{5}$$

$$\overline{Z} = \sum_{t_{now}}^{t_{now}+t_{exec}} \frac{z_t}{t_{exec}}. \tag{6}$$

A relationship exists between the remaining times of the tasks in two different load series $\langle Z_1 \rangle$ and $\langle Z_2 \rangle$, which can be formulated as follows:

$$\frac{tRemain_1}{1+\overline{Z_1}} = t_{nom} = \frac{tRemain_2}{1+\overline{Z_2}}, \tag{7}$$

where Z_1 and Z_2 are the average loads of the two load series in the remaining times $tRemain_1$ and $tRemain_2$, respectively.

We assume that a particular task is characterized by a load series of $\langle Z_t \rangle = z_0, z_{1delta}, z_{2delta}, \ldots, z_{now-delta}, z_{now}$, where $now = m \times delta(m \in Z^*)$ at time t and the progress of this task is expressed as $progress_{now}$. When z_t varies from 0 to t_{now}, the average system load can be calculated and expressed as Eq. (8):

$$\overline{Z_{now}} = \sum_{0}^{now} \frac{z_t}{n}, \tag{8}$$

where n represents the number of elements in $\langle Z_t \rangle$. If the system load is maintained as $\overline{Z_{now}}$, then the remaining execution time is denoted as $tRemain_{now}$, which can be computed as follows:

$$tRemain_{now} = (1 - progress_{now}) \times \frac{t_{now}}{progress_{now}}. \tag{9}$$

As the system load continuously change and shows uncertainty, the average load Z_{est} in the remaining time can be estimated using the following formula:

$$\overline{Z_{est}} = \frac{\sum_{now-k\times delta}^{now} + \overline{Z_{now}}}{k+1}, \tag{10}$$

where k is set at 5 in this work. In future research, a more effective method should be developed to estimate the average load in the remaining time.

Hence, a more accurate method to estimate the remaining execution time $tRemain_{est}$ can be written as follows:

$$tRemian_{est} = \frac{(1+\overline{Z_{ext}})tRemain_{now}}{1+\overline{Z_{now}}}. \tag{11}$$

This model is the ERUL, which benefits the estimation of the remaining execution time. Furthermore, the ERUL works more efficiently.

3.2.3. Speculative execution priority strategy

In map tasks, the mapper function of the user is called to process the data block. The map speculative execution priority can be calculated and expressed as Eq. (12):

$$mapSpecPriority = \frac{tRemain_{map}}{Data_{rem}}, \tag{12}$$

where $tRemain_{map}$ denotes the remaining time estimated by ERUL.

The completion rate of fetching data from the map is considered in our scheme to highlight the effect of the copy phase on the execution time of reduce tasks. The reduce speculative execution priority is estimated as follows:

$$reduceSpecPriority = tRemain_{reduce} \times \frac{(Map_{total}+1)\times\alpha}{(Map_{fetched}+1)+\beta}, \tag{13}$$

where $tRemain_{reduce}$ is the remaining time estimated by ERUL, Map_{total} is the total number of map tasks, and $Map_{fetched}$ is the number of the map tasks from which the reduce task has finished fetching data.

Considering that no input data are obtained from a map task, we assume that the reduce task has finished data fetching. α is set at 1/3, β is set at 2/3, and $\alpha + \beta = 1.0$. In Eq. (13), the effect of the data fetching rate is considered in the speculative execution priority of the reduce stage in the copy phase. As $Map_{fetched}$ increases gradually, *reduceSpecPriority* decreases accordingly. The copy phase continues until the condition $Map_{fetched} = Map_{total}$ is satisfied. As such, the speculative priority of the reduce task may be regarded as only having a relationship with the computing capability of nodes. The calculation process is the same as that of map task.

3.3. Selection of candidate nodes

In the presence of each type of tasks, the processing capabilities of nodes are different. When a task of the same type is completed on the node, we let *preCap* be the previous processing capability of the node for this type. *exeTime* represents the

execution time of a task, which is executed on a node until the task is successfully completed. The processing *capability* can be modeled as follows:

$$capability = \begin{cases} \alpha \times preCap + (1-\alpha) \times \dfrac{exeTime}{Data_{total}}, & for\ map\ tasks \\ \alpha \times preCap + (1-\alpha) \times exeTime, & for\ reduce\ tasks, \end{cases} \tag{14}$$

where α represents the smoothing factor and its value can be set at 0.3.

If none of the tasks in the same type is completed on the node, then the processing capability is expressed as follows:

$$capability = \begin{cases} \dfrac{(time_{cost} + tRemain)}{Data_{total}}, & for\ map\ tasks \\ time_{cost} + tRemain, & for\ reduce\ tasks, \end{cases} \tag{15}$$

where $time_{cost}$ is the time spent by the first running task in the type and *tRemain* is the remaining time of the task estimated by ERUL.

With the processing capability of the node of each type of tasks, fast processing nodes and slow processing nodes can be more easily identified. If a work node has a slow processing capability with certain type of tasks, then no backup tasks for such a type are allowed or executed on the slow node.

3.4. Ensuring effectiveness of speculative execution

The task with the highest priority is not always worth being speculated. A case may arise when the backup task finishes later than the original task. Consequently, speculative execution no longer reduces the execution time of the job; instead, more resources and bandwidths of the working nodes are consumed extravagantly.

The MCP model is utilized to guarantee the effectiveness of speculative execution. The following condition should be satisfied:

$$\frac{rem\_time}{backup\_time} > \frac{(1+2\eta)}{1+\eta}, \eta \in (0, +\infty). \tag{16}$$

If we let η be the *load_factor* of the Hadoop cluster, then we obtain the following expression:

$$\eta = load\_factor = \frac{numpending\_tasks}{numfree\_slots}, \tag{17}$$

where *numpending_tasks* represents the number of pending tasks and *numfree_slots* denotes the number of the free slots in the cluster.

In this work, *rem_time* is the remaining time of the task estimated by ERUL and *backup_time* is computed on the basis of the following expression:

$$backup\_time = \begin{cases} procCap \times Data_{total}, & for\ map\ tasks \\ procCap, & for\ reduce\ tasks, \end{cases} \tag{18}$$

where *procCap* denotes the processing capability of the candidate node for this type of tasks.

3.5. Hadoop-ERUL speculative execution strategy

When the JobTracker receives a heartbeat from the TaskTracker, the Hadoop-ERUL speculative execution strategy is applied as follows:

- Select the task with the highest priority as a candidate based on the order of non-rack map, non-locality map, locality map, and reduce task types.
- Determine the type of the task on the TaskTracker and its capability to process this type of tasks.
- Determine whether the TaskTracker is a fast node; otherwise, go back to the first step.
- Determine whether the task can do speculative execution on this node; otherwise, go back to the first step.
- Backup the candidate task on the TaskTracker.

A detailed description of the Hadoop-ERUL strategy is provided in Algorithm 1.

Algorithm 1. Hadoop-ERUL

Require: A task object.
Ensure: A remaining time *tRemain*.

```
1: startTime ← getSystemCurrentTime();
2: currentTime ← startTime;
3: preTime ← startTime;
4: loadSum ← getSystemLoad();
5: loadCount ← 1;
6: Initial all array lastLoadSerial[] elements by loadSum;
7: while isRunning do
8:   preTime ← currentTime;
9:   currentTime ← getSystemCurrentTime();
10:  if currentTime - preTime ≥ δ then
11:    currentLa ← getSystemLoad();
12:    loadSum ← loadSum + currentLa;
13:    loadCount ← loadCount + 1;
14:    avgLa ← loadSum/loadCount;
15:    for i← 1 to lastLoadSerial.length − 1 do
16:      lastLoadSerial_{i-1} ← lastLoadSerial_i;
17:    end for
18:    lastLoadSerial_{length-1} ← currentLa;
19:    lastLoad ← (Σ_0^{length} lastLoadSerial_i + avgLa) / (lastLoadSerial.length + 1);
20:    tRemain ← ((1 − task.progress) × task.executTime) / task.progress;
21:    tRemain ← ((1 + lastLoad) × tRemain) / (1 + avgLa);
22:  else
23:    sleep(δ/2);
24:  end if
25: end while
26: return a remaining time tRemain.
```

3.6. Hadoop-exMCP speculative execution strategy

Differences are among the capabilities of the slots in distinct nodes. The value of each slot should be considered when the cost and profit of the slot are calculated. Based on a previous study [17], such a conclusion is obtained as follows:

$$profit_{backup} > profit_{not\_backup} \Leftrightarrow rem\_time > t_B + \frac{\eta}{1+\eta} \times t_A, \tag{19}$$

where η is the load factor in a Hadoop cluster, as formulated in Eq. (17).

To improve the performance of the speculative strategy MCP, we develop Hadoop-exMCP strategy. The processes of Hadoop-exMCP are as follows:

The information of each task in a job is stored in a JobTracker, which includes the progress of the allocated tasks. The TaskTracker sends its heartbeat, which includes information on the running task and node status, to the JobTracker. The received information is combined with the type of the task, and the JobTracker updates the remaining execution time of these nodes. Based on the reported information, the computation capability of the node for executing the corresponding tasks is calculated. The threshold for determining the fast and slow nodes is obtained. The fast node, not the slow node, is assigned to implement backup tasks.

When a node executes a task, each task is filtered with the Hadoop-exMCP strategy to determine whether it satisfies the following requirements of the backup task:

- The tasks are filtered in accordance with Off-Rack map tasks, Rack-Local map tasks, Data-Local map tasks, and reduce tasks. The reduce task cannot enter the reduce phase until the map task is completed. Therefore, the priority of a map task is higher than that of a reduce task. In addition, the execution time decreases in the order of Off-Rack map tasks, Rack-Local map tasks, and Data-Local map tasks. As such, a high priority corresponds to a long execution time.
- A sequential task is selected to ensure that this task has been executed for a period of time (such as speculativeLag = 30 s). This point guarantees that a task is more accurate when the corresponding Exponentially Weighted Moving Average (EWMA) speed is evaluated. The task remains stable after a stretch is run. The remaining time t_{rem} of the task can be calculated on the basis of Eq. (4).

- The type of the task, which is executed in the node, is verified. The time t_{backup} required for executing this type of tasks in the requested node is then evaluated. The required time $t_{original}$ for the selected task running in the original node is also calculated.
- t_{rem}, t_{backup}, and $t_{original}$ of the selected task are substituted in Eq. (19). Then, whether this task should be run on the requested node is determined by applying the exMCP strategy.

4. Experiments and evaluation

Several important performance metrics are introduced in this section to evaluate the proposed strategies. Hadoop-NONE, Hadoop-NAIVE, and Hadoop-LATE are compared with Hadoop-ERUL. Hadoop-NONE, Hadoop-NAIVE, and Hadoop-LATE are also compared with Hadoop-exMCP.

4.1. Performance metrics

4.1.1. Makespan

Makespan, or scheduling length, is defined as the completion time of the latest task in a job. For most scheduling algorithms in a Hadoop cluster, makespan is one of the main metrics used to evaluate performance.

4.1.2. Throughput

The throughput of jobs in a Hadoop cluster is the amount of jobs that the cluster can handle per unit time, that is, the number of jobs that correspond to the cluster.

4.2. Results of the evaluation of the ERUL model of the remaining time and the analysis

We acquire the progress rate of the task and the remaining time estimated by the LATE and ERUL strategies at time intervals of 5 s. The real remaining time is acquired once the task is completed. The relative error between the estimated time and the real time can be calculated as follows:

$$RelativeError = \frac{|estim\_time - real\_time|}{real\_time}. \tag{20}$$

Fig. 1 shows the comparison of the evaluation effects between ERUL and LATE. The images on the left show the load changes in the system. The images on the right show the relative error of the evaluation. Fig. 1(a) shows the system load in a basically stable situation, whereas Fig. 1(b) shows the system load in a dynamically changing situation. In the basically stable situation, ERUL and LATE with a relative error of less than 0.05 can estimate the remaining time. In the dynamically changing situation, ERUL is more efficient than LATE because the former reduces the relative error to 0.25 and consumes 58% less time than the latter.

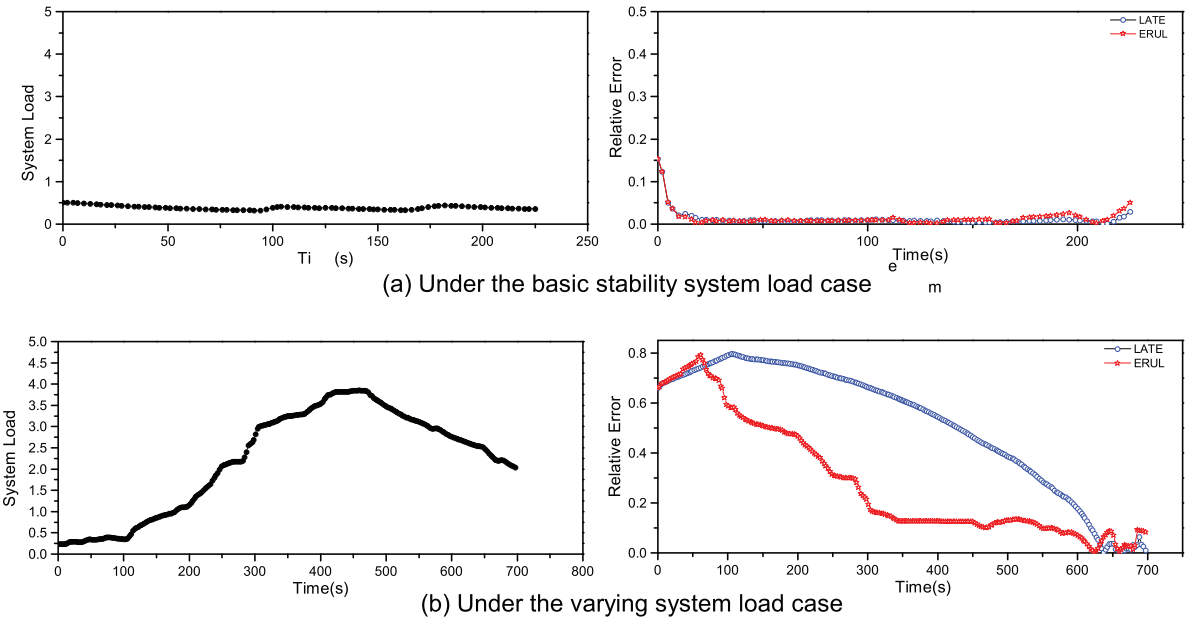

Fig. 1. Evaluation effects of ERUL and LATE.

4.3. Hadoop-ERUL strategy experiments in heterogeneous environments

Clusters in heterogeneous environments include seven physical nodes; each node contains a dual processor (2.2 GHz AMD Opteron™ Processor 4122 with four physical cores), 8 GB of RAM, and 500 GB disk. Three of these physical nodes run on virtual machine nodes in VirtualBox. The three virtual nodes are distributed on two physical nodes; each virtual machine is equipped with a dual core processor, 2 GB of RAM, and 100 GB disk. We use the default configuration of the HDFS. In heterogeneous environments, five map slots and five reduce slots are assigned to each physical node TaskTracker; two map slots and two reduce slots are assigned to each virtual node. Sort, Grep, and WordCount in Hadoop are used as primary workloads. Furthermore, 5 GB data randomly generated by RandomWriter are utilized as the input of Sort Job. A 4 GB text file generated by words and random character strings with lengths not longer than 12 is used as input of Grep and WordCount. In addition, "hello" is utilized as the statistical string of Grep. Three strategies, namely, Hadoop-NONE, Hadoop-NAIVE, and Hadoop-LATE, are compared with Hadoop-ERUL. The performance of the Hadoop-ERUL strategy is evaluated on the basis of the number and rate of execution successes of the speculative tasks and the completion time of the job. Each workload is tested six times. The performances of the strategies are compared in terms of the average, the most efficient, and the least efficient cases to prevent the changing environments from affecting the results.

The Sort workload generates 140 map tasks and 36 reduce tasks. Table 1 shows the maximum and minimum execution times of the tasks running with different strategies. The comparison results show that Hadoop-ERUL affects the cluster performance to a less extent than other strategies. Hadoop-ERUL can effectively reduce the maximum execution time of map and reduce tasks.

The performance of these strategies is determined by calculating the number of the speculative execution tasks and by counting the success rate of the tasks under the speculative strategies. Table 2 shows the Sort workload speculation effects when different strategies are used. This result is obtained from the summation of the number of backup tasks with the jobs running six times in each strategy. The comparison results also show that the number of the speculations of the reduce task in Hadoop-ERUL decreases significantly, and the success rate of reduce tasks improves significantly. However, misjudgment still occurs in Hadoop-ERUL. The main reason for misjudgment is the inaccurate estimation of the progress of reduce tasks, which should be improved. As such, ERUL can be used not only to control the number of tasks to be speculated effectively but also to detect and backup the stragglers accurately for execution on suitable nodes. Therefore, the response time of the job can be shorter than that of other strategies.

The purpose of all types of optimization strategies for scheduling is to reduce the execution time of the job. Fig. 2 shows the comparison of the execution times of sorting workloads with different strategies. Notably, the Hadoop-ERUL strategy results in 16% more improvements than Hadoop-NONE and 14% more improvements than Hadoop-LATE.

The number of reduce tasks is not limited when speculation is performed using the Hadoop-NAIVE strategy. Hadoop-NAIVE consumes more time than Hadoop-NONE. In addition, the backup tasks in Hadoop-NAIVE do not select nodes with a high processing speed to execute. Therefore, many backup tasks are executed not as rapidly as the original tasks.

Misjudgment and false negatives inevitably exist in Hadoop-LATE because the method by which square deviation is used to determine whether a task is a straggler exhibits certain defects. However, Hadoop-LATE performs more efficiently than Hadoop-NAIVE because the former has a limited number of tasks to be speculated compared with the latter. Hadoop-ERUL considers the effect of system loads on execution time; thus, Hadoop-ERUL can estimate the remaining time of tasks more accurately to determine stragglers. In this strategy, nodes are classified more accurately as fast and slow nodes based on the time consumed on processing different types of tasks.

Pressure tests are conducted on two machines by increasing the system load and running Grep workloads. Fig. 3 shows the Grep workload running under dynamic system loads. The performance of Hadoop-ERUL is improved by 33%, which is 26% more than that of Hadoop-NONE and Hadoop-LATE. Accordingly, Hadoop-ERUL can yield better speculation results under variable load and produce notable improvements on the execution time of a job than Hadoop-LATE. Experiments are conducted on the data skew of map tasks in the WordCount workload. The input data blocks are divided into two groups: one group has a size of 128 MB and the other group has a size of 64 MB. The corresponding performance under the Hadoop-ERUL strategy is compared with that of the Hadoop-NONE and Hadoop-LATE strategies.

Fig. 4 shows the comparison of the execution time of a job when data skew occurs in map tasks. Hadoop-ERUL consumes approximately 11% less execution time than Hadoop-NONE and 7% less execution time than Hadoop-LATE. The success rate of the speculation of the map tasks in Hadoop-LATE is 58%; by contrast, The success rate of the speculation of the map tasks in Hadoop-ERUL is up to 97%.

Table 1
Comparisons of Sort workload's executive time under different strategies.

Strategy	Max map execution time (s)	Min map execution time (s)	Max reduce execution time (s)	Min reduce execution time (s)
Hadoop-NONE	238	3	661	174
Hadoop-NAVIE	48	3	735	178
Hadoop-LATE	49	3	638	173
Hadoop-ERUL	46	3	498	171

Table 2
Speculation effects of Sort workload with different strategies.

Strategy	Sum of backup		Sum of success		Backup success rate	
	Map	Reduce	Map	Reduce	Map (%)	Reduce (%)
Hadoop-NAIVE	10	71	9	28	90	39.4
Hadoop-LATE	14	43	11	21	78.6	48.8
Hadoop-ERUL	12	34	12	33	100	97.1

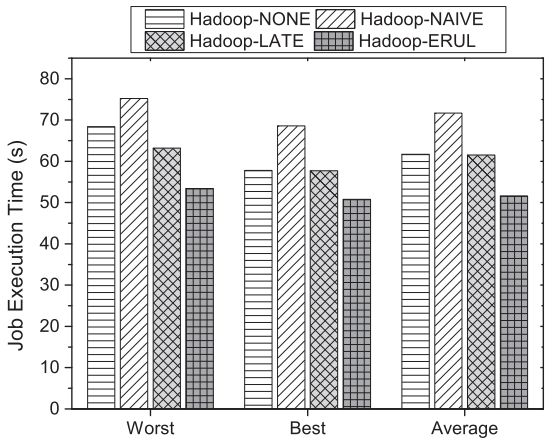

Fig. 2. Comparisons of job's execution time of Sort workload under different strategies.

These results show that Hadoop-ERUL is more accurate than the other strategies in determining stragglers when data skew occurs in map tasks. However, several failure cases still occur when the input of reduce tasks is assumed as even, which will be investigated in our future work.

Cridmix2 is utilized to compute the throughput of a cluster under different speculative strategies. Fig. 5 shows that Hadoop-ERUL enhances the throughput by 10% and 17% more than that of Hadoop-LATE and Hadoop-NAIVE, respectively. As Hadoop-ERUL effectively reduces the makespan of each job, the execution efficiency is significantly improved and the throughput of the cluster is increased.

4.4. Experiments of the Hadoop-exMCP strategy in heterogeneous environments

The performance of the Hadoop-exMCP strategy in heterogeneous systems is evaluated in this section. The makespan and the throughput of the jobs in a cluster are the main metrics to assess Hadoop-NONE, Hadoop-LATE, and Hadoop-MCP.

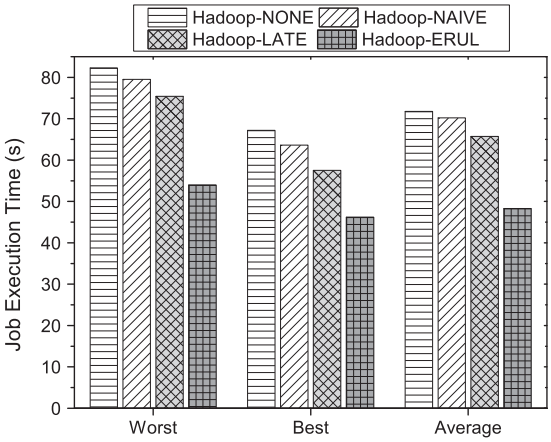

Fig. 3. Comparisons of Grep Workload under dynamic system loads.

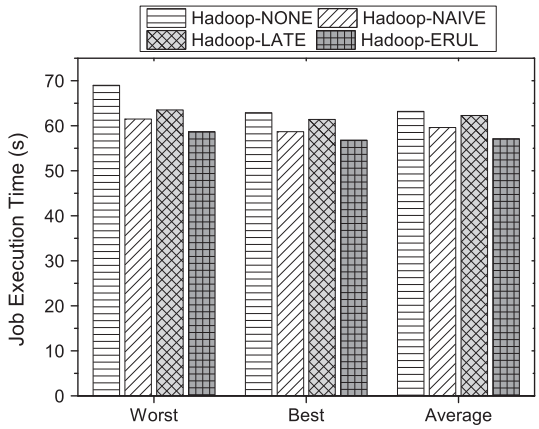

Fig. 4. Comparisons of WordCount Workload with data skew in map tasks.

The cluster used in our experiments runs with Ubuntu Server 12.04 (64 bit OS). The version of Hadoop is configured with Hadoop-1.2.0. Three different types of computers with diverse calculating capabilities are used to establish a heterogeneous computer system. The detailed configuration is shown in Table 3. Therefore, this Hadoop cluster contains 60 map slots and 60 reduce slots. We keep the default configuration for the HDFS and use the FairScheduler as the task scheduler.

The jobs used for the test consist of Sort, WordCount, Grep, and Gridmix2. Sort executes the function of sorting a large amount of data and outputting the ordered results. Sort is classified as computational and I/O-intensive operations. Grep is used to count the times that the specified word or regular expression appears in the text file. The word that satisfies the condition and the number of appearances is outputted in the HDFS. The input data of Sort jobs is generated randomly by the RandomWriter job. The size of the random data for the test is approximately 5 GB. This 5 GB text file comprises the string with a length of less than 12. This text file is also the input job of WordCount and Grep. Furthermore, this text file starts with “helloast”, which is the regular expression used to count Grep jobs. Gridmix2 includes three small-scale jobs and three large-scale jobs. The small job is set with 10 reduce tasks, whereas the large job is set with 70 reduce tasks. Each of the following comparisons is the result of the jobs running six times under different strategies. The least efficient, most efficient, and average values of the three cases are utilized to prevent the effect of various environments on the text results.

The performance of the strategies is evaluated by speculating the number of execution tasks and the success rate of tasks. Reducing the makespan of the jobs in a Hadoop cluster is one of the major metrics of optimization strategies. The makespan is considered as the standard to evaluate different strategies when diverse jobs are scheduled. One of the important reasons for a long makespan is the misjudgment of slow tasks. Fig. 6 shows the comparison of the makespan of Sort jobs running with diverse speculative strategies. Hadoop-exMCP surpasses Hadoop-MCP and Hadoop-LATE with makespan reductions of 13% and 26%, respectively.

With respect to the WordCount jobs, 81 map tasks are generated and 30 reduce tasks are set. Fig. 7 shows the performance of the four strategies. Hadoop-MCP can reduce the makespan by 6% and 12% of Hadoop-MCP and Hadoop-LATE, respectively.

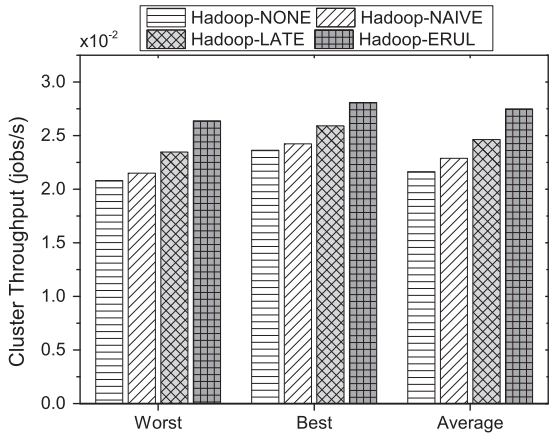

Fig. 5. The throughput of jobs in a cluster under different strategies.

Table 3
Parameter configuration.

Node styles	CPU	Memory (GB)	Disk (GB)	Numbers	Slots assignment	Use
Sugon server	2.2 GHz, 8 cores	8	500	5	Map: 7 Reduce:7	1 master node 4 storage and computer node
Dell workstation	3.07 GHz, 4 cores	4	500	4	Map: 5 Reduce: 5	storage and computer node
PC	3.2 GHz, 2 cores	2	500	4	Map: 5 Reduce: 5	storage and computer node

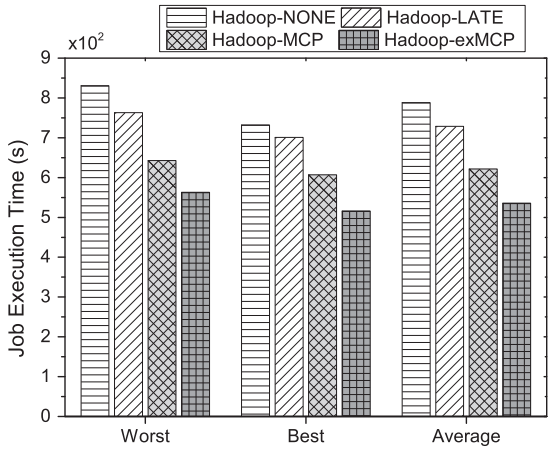

Fig. 6. Comparisons of Sort jobs in a cluster under different strategies.

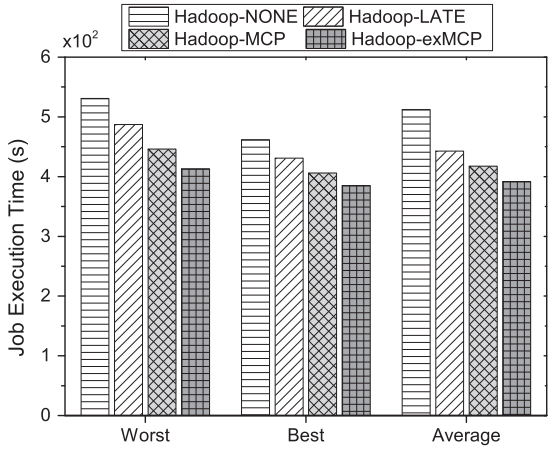

Fig. 7. Comparisons of WordCount jobs under different strategies.

For the Grep jobs, approximately 81 map tasks are generated and 16 reduce tasks are set. Two sub-jobs are included. One sub-job is used to search and account for Grep-search jobs. The other sub-job is used to sort and account for Grep-search jobs. As such, the makespan of Grep jobs is the summation of the execution time of the two sub-jobs. As shown in Fig. 8, the makespan of Grep jobs with Hadoop-exMCP is decreased by 8% and 21% compared with Hadoop-MCP and Hadoop-LATE, respectively.

Fig. 9 shows the comparison of the throughput of Gridmix2 jobs in a Hadoop cluster under different strategies. Hadoop-exMCP improves the throughput of Gridmix2 jobs by 11% and 18% against Hadoop-MCP and Hadoop-LATE, respectively. As the value of the slot is considered in Hadoop-exMCP, the values of the original node and the selected node as a backup are considered when the backup tasks are processed. The throughput is then improved.

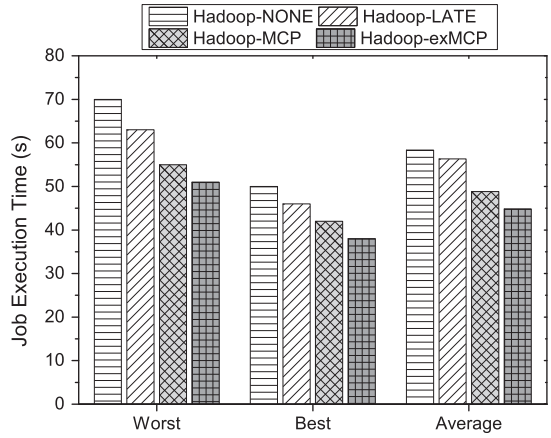

Fig. 8. Comparisons of Grep jobs under different strategies.

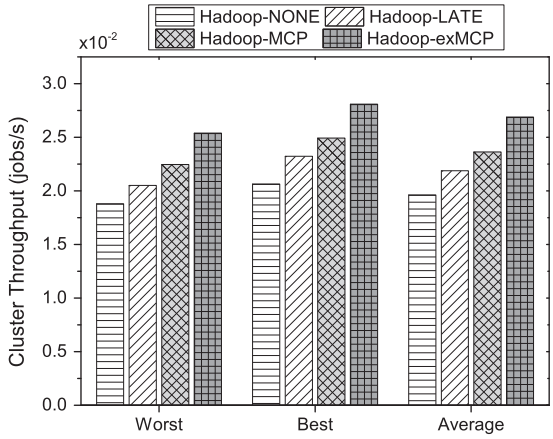

Fig. 9. The throughput of Gridmix2 jobs in a Hadoop cluster under different strategies.

The proposed Hadoop-ERUL and Hadoop-exMCP strategies can be implemented easily in *Yet Another Resource Negotiation* (YARN) [22], which is the latest MapReduce framework. Based on our experimental results, our conclusion is that Hadoop-exMCP significantly surpasses Hadoop-MCP and Hadoop-LATE in terms of the makespan and throughput of jobs.

5. Conclusion

The impact of system loads is important while making the speculative strategy in a MapReduce system. Straggler tasks are speculatively backed up on other nodes to execute in some speculative execution strategies in order to reduce the completion time of a job. Considering the linear relationship between the system load and execution time of tasks, we presented the ERUL model. The model functions more accurately in estimating the remaining running time of reduce tasks than LATE. Based on this model, a heuristic speculative execution strategy called Hadoop-ERUL is established. Hadoop-ERUL is devised to provide some improvements to address the existing problems in LATE, including data skew in map tasks and poor effects of speculative execution. Considering the different values of slots and making full use of them, we proposed the Hadoop-exMCP strategy. Both Hadoop-ERUL and Hadoop-exMCP are implemented on Hadoop-1.2.0 and can be implemented easily on Apache Hadoop NextGen MapReduce, such as *YARN*. Extensive experiments show that the running time of the job can be reduced by 26% and 19% with Hadoop-ERUL and Hadoop-exMCP, respectively. The response time of the job is also decreased effectively.

Scheduling optimization in heterogeneous distributed environments, such as Hadoop, is in the initial development stage. Further studies should be conducted to develop a more robust and accurate speculative execution strategy to estimate the remaining time of a running task and to address the problems caused by the input data skew of reduce tasks.

Acknowledgements

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005 and 61432005), and the National Natural Science Foundation of China (Grant Nos. 61370095 and 61472124), and Graduate Innovative Fund of Hunan Province (Grant No. CX2011B127).

References

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM 2008;51(1):107–13.
[2] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: distributed data-parallel programs from sequential building blocks. SIGOPS Oper Syst Rev 2007;41(3).
[3] Hadoop hdfs architecture guide. <http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html> [accessed on 12.11.14].
[4] Sakr S, Liu A, Fayoumi AG. The family of MapReduce and large-scale data processing systems. ACM Comput Surv 2013;46(1):1–44.
[5] Schwerz A, Liberato R, Wiese I, Steinmacher I, Gerosa M, Ferreira J. Prediction of developer participation in issues of open source projects. In: 2012 Brazilian Symposium on Collaborative Systems (SBSC); 2012. p. 109–14.
[6] Li K. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. IEEE Trans Comput 2012;61(12):1668–81.
[7] Zhang L, Li K, Xu Y, Mei J, Zhang F, Li K. Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster. Inf Sci 2015 <http://www.sciencedirect.com/science/article/pii/S0020025515001231> . doi:10.1016/j.ins.2015.02.023.
[8] Xu Y, Li K, Hu J, Li K. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Inf Sci 2014;270:255–87.
[9] Shen H, Roysam B, Stewart CV, Turner JN, Tanenbaum HL. Optimal scheduling of tracing computations for real-time vascular landmark extraction from retinal fundus images. IEEE Trans Inf Technol Biomed 2001;5(1):77–91.
[10] Tian H, Shen H, Sang Y. Maximizing network lifetime in wireless sensor networks with regular topologies. J Supercomput 2014;69(2):512–27.
[11] Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I. Improving MapReduce performance in heterogeneous environments. Proceedings of the 8th USENIX conference on Operating Systems Design and Implementation, OSDI'08. Berkeley (CA, USA): USENIX Association; 2008. p. 29–42.
[12] Li L, Tang Z, Li R, Yang L. New improvement of the Hadoop relevant data locality scheduling algorithm based on late. In: 2011 International conference on Mechatronic Science, Electric Engineering and Computer (MEC); 2011. p. 1419–22.
[13] Apache HBase. <http://hbase.apache.org/> [accessed on 12.11.14].
[14] Apache zookeeper. <http://zookeeper.apache.org/> [accessed on 12.11.14].
[15] Apache pig. <http://pig.apache.org/> [accessed on 12.11.14].
[16] Apache hive. <https://hive.apache.org/> [accessed on 12.11.14].
[17] Chen Q, Liu C, Xiao Z. Improving MapReduce performance using smart speculative execution strategy. IEEE Trans Comput 2014;63(4):954–67.
[18] Wu H, Li K, Tang Z, Zhang L. A heuristic speculative execution strategy in heterogeneous distributed environments. In: 2014 Sixth international symposium on Parallel Architectures, Algorithms and Programming (PAAP); 2014. p. 268–73.
[19] Dinda PA, O'Hallaron DR. Host load prediction using linear models. Cluster Comput 2000;3(4):265–80.
[20] Box GEP, Jenkins G. Time series analysis, forecasting and control. Holden-Day, Incorporated; 1990.
[21] White T. Time series analysis, forecasting and control. O'Reilly Media; 2012.
[22] Apache Hadoop NextGen MapReduce (yarn). <http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/YARN.html> [accessed on 12.11.14].

Xin Huang received his M.S. in computer science from the Hunan University, Changsha, in 2010. He is currently pursuing his Ph.D. degree at the Hunan University in computer sciences. His research interests include parallel high-performance computing, cloud computing and automotive distributed systems.

Longxin Zhang is working towards the Ph.D. degree at the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include real-time systems, power aware computing and fault-tolerant systems, modeling and scheduling for distributed computing systems, distributed system reliability, parallel algorithms, cloud computing, and big data computing.

Renfa Li received his B.S., M.S. and Ph.D. in computer science from Tianjin University, Tianjin. He has been a professor and Ph.D. supervisor in Hunan University since 2000. His research interests include embed systems, cyber-physical systems and wireless sensor networks.

Lanjun Wan received his M.S. degree in computer science from the Hunan University of Technology in 2009. He is currently pursuing his Ph.D. degree at the college of computer science and electronic engineering in Hunan University. His research interests include high performance parallel computing, parallel algorithm design and implementations, and hybrid CPU-GPU computing.

Keqin Li is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing. He is an IEEE fellow.