Distributed Task Scheduling with Security and Outage Constraints in MapReduce

Wei Hu
School of Computer Science and Engineering
Southeast University
Nanjing, China, 211189
Email: huwei@seu.edu.cn

Jialin Qian
School of Computer Science and Engineering
Southeast University
Nanjing, China, 211189
Email: jlqian@seu.edu.cn

Xiaoping Li
School of Computer Science and Engineering
Southeast University
Nanjing, China, 211189
Email: xpli@seu.edu.cn

***Abstract*—The emergence of MapReduce, a simple software framework, is helping to deal with vast amount of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of n-odes) of commodity hardware in a reliable, fault-tolerant manner. Extensive researches and popularity are gained by MapReduce recently. In this paper, we consider the MapReduce task scheduling problem with security and outage constraints, which are performance effected and not well resolved. The objective is to minimize the makespan while meet data locality and security requirement. A heuristic algorithm with three components is proposed for the problem under study. The simulated results verified the effectiveness of the proposed method, which is closely dependent on the outage probability and the number of worker nodes.**

***Index Terms*—MapReduce, Task Scheduling, Security, Outage**

I. Introduction

Task scheduling is crucial for the performance of MapReduce [1] in big data analysis. Because big data are always distributed in different places, distributed tasks are involved. There are many constraints exerting influence on effectiveness of distributed task scheduling among which security and outage are two important ones.

There are many applications in which security is strict. For example, different roles have different accessing levels in the data center of the Public Security Bureau. Though security has been studied in distributed computing systems (such as Grid Computing and Cloud Computing [2], [3]), it is different from the case in MapReduce [4]. The former is always process oriented and the latter is data oriented. Encryption is a common way to guarantee the security requirement when operating sensitive data which makes the scheduling problem much more difficult.

During the execution of jobs or tasks, failures or cancels might happen. According to the report by Kavulya et al. in [5], the probability is 3%. These failures can be classified into three kinds of failures [6]: task, worker (or node), and master failures. In this paper, we just focus on the node failure because it mostly happens in practical MapReduce systems. Node failures lead to the node outage which happen when a worker node does not accept requests from the master for some reasons, such as system maintenances or unavoidable events. When an outage happens, tasks dispatched on the node have to be migrated to other available nodes.

Though Lin et al. [7] considered the task scheduling only with node outages, we consider distributed MapReduce task scheduling in homogeneous clusters to minimize makespan with security and outage constraints in this paper. To the best of our knowledge, this problem has not been studied yet. A method for the problem is proposed which verifies securities and adjust the outage. The main contributions of this paper are summarized below:

- A mathematical model is constructed for the considered problem.
- Taking into account the security and outage constraints, a heuristic is proposed with three components.
- Parameters of the proposed heuristic are calibrated over a number of instances and the proposal is evaluated.

The remainder of the paper is organized as follows. Related works are reviewed in Section II. Section III constructs the mathematical model of the considered problem. A heuristic is proposed in Section IV. Simulated results and evaluations are shown in Section V, followed by conclusion and future work in Section VI.

II. Related Work

There are many existing works on task scheduling in MapReduce from different views. Various constraints are taken into account in the scheduling problems. Commonly considered constraints are deadlines [8], [9], budgets [10], [11], and data locality [12]–[14].

Security is important in scheduling problems in cloud computing [3], [4] and big data fields. However, there are only a few works on security related task scheduling in Mapreduce. Zhang et al. [15] developed a MapRedue-based system Sedic to automatically partition a computing job according to the security levels of the data. Computations are arranged to hybrid clouds. Map tasks with sensitive data are always kept on privacy clouds. Inter-cloud communication is simply considered. A Security-Aware and Budget-Aware (SABA) algorithm was presented by Zeng et al. [4]. Taking into account security and cost, immoveable datasets are introduced. A new dynamic workflow scheduling mode is constructed and the proposal

978-1-5090-6199-0/17/$31.00 ©2017 IEEE

obtains better solutions. Sood [3] used encryption to protect data by providing user identity and key authentication in the scheme, which is suitable for cloud computing. In fact, encryption is not commonly adopted because it is expensive and hard to scale.

Some researchers studied task scheduling problems in cloud computing with outage consideration. Lin et al. [7] improved the Hadoop's fault tolerance by utilizing a checkpointing mechanism for map tasks. Node or task outages are concerned in the pattern of online scheduling.

In addition, batch job scheduling is always studied to minimize makespan. Verma et al. [16] allocated resources to a batch of jobs in terms of the estimated makespan. Three heuristics were designed by Li et al. [17] to resolve the general two-stage hybrid flowshop scheduling problem with schedule-dependent setup times. But no security constraint has been taken into account when scheduling batch jobs.

III. Problem Description and Mathematical Model

A set of MapReduce jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ are allocated to a MapReduce cluster with P nodes, $\{N_1, N_2, \cdots, N_P\}$. Each node has one or more slots. We denote the set of slots S which consists of the map slot set S_m and the reduce slot set S_r, i.e., $S = S_m \bigcup S_r$. There are q_m map slots in S_m and q_r reduce slots in S_r, i.e., there are $q = q_m + q_r$ slots in total in the cluster. $a \in \{m, r\}$ denotes a MapReduce phase where m is the map phase and r the reduce phase. $V_i^a = \{v_{i,j}^a\}$ is the task set of job J_i in phase a. Let $\mathcal{T}_a$ be the set of all tasks in $\mathcal{J}$ in the phase a, i.e., $\mathcal{T}_a = \bigcup_{i=1}^n V_i^a$. The task priority vector $L_{i,j}^m$ represents priorities of task $v_{i,j}^m$ on each map slot. There are q_m elements in $L_{i,j}^m$ for the q_m map slots in a cluster. $l_{i,j,v}^m \in \{0, 1\}$ is a decision variable. $l_{i,j,v}^m = 1$ if the priority of task $v_{i,j}^m$ is high enough to be processed on slot v. Otherwise, $l_{i,j,v}^m = 0$. $h_{u,v} \in \{-1, 0, 1\}$ is the security level of migrating data from slot u to slot v. There are q_m security levels for the q_m map slots. $h_{u,v} = -1$ means that encryption is required to transfer data from slot u to slot v, $h_{u,v} = 0$ implies no data is transferred from slot u to slot v when they are on the same node, $h_{u,v} = 1$ demonstrates that data on slot u can be migrated to slot v without encryption. Both $L_{i,j}^m$ and H_u are obtained by experiences from historical executing data.

The start time $b_{i,j}^m$ of the map phase of task $v_{i,j}^a$ is initialized to 0 if there is no task processed before $v_{i,j}^a$. The task processing time $p_{i,j}^a$ of task $v_{i,j}^a$ is estimated by historical executions. We assume that the input data of each map task is located on one node and each task can be performed only by one slot. The setup time $s_{i,j}^a$ is determined by five factors: the data size, data location, the communication rate, the priority of the task, and the security of the data. The completion time $c_{i,j}^m$ of task $v_{i,j}^m$ is calculated by

$$c_{i,j}^m = b_{i,j}^m + x_{i,j,v} \times s_{i,j,v}^m + p_{i,j}^m$$

where $x_{i,j,v}$ is a decision variable defined by $x_{i,j,v} = \begin{cases} 0 & \text{if } \; l_{i,j,v}^m = 1 \\ 1 & \text{otherwise} \end{cases}$. Task $v_{i,j}^m$ is allowed to be executed on slot u if $l_{i,j,u}^m = 1$. Otherwise, another slot v satisfying the task priority constraint is found to execute $v_{i,j}^m$ and the required data are transferred from u to v. Data encryption is checked before transferring based on the security vector. $s_{i,j,v}^m = s_{i,j}^{u,v} + y_{u,v} \times E_{i,j}^u$ in which $s_{i,j}^{u,v} = g_{i,j}^u / \tau_t$ and $E_{i,j}^u = g_{i,j}^u / \tau_e$. $g_{i,j}^u$ is the input data size of $v_{i,j}^m$ on the node to which the map slot u belongs. τ_t and τ_e are communication rates of data transferring and data encryption respectively. $y_{u,v}$ is a decision variable defined as $y_{u,v} = \begin{cases} 1 & \text{if } \; h_{u,v} = -1 \\ 0 & \text{otherwise} \end{cases}$.

For reduce tasks, we assume that they cannot start until all map tasks of the same job have completed. Therefore, the start time $b_{i,j}^r$ of reduce task $v_{i,j}^r$ is the latest completion time of all map tasks of J_i. Similarly, we have

$$c_{i,j}^r = b_{i,j}^r + s_{i,j}^r + p_{i,j}^r$$

where $s_{i,j}^r = \frac{\sum_{k=1}^{|V_i^m|} d_{i,j}^k}{|V_i^r| \tau_t}$. $d_{i,j}^k$ is the data size needed by $v_{i,j}^r$ from the output of map task $v_{i,k}^m$. Therefore, for $\forall i \in \{1, 2, \cdots, n\}$ and $j \in \{1, 2, \cdots, |V_i^r|\}$, the objective of the problem under study can be calculated by

$$C_{max} = \min\{\max\{c_{i,j}^r\}\} \tag{1}$$

IV. Proposed Scheduling Method

For the considered problem, a scheduling method SMSO (Security-Aware with Outages Scheduling method) is proposed which includes three components: task matchmaking, queue ordering, outage verifying.

A. Task matchmaking

Task matchmaking tries to find the possible set of map slots S_m to allocate map tasks $\mathcal{T}_m$ to them. Data locality, task priorities and data security constraints are considered when trying to dispatch the tasks. In addition, load balance of the cluster is controlled by a threshold θ (θ=1.2 in this paper). The map slot u on node N_k ($k \in \{1, 2, \cdots, P\}$) is found for the pending task $v_{i,j}^m$ which owns the required data. Task $v_{i,j}^m$ is appended to the end of task queue M_u if the load balance and task priority requirements are satisfied simultaneously. Otherwise the priority of the task $v_{i,j}^m$ is relaxed. The following strategy is executed for tasks with lower priorities: The security level vector H_u and the task priority vector $L_{i,j}^m$ are checked until a slot meeting the data security, the task priority and load balancing requirements is found. Otherwise, task $v_{i,j}^m$ is allocated to the node with the least number of tasks for load trade-off consideration. The task matchmaking algorithm is formally described in Algorithm 1. The time complexity of Algorithm 1 is $O(|\mathcal{T}_m| q_m^2)$.

B. Queue ordering

After map tasks are matched in all slots, map tasks in each M_u are ordered using the LPT strategy. When all map tasks of J_i finish, the reduce tasks of J_i are allocated to the slot u with the earliest available time. C_{max} is calculated according to Equation (1).

Algorithm 1: Task matchmaking (TMM)

1 Input: The map task set $\mathcal{T}_m$, the map slot set S_m.
2 Output: A set of map task sequences $\mathcal{R}_m$.
3 **begin**
4 **for** *each* $v_{i,j}^m \in \mathcal{T}_m$ **do**
5 **for** *each* $u \in S_m$ **do**
6 **if** $l_{i,j,u}^m$ = *1* **then**
7 **if** $|M_u| < (|\mathcal{T}_m|/q_m) \times \theta$ **then**
8 $M_u \leftarrow M_u \cup \{v_{i,j}^m\}$;
9 Continue; `// Go to step (4) to deal with the next` $v_{i,j}^m$;
10 **else**
11 **for** *each* $h_{u,v} \in H_u$ **do**
12 **if** $h_{u,v}$= *1* && $l_{i,j,v}^m$ = *1* && $|M_v| < (|\mathcal{T}_m|/q_m) \times \theta$ **then**
13 $M_v \leftarrow M_v \cup \{v_{i,j}^m\}$;
14 Continue; `// Go to step (4) to deal with the next` $v_{i,j}^m$;
15 **for** *each* $w \in S_m$ **do**
16 Find the w with $\min |M_w|$;
17 $M_w \leftarrow M_w \cup \{v_{i,j}^m\}$;
18 Continue; `// Go to step (4) to deal with the next` $v_{i,j}^m$;
19 **for** *each* $u \in S_m$ **do**
20 $\mathcal{R}_m \leftarrow \mathcal{R}_m \cup M_u$;
21 **return** $\mathcal{R}_m$.

The time complexity of Steps 4 $\sim$ 5 is $O(q_m|M_u|\log|M_u|)$, that of Steps 6 $\sim$ 11 is $O(|n(|V_i^m| + q_r + |V_i^r|\log|V_i^r| + |V_i^r|)$, and that of Steps 12 $\sim$ 13 is $O(q)$. Therefore, the time complexity of the Queue Ordering procedure is $O(q_m|M_u|\log|M_u| + n(|V_i^m| + q_r + |V_i^r|\log|V_i^r|))$.

C. Outage verifying

Statuses of the nodes are checked by outage verifying. When the master node finishes the matchmaking and ordering procedures, it sends a heartbeat to all of its worker nodes. Once each worker node receives the heartbeat message, it sends back a response message to the master node. If the master node does not receive the response after a period of time from a worker node, it assumes the worker node in outage. Tasks dispatched on the outage node are reassigned to other available and suitable worker nodes. In this process, outage nodes N_{outage} and the task queues on them $\mathcal{T}_{outage}$ for reassignment are kept. The outage verifying process is formally described in Algorithm 3.

Algorithm 2: Queue ordering (QO)

1 Input:A set of map task sequences $\mathcal{R}_m$;
2 Output: Map and reduce task sequences $\mathcal{R}$;
3 **begin**
4 **for** *each* $u \in S_m$ **do**
5 Sort map tasks in the non-increasing order of their processing times;
6 **for** *each* $J_i \in \mathcal{J}$ **do**
7 Find J_i with $\min\{\max_{j\in\{1,2,\ldots,|V_i^m|\}} C_{i,j}^m\}$;
8 Find u with $\min_{u\in\{1,2,\ldots,q_r\}}\{F_u^r\}$;
9 Sort reduce tasks in V_i^r in the non-increasing order of their processing times;
10 **for** *each* $v_{i,j}^r \in V_i^r$ **do**
11 $M_u \leftarrow M_u \cup \{v_{i,j}^r\}$;
12 **for** *each* $u \in S$ **do**
13 $\mathcal{R} \leftarrow \mathcal{R} \cup M_u$
14 **return** $\mathcal{R}$;

Though there are two loops, the total number of checking is q in the worst case, the time complexity of Algorithm 3 is $O(q)$.

Algorithm 3: Outage verifying (OV)

1 Input: The ordered queue list $\mathcal{R}$;
2 Output: Reassigned task set $\mathcal{T}_{outage}$; Outage node set N_{outage};
3 **begin**
4 $flag \leftarrow true$, $N_{outage} \leftarrow \emptyset$;
5 **for** *each node* N_k **do**
6 **if** *the heartbeat on* N_k *is overtime* **then**
7 **for** *each slot* i *on node* N_k **do**
8 $\mathcal{T}_{outage} \leftarrow \mathcal{T}_{outage} \cup M_i$;
9 $M_i \leftarrow \emptyset$;
10 $N_{outage} \leftarrow N_{outage} \cup \{N_k\}$;
11 $flag \leftarrow false$;
12 Update S_m by removing all slots in N_{outage};
13 **return** $flag$;

D. Scheduling Method with Security-aware Outages

Based on the above three components, the proposed Security-Aware with Outages Scheduling (SMSO) method is formally depicted in Algorithm 4. Because the iteration number cannot be predicted, it is hard to estimate the time complexity of SMSO.

V. Simulated Results

To evaluate effectiveness, the proposed method is performed over a large number of instances. The algorithm is encoded in Java, compiled with Eclipse Helios Release JDK1.6 and ran

Algorithm 4: Scheduling Method with Security-aware Outages (SMSO)

begin
 $\mathcal{R}_m \leftarrow TMM(\mathcal{T}_m, S_m)$;
 $\mathcal{R} \leftarrow QO(\mathcal{R}_m)$;
 while $OV(\mathcal{R})$ **do**
  $\mathcal{R}_m \leftarrow TMM(\mathcal{T}_{outage}, S_m)$;
  $\mathcal{R} \leftarrow QO(\mathcal{R}_m)$;
 return $\mathcal{R}$;

on a PC with an Intel Core i5-3479 3.7GHz processor and 4GB RAM.

Let γ be the initial outage probability of nodes, which $\gamma \in \{0, 0.005, 0.01, 0.015, 0.02,\ 0.025, 0.03\}$. The number of nodes q takes values from $\{10, 20, 50, 100, 150\}$. Ten instances are randomly generated for each job number $n \in$\{20, 50, 200, 500\}, i.e., there are $4 \times 10 = 40$ instances. Combined with q and γ, there are $5 \times 7 \times 40 = 1400$ tests in total. The number of map tasks is stochastically generated in normal distribution by $N(154, 558^2)$ and the number of reduce tasks is normally distributed in $N(19, 145^2)$. Processing times of tasks are uniformly distributed in $[0, 2020)$. The communication and encryption rates are set as 100MB/s and 50 MB/s respectively. Let C^*_{max} be the best solution found so far. RPD (Relative Percentage Deviation) is used to measure the performance of the proposed method which is defined as below

$$RPD = \frac{C_{max} - C^*_{max}}{C^*_{max}} \times 100\%$$

A. Parameter Calibration

The two parameters γ and q are tested on random instances according to the above setting manners. We analyze the results by the Analysis of Variance technique (ANOVA) which is a very robust parametric technique. A number of hypotheses should be ideally met by the experimental data. Among these, the main three are (in order of importance): independence of the residuals, homoscedasticity or homogeneity of the factor's levels variance and normality in the residuals of the model. Apart from a slight non-normality in the residuals, we can accept all hypotheses easily. The response variable in the experiments is the RPD for each algorithm in every instance. The mean plots and 95.0% Tukey HSD intervals on RPD for γ and q are shown in Figures 1 and 2, respectively.

Figure 1 indicates that RPD increases with the increase of γ, i.e., RPD gets the smallest value when γ is 0 and the largest value when γ is 0.3. The observed differences are not always statistically significant when $\gamma > 0$. Therefore, we test two cases both when $\gamma = 0$ and when $\gamma = 0.3$ in the following experiments. From Figure 2, we can observe that RPD also increases with the increase of q. However, the differences are statistically significant for $q < 50$ and $q \geq 50$ though the differences are not statistically significant in each interval.

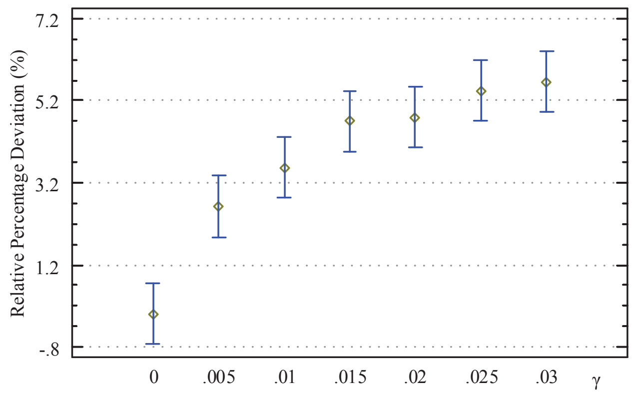

Fig. 1. The mean plots and 95.0% Tukey HSD intervals on RPD for γ.

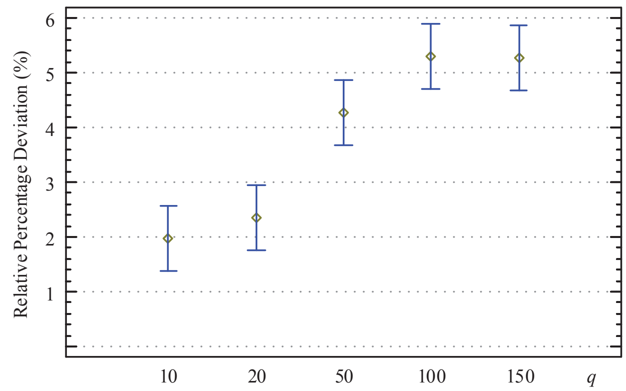

Fig. 2. The mean plots and 95.0% Tukey HSD intervals on RPD for q.

B. Algorithm Evaluation

We evaluate the proposed algorithm SMSO on a lot of random instances. C_{max} values and RPD over 10 instances for $\gamma = 0$, and $\gamma = 0.03$ are listed in Table I respectively for all q cases when job number $n = 500$. Table I implies that the outage probability γ exerts influence on performance of the algorithm. The makespan increases as increasing from $\gamma = 0$ to $\gamma = 0.03$, which in accordance with practice, i,.e., reallocated resources always result in longer makespan. However, the increasing rate is less than 20% and the rates are less than 10% for most cases. In other words, the proposed SMSO method is effective in settings with certain outages.

VI. Conclusion and Future work

In this paper, we considered a kind of MapReduce task scheduling problems with security and outage constraints to minimize makespan of batch jobs in a homogeneous cluster. The problem was mathematically modelled. An effective composite method with three components was proposed and evaluated over a great number of instances. Simulated results showed that performance of the proposal depends mainly on the outage probability and the number of nodes in the cluster.

In the future, similar scheduling problem in heterogeneous cluster environment are promising topics which are widespread in practice.

TABLE I
C_{max} AND RPD OF 10 INSTANCES FOR $\gamma = 0$, AND $\gamma = 0.03$ WHEN JOB NUMBER $n = 500$.

Instance	γ	Node number				
		10	20	50	100	150
1	0	599354	290198	118445	60482	41935
	0.03	630758	305137	121610	66970	41555
RPD		**5.24%**	**5.15%**	**2.67%**	**10.73%**	**0.96%**
2	0	589509	292573	116249	59961	40781
	0.03	701560	312911	121779	63135	43056
RPD		**19.01%**	**6.95%**	**4.76%**	**5.29%**	**5.58%**
3	0	588378	295599	119913	60502	41031
	0.03	622599	302662	127130	62758	42614
RPD		**5.82%**	**2.39%**	**6.02%**	**3.73%**	**3.86%**
4	0	584398	292851	114921	58425	42038
	0.03	614845	306202	116942	60389	41466
RPD		**5.21%**	**4.56%**	**1.76%**	**3.36%**	**1.38%**
5	0	602515	296501	118801	60760	41613
	0.03	627487	312657	123813	61422	42303
RPD		**4.14%**	**5.45%**	**4.22%**	**1.09%**	**1.66%**
6	0	610013	302088	118763	58486	41291
	0.03	610013	302088	125372	61210	43211
RPD		**0.00%**	**0.00%**	**5.56%**	**4.66%**	**4.65%**
7	0	595108	296636	117375	60604	41136
	0.03	627098	303252	124826	61566	43042
RPD		**5.38%**	**2.23%**	**6.35%**	**1.59%**	**4.63%**
8	0	602657	284826	117569	59690	40722
	0.03	662658	304511	124143	66423	42897
RPD		**9.96%**	**6.91%**	**5.59%**	**11.28%**	**5.34%**
9	0	603540	291642	120952	60260	42627
	0.03	603540	291642	123657	62636	44790
RPD		**0.00%**	**0.00%**	**2.24%**	**3.94%**	**5.07%**
10	0	604919	298924	121644	62620	42088
	0.03	641380	305345	123164	64607	42853
RPD		**6.03%**	**2.15%**	**1.25%**	**3.17%**	**1.82%**

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (No. 61572127, 61272377).

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters." *In Proceedings of Operating Systems Design and Implementation (OSDI*, vol. 51, no. 1, pp. 107–113, 2004.

[2] W. Yu, G. Xu, Z. Chen, and P. Moulema, "A cloud computing based architecture for cyber security situation awareness," in *Communications and Network Security (CNS), 2013 IEEE Conference on*, Oct 2013, pp. 488–492.

[3] S. K. Sood, "A combined approach to ensure data security in cloud computing," *Journal of Network & Computer Applications*, vol. 35, no. 6, pp. 1831–1838, 2012.

[4] L. Zeng, B. Veeravalli, and X. Li, "Saba: A security-aware and budget-aware workflow scheduling strategy in clouds," *Journal of Parallel & Distributed Computing*, vol. 75, pp. 141–151, 2014.

[5] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster." in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010, pp. 94–103.

[6] J. A. Quiane-Ruiz, C. Pinkel, J. Schad, and J. Dittrich, "Rafting mapreduce: Fast recovery on the raft," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, 2011, pp. 589–600.

[7] C. Y. Lin, T. H. Chen, and Y. N. Cheng, "On improving fault tolerance for heterogeneous hadoop mapreduce clusters," in *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, 2013, pp. 38–43.

[8] Z. Tang, J. Zhou, K. Li, and R. Li, "A mapreduce task scheduling algorithm for deadline constraints," *Cluster Computing*, vol. 16, no. 4, pp. 651–662, 2013.

[9] H. Li, X. Wei, Q. Fu, and L. Yuan, "Mapreduce delay scheduling with deadline constraint," *Concurrency & Computation Practice & Experience*, vol. 26, no. 3, pp. 766–778, 2014.

[10] J. Li, S. Su, X. Cheng, and Q. Huang, "Cost-conscious scheduling for large graph processing in the cloud," in *IEEE International Conference on High PERFORMANCE Computing and Communications*, 2011, pp. 808–813.

[11] A. M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *Cloud Computing, Second International Conference, CloudCom 2010, November 30 - December 3, 2010, Indianapolis, Indiana, USA, Proceedings*, 2010, pp. 351–359.

[12] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 265–278.

[13] M. Hammoud and M. F. Sakr, "Locality-aware reduce task scheduling for mapreduce," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 570–576.

[14] M. Sun, H. Zhuang, X. Zhou, K. Lu, and C. Li, *HPSO: Prefetching Based Scheduling to Improve Data Locality for MapReduce Clusters*. Springer International Publishing, 2014.

[15] K. Zhang, X. Zhou, Y. Chen, X. F. Wang, and Y. Ruan, "Sedic: privacy-aware data intensive computing on hybrid clouds," in *ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, Usa, October*, 2011, pp. 515–526.

[16] A. Verma, L. Cherkasova, and R. H. Campbell, "Aria: Automatic resource inference and allocation for mapreduce environments," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 235–244.

[17] X. Li, T. Jiang, and R. Ruiz, "Heuristics for periodical batch job scheduling in a mapreduce computing framework," *Information Sciences*, vol. 326, pp. 119–133, 2016.