A Speculative Execution Strategy Based on Node Classification and Hierarchy Index Mechanism for Heterogeneous Hadoop Systems

Qi Liu*, Weidong Cai*, Jian Shen*, Zhangjie Fu**, Xiaodong Liu***, Nigel Linge****

*Nanjing University of Information Science and Technology, 219 Ningliu Road, Nanjing, Jiangsu, 210044, China

** Jiangsu Engineering Centre of Network Monitoring, Nanjing University of Information Science and Technology, Nanjing, Jiangsu, China

***School of Computing, Edinburgh Napier University, 10 Colinton Road, Edinburgh EH10 5DT, UK

****The University of Salford, Salford, Greater Manchester, M5 4WT, UK

qi.liu@nuist.edu.cn, caiweidongsuzhou@163.com, s_shenjian@126.com, wwwfzj@126.com, x.liu@napier.ac.uk,n.linge@salford.ac.uk

***Abstract*—MapReduce (MR) has been widely used to process distributed large data sets. MRV2 working on Yarn, as a more advanced programing model, has gained lots of concerns. Meanwhile, speculative execution is known as an approach for dealing with same problems by backing up those tasks running on a low performance machine to a higher one. In this paper, we have modified some pitfalls and taken heterogeneous environment into consideration. Besides, Node classification is used and a novel hierarchy index mechanism is created. We also have implemented it in Hadoop-2.6 and the strategy above is called Speculation-NC while optimized Hadoop is called Hadoop-NC. Experiment results show that our method can correctly backup a task, improve the performance of MRV2 and decrease the execution time and resource consumption compared with traditional strategies.**

***Keyword*—MapReduce; Speculative Execution; Time Prediction; Node Classification; Hierarchy Index Mechanism**

Manuscript received March 23, 2016. This work is a follow up of the invited journal of the accepted conference paper for the 18th International Conference on Advanced Communication Technology. This work is supported by the NSFC (61300238, 61300237, 61232016, U1405254, 61373133), Marie Curie Fellowship (701697-CAR-MSCA-IFEF-ST), Basic Research Programs (Natural Science Foundation) of Jiangsu Province (BK20131004), Scientific Support Program of Jiangsu Province (BE2012473) and the PAPD fund.

Q. Liu is with the College of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, 210044, China (e-mail: qi.liu@nuist.edu.cn).

W. Cai is with the College of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, 210044, China (corresponding author to provide phone: +8615251708925; fax: +86-15251708925; e-mail: caiweidongsuzhou@163.com).

S. Jian is with the College of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, 210044, China (e-mail: s_shenjian@126.com).

Z. Fu is with the Jiangsu Engineering Centre of Network Monitoring, Nanjing University of Information Science and Technology, Nanjing, 210044, China (e-mail: wwwfzj@126.com).

X. Liu is with School of Computing, Edinburgh Napier University, 10 Colinton Road, Edinburgh EH10 5DT, UK (E-mail: x.liu@napier.ac.uk).

N. Linge is with the School of Computing, Science and Engineering, University of Salford, Salford, M5 4WT, UK (E-mail: n.linge@salford.ac.uk).

I. INTRODUCTION

THE time that data grow bigger and bigger is coming. The fast development of Cloud Computing makes that big data can be faster and more easily processed [1]. Nimbus provides a high quality of extensible architecture, which supports Xen and KVM virtual machine, and can be used as a PBS resource scheduler. OpenStack is an open source cloud computing platform designed by Rackspace and National Aeronautics and Space Administration (NASA) for public and private cloud. It is compatible with Amazon's EC2 and S3, and can provide the same service as the Amazon [2]. Sector and Spheres [3] is another open source platform designed implemented by Gu, which is used to process geographically dispersed large data sets in parallel. Hadoop cloud computing is one of the most popular open source cloud platforms, it is mainly supported by Yahoo, and applied to the Facebook and Amazon, Twitter, Baidu and other companies like IBM and the New York times. At present, in the latest Release version is 2.7.1. In 2010, Facebook announced that it has the largest Hadoop cluster. In 2011, its data storage space has reached 30 PB.

MapReduce, proposed by Google, which is one of the most important parts in Hadoop, has been the most popular distribute programming model in a cloud environment. With map and reduce procedures used in the cloud infrastructure transparently, those datasets can be processed easily and efficiently.

Many enterprises and companies obtain a large amount of business profits through analyzing and dealing with a lot of new data in time [4]. Data analysis application may have different complexity, resource requirement and data delivery deadlines. As a result, this diversity creates new requirement of job scheduling, workload management and program design. Although different users have different goals, a similar goal is to improve the performance of the framework. Several projects have been launched to relieve the difficulty in writing complex data analysis or data mining programs, e.g., Pig [5] and Hive [6] built above the MapReduce engines in the

Hadoop environment.

However, cloud systems suffer from poor load-scheduling strategies, which can consume much more execution time and temporary space than expected. While theoretically infinite computing resources can be provided in a cloud, unreasonable increment of mappers/reducers cannot achieve process efficiency, and may waste more storage to complete. Many optimization schemes have been proposed [7-13].

II. Related Work

General solutions on performance evaluation and load efficiency in a cloud system have been examined and presented. PQR2, an approach to accurate performance evaluation of distributed application in a cloud was presented [8]. Jing et al. presented a model that can predict resource consumption of MapReduce processes based on a classification and regression tree [9].

Mars were developed to run on NVIDIA GPUs, AMD GPUs as well as multicore CPUs and implemented in Hadoop. Mars hides the programming complexity of GPUs behind the simple and familiar MapReduce interface, and automatically manages task partitioning, data distribution, and parallelization on the processors. Six representative applications also have been implemented on Mars The experimental results show that integrating Mars into Hadoop enabled GPU acceleration for a network of PCs [10]. However, the implementation is much more complex while in a cluster, the performance of GPUs and CPUs.

PrIter, a distributed computing framework was developed. The prioritized execution of iterative computations is supported in it. PrIter either stores intermediate data in memory for fast convergence or stores intermediate data in files for scaling to larger data sets. PrIter achieves up to 50 × speedup over Hadoop for a series of iterative algorithms. In addition, PrIter is shown better performance for iterative computations than other state-of-the-art distributed frameworks such as Spark and Piccolo [11]. But, the biggest pitfall is that it only adapts to iterative algorithms and cannot be applied to all algorithms.

Besides the above methods, some researchers are studying optimizing the speculative execution strategy in MapReduce. Zaharia et al. [12] proposed a modified version of speculative execution called Longest Approximate Time to End (LATE) algorithm that uses a different metric to schedule tasks for speculative execution. Instead of considering the progress made by a task so far, they compute the estimated time remaining, which gives a more clear assessment of a straggling tasks' impact on the overall job response time. But the time every stage occupies is not stable and the std representing standard deviation used in LATE cannot represent all cases. To solve the disadvantages in LATE, MCP [13] was proposed by QI CHEN et al. MCP uses average progress rate to identify slow tasks while in reality the progress rate can be unstable and misleading. Straggles can be appropriately judged when there is data skew among the tasks. However, there are still a lot of pitfalls in MCP. Moreover, it can only be used in MR, not including MRV2.

All research works above have devoted themselves to particular and/or comprehensive load and usage efficiency in a distributed environment. However, in a Heterogeneous environment, reasonable scheduling is still a hard problem. In LATE, MCP or the newest speculative execution strategy in Hadoop-2.6, the biggest pitfall is that situation of heterogeneous environment is not well concerned.

III. Our Strategy

A. Remaining Time Estimation of Current Task

Remaining time of the current task can be calculated by adding the remaining time of current phase and following phases. Detailed method is shown in (1) (2) and (3). $factor_d$ represents the volume of current input data and average value of historical tasks. where T_{rem} represents the remaining time of current task, which consists of the remaining time of current phase and following phases, marked as T_{cp} and T_{fp} . Avg_p includes the average consumption of shuffle phase, sort phase and reduce phase on some node. While the type of current task is map task, the remaining time only includes map phase.

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

$$T_{rem} = \frac{rem\_data_{cp}}{bandwidth_{cp}} + \sum_{p}^{in}{}_{fp} Avg_p * factor_p \tag{2}$$

$$factor_d = \frac{data_{input}}{data_{avg}} \tag{3}$$

While there is no historical information on the node, global historical data is used to calculate the average finishing time of current phase

B. Time Prediction of a Backup Task

When a task finishes, our strategy automatically stores task running time on some node. If current task is a map task, stored data would be hostname and average duration. Otherwise, while it is a reduce task, duration of per-phase would be written down and the duration of whole phase would be recorded at the same time. 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.

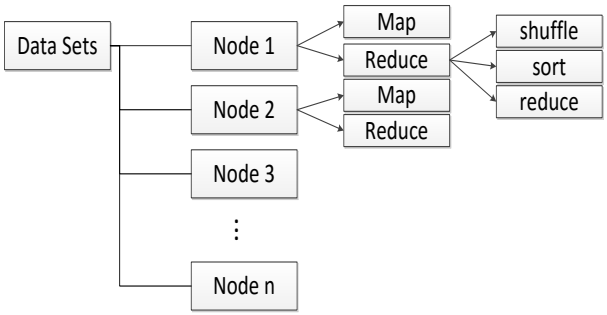

Fig.1 Hierarchy Index Mechanism

As show in Fig.1, a data set is found according to hostname at the beginning of storage procedure. In a same data set, data are collected from the task running on the same host. Every storage area is further divided in two sub storage areas called Map and Reduce. If the task finished just now is map type, the running time is directly recorded in Map. Moreover, while it is a reduce task, detailed running information, containing shuffle time, sort time and reduce time, is respectively written

in 3 sub blocks. Finally, summary information of last level (shuffle sort or reduce) is stored in its up level: Reduce. We need to search some data, a binary search is started based on index mechanism.

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 a data set hostname mapping to is empty, overall running time of different hosts would replace the detailed one. Otherwise, average time of some phase is directly obtained. Detailed algorithm is shown in Algorithm 1.

Algorithm 1 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 data set is empty
4 If $Type$ equals Map
5 Get the average running time of different hosts, Recorded as Avg_{map}, $T_{back} = Avg_{map}$
6 Else if $Type$ 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
Get the average running time of the same host as Avg_{map} and $T_{back} = Avg_{map}$
13 Else
14 Get sum of per phase average running time of same host, Recorded as T_{back}
15 End If
16 End If
17 Return T_{back}

C. Node Selection and Task selection

Node selection is import for that data-local would help framework execute job faster. Besides, node to be selected should have relatively good performance at that time. The object of task selection is judging whether a task runs on a poor performance node. Also, in our strategy, the condition of start a back task is continuously changing according to (4). That is to say, if current progress is 50% and T_{rem} equals 15 seconds, T_{back} must smaller than 10, then the backup task would start. Once a backup task starts, we kill the native task immediately.

$$\frac{T_{rem}}{T_{back}} > Condition = 1 + TaskProgress \qquad (4)$$

IV. Experiment And Analysis

In order to test the performance and benefits of our load balancing strategy, we simulate a real environment. Test-bed is made up of a personal computers and a server. The server is equipped with 288 GB of memory, a 10 TB SATA driver. The personal computers is equipped with 12GB of memory, a single 500GB disk and four Intel(R) Core(TM) 2.4GHz two-core processors. On the server, we run eight virtual machines and each virtual machine is given different amounts of memory and different number of processors. The detailed information is shown in Table I in Section IV.

Then, the virtual machines run as the data nodes and the server run as the name node. To evaluate the performance of load strategy, we use WordCount, Sort and Grep. They are common applications used in MRV2 to evaluate strategy. The Purdue Benchmarks Suite provides us with this application workload and we use the free datasets [14] as the workloads input. Detailed inputs of these applications are shown in Table II.

TABLE I
THE DETAILED INFORMATION OF EACH VIRTUAL MACHINE

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

TABLE II
INPUT OF THESE APPLICATIONS

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

The evaluation method is this paper can be expressed as the Eq. (5), which represents the different between another strategy and our Hadoop-NC divided by another strategy.

$$Improvement = \frac{OtherStrategy - NC}{OtherStrategy} \qquad (5)$$

A. Evaluation of Node Classification

To prove Node of classification is reasonable, we first ran WordCount for 5 times under the circumstance that speculative execution is disable. Then we collected the running information of map phase, and average consuming time was calculated.

Fig.2 shows average consuming time of each node to finish a task. Difference is obvious and tasks running on node3 consume much longer time than others. This proves that it is unreasonable to use mean consuming time to calculate.

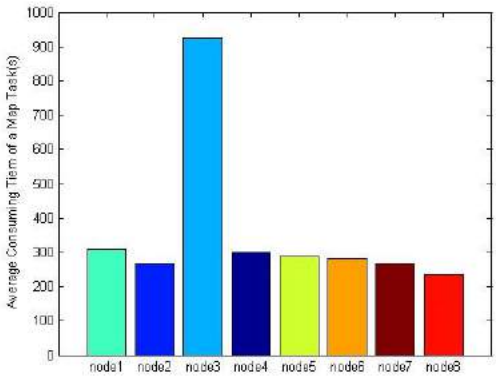

Fig.2 Average Consuming Time of Each Node

Fig.3 shows detailed running time of tasks on node1 and node8. This picture is depicted by 24 groups of sample s from all samples. Although lots of peaks in the figure, if we replace these data with mean value, these 2 lines would be relatively smooth. Those points have been smoothed can be seen as tasks generated by soft straggler, as shown in Fig.4.

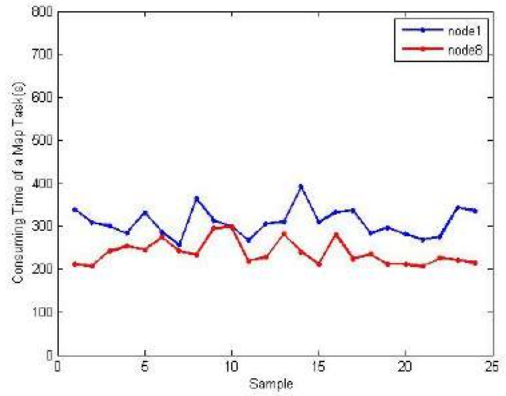

Fig.3 Detailed Running Time of Tasks on node1 and node8

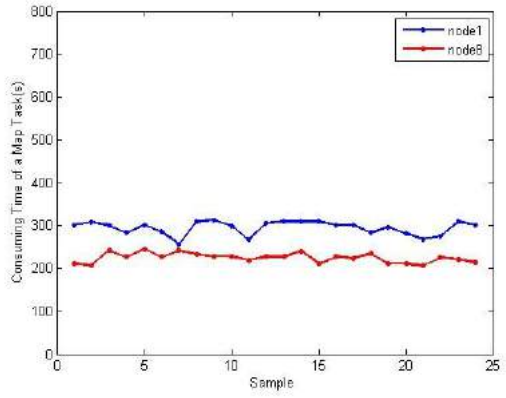

Fig.4 Data After Smoothing

Concluded from Fig.3 and Fig.4, peaks are in Fig.3 can been seen as software straggles. And in figure.4, we assume that this the straggles have been processed and these tasks have been transferred in other nodes. Then, the tendency of time that tasks running on the same node is relative stable. So, we our method classify the time according to node is reasonable. Based on these, further experiments were operated, we used WordCount and Sort to evaluate our performance, which can respectively represent 2 types of application that one consumes CPU and the other is more sensitive to Memory.

B. Evaluation of Hierarchy Index Mechanism

In this part, we will evaluate it from the prosperity of time complexity.

If we put all running information into a same list, then, when we search the list for a piece of data, the time complexity of search would be O(k) (k represents the number of all pieces of data). However, in our storage mechanism, if we successfully find data, average search time can be can simplified as $\log_2(n)+k/n$ (n is the number of nodes). So the average time complexity can be simplified as $O(\log_2(n))$.

Obviously, our hierarchy index mechanism can get a better performance, which has a smaller time complexity.

C. Overall Evaluation of Performance

Fig.5 shows for WordCount case, NC finishes jobs 12% faster than Hadoop-Original and 20% faster than Hadoop-None on average.

Fig.6 shows that on average, NC finishes jobs 3% faster than Hadoop-Original and 19% faster than Hadoop-None. It is very terrible that Original speculative execution strategy has poor performance, but our Hadoop-NC still can reduce the execution time.

Fig.7 shows for Grep case, NC finishes jobs 18% faster than Hadoop-Original and 10% faster than Hadoop-None on average.

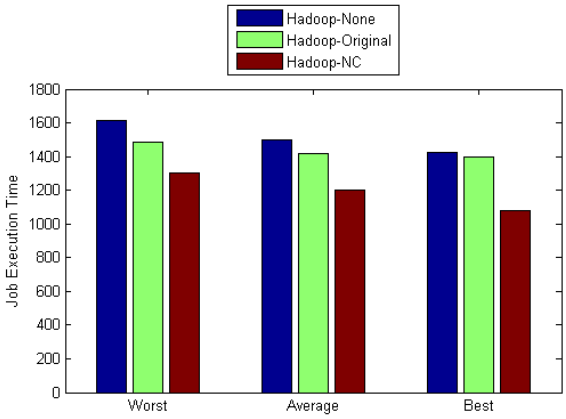

Fig.5 Job Execution Time of WordCount

To analyze the reason why Original strategy has worse performance, we calculated the running information of these two applications under different strategies and drew a table.

As shown in Table III. For WordCount, the backup success rate of Hadoop-NC is 20% and 16.7% higher than Hadoop-Original for map and reduce tasks. For Sort, the backup success rate of Hadoop-NC is 24.3% and 30.7% higher than Hadoop-Original. For Grep, the backup success rate of Hadoop-NC is 30% higher than Hadoop-Original.

Also, resource consumption in cloud environment is also an important indicator to evaluate the performance of PaaS platform. Traditional speculative strategies only evaluate the performance from the prosperity of job execution time. However, resource consumption is usually ignored. In this paper, resource consumption is represented the number of containers and the time of the container occupied. So an Eq. is gotten as shown in (6).

$$Consumption = Containers * Seconds \quad (6)$$

TABLE III
DETAILED COMPARISON

Workloads	Strategy	Sum of Backed-up Map Tasks	Sum of Successful Backed-up Map Tasks	Backup success rate (%)
WordCount	Hadoop-Original	85	40	47.1
	Hadoop-NC	55	38	69.1
Sort	Hadoop-Original	84	24	28.6
	Hadoop-NC	70	37	52.9
Grep	Hadoop-Original	68	32	47.1
	Hadoop-NC	35	27	77.1

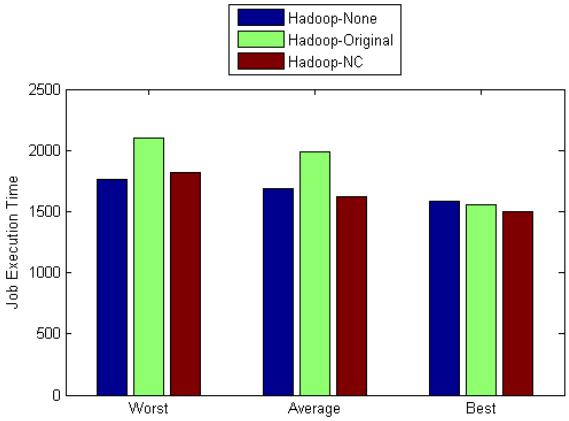

Fig.6 Job Execution Time of Sort

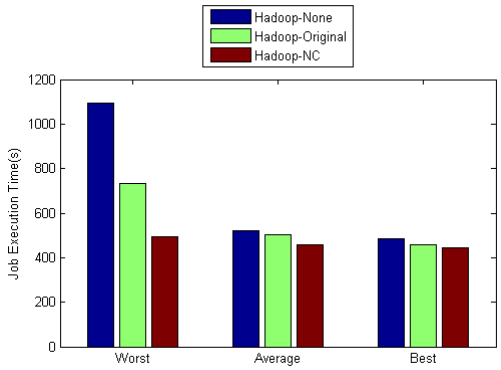

Fig.7 Job Execution Time of Grep

When it comes to resource consumption, shown in Fig.8, our strategy has an obvious improvement for WordCount samples, it save about 10% and 13% than Hadoop-None and Hadoop-Original. However, for Sort samples, although our strategy has an improvement over Hadoop-Original, more resource than Hadoop-None is consumed. It is very normal that if the speculative strategy cannot find straggle on time. For Gerp, because it consumes little time and the saved resource is not very obvious, but our strategy still has an improvement compared with Hadoop-None and Hadoop-Original.

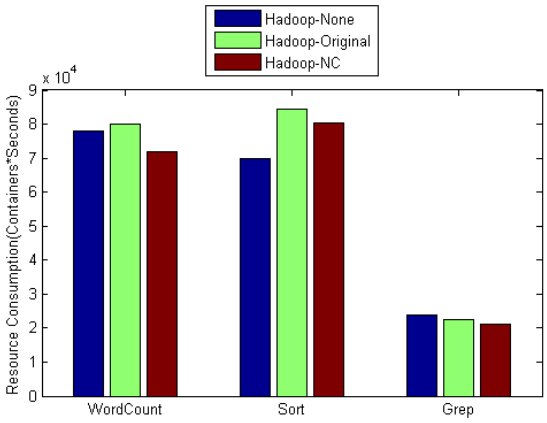

Fig.8 Resource consumption

I. CONCLUSION

In this paper, a new strategy called Speculation-NC is introduced and implemented in Hadoop-2.6. Experiment results have shown that our method can relatively save time and resource for WordCount sample. However, there is still much work can be done to improve the performance of MRV2 for Sort sample further, both in execution and resource consumption.

REFERENCES

[1] 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.

[2] X. Wen, G. Gu, Q. Li, Y. Gao and X. Zhang, "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," *In Proceedings of the 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pp. 2457-2461, 2012.

[3] Y. Gu and L. R. Grossman, "Sector and Sphere: the design and implementation of a high-performance data cloud," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1897, pp. 2429-2445, 2009.

[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.

[5] Apache pig. <http://pig.apache.org/> [accessed on 14.09.15].

[6] Apache hive. <https://hive.apache.org/> [accessed on 14.09.15].

[7] 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. 98, no. 1, pp. 190-200, 2015.

[8] A. Matsunaga and J. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 495-504, 2010.

[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.
[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.
[11] Piao, J. Tai and J. Yan, "Computing resource prediction for mapreduce applications using decision tree," *In Web Technologies and Applications*, pp. 570-577, 2012.
[12] M. Zaharia, A. Konwinski, A. Joseph, R. Katz and I. Stoica, "Improving Mapreduce Performance in Heterogeneous Environments," *OSDI*, vol. 8, no. 4, 2008.
[13] 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.
[14] F. Ahmad, S. Chakradhar, A. Raghunathan and T. Vijaykumar, "Tarazu: optimizing MapReduce on heterogeneous clusters," *ACM SIGARCH Computer Architecture News*, pp.61-74, 2012.

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 her 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. Hi research interests include Cloud Computing, Distributed Computing 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.