Acta Technica 62 No. 2A/2017, 51–62 © 2017 Institute of Thermomechanics CAS, v.v.i.

The analysis and optimization strategy of network failure recovery

Changtian Ying[1,2], Weiqing Wang[1], Jiong Yu[2], Hong Jiang[2], Lei Qi[2]

Abstract. In the traditional network failure mechanism, the programmer takes the responsibility to select the network checkpoints, which may lead to the problem of unavailable service and much more recovery time. To address this issue, we first analyzed the architecture of network failure in this paper, established the fault tolerance model on the basis of network characteristics, and then proposed the optimization strategy for network failure including the checkpoint algorithm and the recovery algorithm. The checkpoint algorithm chose the appropriate checkpoints based on the analysis of the network, and the recovery algorithm took advantage of the latest checkpoints to recover the lost data. Finally, we conducted the experiments to evaluate, and both of the two datasets had the less recovery time and better recovery ratio. The experiment results verified the validity of the failure recovery strategy.

Key words. Network optimization, unavailable service, failure recovery.

1. Introduction

The past few years have seen a major change in computing systems, as growing data volumes and stalling processor speeds require more and more applications to scale out to distributed systems. To face the challenges brought by the big data [1, 2], data processing technology has come under heated discussion among domestic scholars in recent years. Spark has won more and more recognition and support in the new generation of large data processing framework. It is a general-purpose high-performance parallel computing framework. Spark uses flexible distributed datasets that are called RDD (resilient distributed datasets) as the data structure. If a partition of an RDD is lost, Spark read the checkpoint data, and uses the linage to re-compute that partition.

The checkpoint/recovery strategy is fault-tolerant technology, which has been

[1]Post Doctoral Research Station of Electrical Engineering, Xinjiang University, Urumqi, Xinjiang, 830046, China

[2]School of Software, Xinjiang University, Urumqi, Xinjiang, 830046, China

widely used in cluster computing. Checkpoint is a common traditional strategy in the domestic and foreign research [3]. The classifications of checkpoint technology are the system level, the application level and the user level [4]. Literature [5] analyzed the characteristics and advantages of different examination method, and proposed an application level check pointing technology. And another method is presented to implement a system level checkpoint [6]. This literature proposed automatic fault tolerance technology based on high performance computing system, which can complete the automatic check pointing/recovery [7]. Literature [8] proposed checkpoint/restore library, which could support the different ways of preserving the checkpoint data in the memory. Literature [9] used mixed strategy to fulfill incremental checkpoint, thereby reducing the amount of checkpoint data. Fault tolerance strategy of literature [10] used multilevel storage and encoding redundancy checkpoints, so as to reduce the fault tolerance overhead.

However, in the Spark checkpoint mechanism, the programmers make the decision for selecting the object and time of checkpoints. The checkpoint is set up only when the checkpoint instruction is executed. This implementation increases the uncertainty of checkpoint strategy, and it is much difficult to maximize the checkpoint performance. If the checkpoint strategy is unsuitable, it may not only reduce the application efficiency, but also increase the risk of application exception. Therefore, if the failure optimization algorithm can efficiently implemented, it can ease the burden of programmers, and improve system efficiency and availability.

In this paper, we have conducted a sophisticated theoretical and technical study on checkpoint technology. First we established the task scheduling and recovery efficiency model, then analyzed relevant factors and established the RDD weight model. Then we proposed optimization strategy to max-relieve checkpoint bottleneck and optimize the performance. The experiments verified the effectiveness of the strategy.

2. Problem analysis

Due to the lazy scheduling mechanism, the task is compiled into multiple DAGs when performs action. And each RDD is divided into several partitions to be calculated by the cluster nodes.

Definition 1 - Partition failure rate. RDD partitions are computed in parallel. When considering the hardware failure, partition failure rate is determined by worker failure rate. If the worker failure rate of w_m is wf_m, then the failure rate of the partition PT_{ijk} is denoted as

$$\mathrm{FR}_{\mathrm{PT}_{ijk}} = pb_{ijkm} wf_m \,. \tag{1}$$

Definition 2 - RDD failure rate. If the RDD partition is unavailable, the entire RDD cannot be used. So the failure rate of RDD_i is defined as

$$\mathrm{FR}_{\mathrm{RDD}_{ij}} = \max(\mathrm{FR}_{\mathrm{PT_{ij1}}},\ \mathrm{FR}_{\mathrm{PT_{ij2}}},\ \mathrm{FR}_{\mathrm{PT_{ij3}}},\ \cdots,\ \mathrm{FR}_{\mathrm{PT_{ijk}}})\,. \tag{2}$$

Definition 3 - Task failure rate. When the task is executed, it is compiled into a RDD DAG, and the failure rate of the task is denoted as

$$\mathrm{FR}_{\mathrm{Task}_i} = 1 - \prod_{j=1}^{m}(1 - \mathrm{FR}_{\mathrm{RDD}_{ij}})\,. \tag{3}$$

Definition 4 - Partition recovery cost. Recovering of missing partition needs the parent partitions and checkpoints. If the ancestor PT_{ijp} has been set as checkpoint, then the recovery cost of the partition PT_{ijk} can be defined as

$$R(\mathrm{PT}_{ijk}) = \alpha_i + \mathrm{read}(\mathrm{PT}_{ij(k-1)}) + \mathrm{proc}(\mathrm{PT}_{ijk})\,,$$

$$R(\mathrm{PT}_{ij(k-1)}) = \alpha_i + \mathrm{read}(\mathrm{PT}_{ij(k-2)}) + \mathrm{proc}(\mathrm{PT}_{ij(k-1)})\,,$$

$$R(\mathrm{PT}_{ij(p+1)}) = \alpha_i + \mathrm{read}(\mathrm{PT}_{ijp}) + \mathrm{proc}(\mathrm{PT}_{ij(p+1)})\,.$$

Assume $\mathrm{T}_{\mathrm{PT}_{ijk}}$ as the time cost of PT_{ijk} and $T_{\mathrm{PT}_{ijk}} = \mathrm{read}(\mathrm{PT}_{ijk}) + \mathrm{proc}(\mathrm{PT}_{ijk})$. Then

$$R(\mathrm{PT}_{ijk}) = \alpha_i + \sum_{l=p}^{k} \mathrm{T}_{\mathrm{PT}_{ijk}}\,. \tag{4}$$

Here, α_i denotes the fault detection overhead. If the checkpoints are not set, all RDDs are calculated from the beginning.

Definition 5 - RDD recovery cost. Suppose the checkpoints as the set $C_i = (c_{i1},\ c_{i2},\ \ldots,\ c_{ip})\}$, where C_i is a subset of Task_i, c_{ip} is the newest checkpoint, and RDD_{ik} is the kth RDD of Task_i. The worker is failed while computing RDD_{ij}, then the recovery cost can be denoted as

$$R_{\mathrm{RDD}_{ij}} = \alpha_i + \mathrm{read}(\mathrm{RDD}_{i(k+1)}) + \mathrm{proc}(\mathrm{RDD}_{i(k+1)}) + \mathrm{read}(\mathrm{RDD}_{i(k+2)}) +$$

$$+ \mathrm{proc}(\mathrm{RDD}_{i(k+2)}) + \cdots + \mathrm{read}(\mathrm{RDD}_{ij}) + \mathrm{proc}(\mathrm{RDD}_{ij}) =$$

$$= \alpha_i + T\mathrm{RDD}_{i(k+1)} + T\mathrm{RDD}_{i(k+2)} + \cdots + T\mathrm{RDD}_{ij} =$$

$$= \alpha_i + \sum_{q=k+1}^{j} T\mathrm{RDD}_{iq}\,. \tag{5}$$

where α_i denotes the fault detection overhead.

$$R_{\mathrm{RDD}_{ij}} = \alpha_i + \sum_{q=1}^{j} T_{\mathrm{RDD}_{iq}}\,. \tag{6}$$

Definition 6 - task recovery cost. In the failure case, the task recovery cost during the task execution process is the completion overhead for the task, and the overhead for the lost RDDs to recovery. If the number of failure times is 0, R_{task_i} is the recovery overhead of Task_i. If the number of failure times is k, then fault recovery

overhead is used by the lost RDDs.

$$R_{\mathrm{task}_i} = \sum_{j=1}^{k} R_{\mathrm{RDD}_{ij}} = \sum_{j=1}^{k} (\alpha_i + \sum_{q=1}^{j} T_{\mathrm{RDD}_{iq}}) \,. \tag{7}$$

Definition 7 - RDD life cycle. For RDD_{ij}, the maximum life cycle is from the starting time of RDD_{ij} to the completion time of task. Denote $ST_{\mathrm{RDD}_{ij}}$ as starting computing time of RDD_{ij}, FT_{task_i} as the completion time for the task, then the maximum life cycle of the RDD_{ij} can be defined as

$$\max LC_{\mathrm{RDD}_{ij}} = FT_{\mathrm{task}_i} - ST_{\mathrm{RDD}_{ij}} \,. \tag{8}$$

For RDD_{ij}, the minimum life cycle is from the start time of RDD_{ij} to the last used completion time of RDD_{ij}. Denote $UT_{\mathrm{RDD}_{ij}}$ as RDD_{ij} of the last used completion time

$$\min LC_{\mathrm{RDD}_{ij}} = UT_{\mathrm{RDD}_{ij}} - ST_{\mathrm{RDD}_{ij}} \tag{9}$$

So the range of the RDD life cycle is expressed as

$$\min LC_{\mathrm{RDD}_{ij}} \leq LC_{\mathrm{RDD}_{ij}} \leq \max LC_{\mathrm{RDD}_{ij}} \,. \tag{10}$$

By analyzing the availability, it is easy to know that the failure probability, the selection of the checkpoint time and objects are important factors influencing the recovery efficiency. Suppose the fault probability does not change the situation, then the recovery overhead of the task R(task) is smaller, the task availability is greater. Therefore, the goal of automatic checkpoint strategy is to minimize task recovery overhead while meeting the requirement of system resources. And it can be formalized as

Object: $\min(R_{\mathrm{task}})$, s.t. $\sum_{i \in \mathrm{Tasks}} A_{im} \leq r_m$.

3. Optimization strategy

3.1. Relevant proof

Theorem 1. It is difficult to choose proper period time according to the time.

Proof: select the period checkpoint time need considering the user experience judgment and the prediction task execution time. For the checkpoint time T_i, there are three kinds of states.

1. T_i is too small, $T_i < \min T_{\mathrm{RDD}}$ (RDD being the minimum running time). It may cause frequently storing data to disk and reducing system throughput.

2. T_i is too large, $T_i > \max T_{\mathrm{RDD}}$ (RDD maximum running time). The time interval of the storing data is too long, if the worker power down, it may lead to a large number of data RDD necessary for recalculation.

3. T_i is moderate, $\min T_{\mathrm{RDD}} < T_i < \max T_{\mathrm{RDD}}$. The time interval is moderate, but with the current state of the system, may not be a good fit with the task.

Theorem 2. The long lineage principle. The longer the RDD lineage is, the

greater its recovery overhead is. And it should be gave higher priority.

Proof: Task_i has two RDDs: $\text{RDD}_{i(j-1)}$ and RDD_{ij}. $\text{RDD}_{i(j-1)}$ is father RDD of RDD_{ij}. Suppose the lineage depth of $\text{RDD}_{i(j-1)}$ is k, the lineage depth of RDD_{ij} is $k+1$, then RDD_{ij} has a longer lineage. Denote RDD_{ip} the latest checkpoint, then recovery overheads of two RDDs are

$$R_{\text{RDD}_{i(j-1)}} = \alpha_i + \sum_{q=p+1}^{j} T_{\text{RDD}_{iq}} ,$$

$$R_{\text{RDD}_{ij}} = \alpha_i + \sum_{q=p+1}^{(j-1)} T_{\text{RDD}_{iq}} . \tag{11}$$

As $R_{\text{RDD}_{ij}} - R_{\text{RDD}_{i(j-1)}} = T_{\text{RDD}_{ij}}$, therefore $R_{\text{RDD}_{ij}} > R_{\text{RDD}_{i(j-1)}}$.

That is, the RDD recovery overhead with longer lineage is greater. When the checkpoint is set, the RDD with long lineage should be selected to reduce the recovery overhead.

Theorem 3. Wide dependency principle. The recovery overhead of narrow dependency is larger than that of the wide dependency.

Proof: Recovery overhead from the parent $\text{RDD}_{i(j-1)}$ to RDD_{ij} is

$$R_{\text{RDD}_{ij}} = \alpha_i + \sum_{l=p}^{m} T(PT_{ijl}) . \tag{12}$$

When the parent data RDD_{ij} is fixed, the difference is that the operation is of wide or narrow dependency. If the operation is narrow dependency and the lth partition PT_{ijl} is lost, only the parent of PT_{ijl} is calculated.

$$R_{\text{RDD}_{ij}}(\text{narrow}) = R(PT_{ijl}) = \alpha_i + \text{read}(PT_{ij(l-1)}) + \text{proc}(PT_{ijl}) . \tag{13}$$

If the operation is wide dependency and the lth partition PT_{ijl} is lost, the partition is calculated by all the parent partitions:

$$R_{\text{RDD}_{ij}}(\text{wide}) = R_{\text{RDD}_{ij}} = \max(R(PT_{ij}1), \cdots , R(PT_{ijk}) . \tag{14}$$

Therefore

$$R_{\text{RDD}_{ij}}(\text{wide}) \geq R_{\text{RDD}_{ij}}(\text{narrow}) .$$

We should preferred RDD with wide dependency as checkpoints, thereby reducing the recovery overhead.

Theorem 4. High computation cost principle. If the RDD with higher computation cost is not stored, it may lead higher recovery overhead. So it is necessary to give priority to the RDD with higher cost.

Proof: For Task_i, in the case of the same parameters, when the cost of the RDD calculation is not the same size, the impact on the recovery of the RDD overhead is different.

When the restore is required and the latest checkpoint is RDD_{ip}, the recovery of RDD_{ij} overhead is

$$R_{\mathrm{RDD}_{ij}} = \alpha_i + \sum_{q=1}^{j} T_{\mathrm{RDD}_{iq}} = \alpha_i + \sum_{q=1}^{j-1} T_{\mathrm{RDD}_{iq}} + T_{\mathrm{RDD}_{ij}} \,. \tag{15}$$

While the other parameters are fixed, if the computation cost of RDD_{ij} is greater, the recovery overhead is greater. Therefore, RDD with greater computation cost should be preferred as checkpoint, thereby reducing the recovery overhead.

3.2. RDD weight model

Based on the above analysis, the recovery overhead key factors are: 1) RDD lineage length, 2) the complexity of operation, 3) RDD computation cost.

Definition 8 - depth of RDD. Depth(RDDij) denotes as the RDD lineage length, which is the layer number for RDD_{ij} in the directed acyclic graph (DAG). Assumed the first layer RDD, depth is defined as 1, and the depth of the last RDD is the maximum depth m of directed acyclic graph (DAG).

Assume two RDD_{ix} and RDD_{iy} have different depths, the depth as $\mathrm{Depth}(\mathrm{RDD}_{ix})$, $\mathrm{Depth}(\mathrm{RDD}_{iy})$, respectively. If $\mathrm{Depth}(\mathrm{RDD}_{ix})>\mathrm{Depth}(\mathrm{RDD}_{iy})$, then the lineage of RDD_{ix} is longer than that of RDD_{iy}.

Definition 9 -the operation complexity. RDD operation divides into two kinds, narrow dependency and wide dependency. Recovery overhead of wide dependency is larger, and is relevant to the partitions number. Denote the operation complexity as $\mathrm{OC}_{\mathrm{RDD}_{ij}}$, and RDD_{ij} has k partitions. When the RDD_{ij} operation is narrow dependency, the operation complexity is defined as 1. On the contrary, wide dependency occurs, when the operation complexity is defined as k.

Definition 10 - computation overhead. According to definition, PT_ijk computation cost requires a comprehensive assessment of the data acquisition cost, the data processing cost, and the evaluation of algorithms. It is difficult to predict it. But we could easily get the start time and completion time of RDD_{ij}. So the computation cost can be expressed as:

$$\mathrm{Cost}_{\mathrm{RDD}_{ij}} = \mathrm{FT}_{\mathrm{RDD}_{ij}} - \mathrm{ST}_{\mathrm{RDD}_{ij}} \,. \tag{16}$$

Definition 11 - RDD weight. The weight of RDD is expressed as follows:

$$\mathrm{CR}_{\mathrm{RDD}_{ij}} = \alpha \times \mathrm{Depth}_{\mathrm{RDD}_{ij}} + \beta \times \mathrm{OC}_{\mathrm{RDD}_{ij}} + \gamma \times \mathrm{Cost}_{\mathrm{RDD}_{ij}} \,. \tag{17}$$

Here, $0 \le \alpha, \beta, \gamma \le 1$, and $\alpha + \beta + \gamma = 1$. When $\alpha = 1$, the weight is determined by RDD lineage depth. When $\beta = 1$, the key is up to the operation complexity. When $\gamma = 1$, the key is decided by computation overhead.

4. Optimization processes

Before performing the task, traverse the DAG of the task. Then get the operation and the properties of each RDD. After analyze the DAG and the current implementation of the progress, calculate the RDD weight.

According to the RDD weight, select the RDDs as checkpoint to perform. Checkpoint time began from the first generation of RDD to the latest generation of RDD. During the task execution, comparison of the new generation of multiple RDD, select the largest RDD as the checkpoint.

After downtime, the spark performs the recovery operation, and recovers from the latest checkpoint RDD. The latest checkpoint is read into memory, thereby reducing the recovery and execution overhead. When a RDD need recover, re-execute the lineage and recovery through its parent node.

To complete the processing tasks, the recovery strategy steps are:

1) Choose free worker, if the current has no idle worker, waiting for the worker assignment.

2) Receive a checkpoint sequence.

3) It has been lost RDD sequence.

4) Determine the need to use what RDD by lineage.

5) If the operation is of wide dependency, or lost the RDD all partitions, you need to read the RDD all partitions to memory.

6) If it is the loss of RDD partition, and the lineage without wide dependency, only need to read the lost partition checkpoint into memory.

7) Read the checkpoint into memory, and compute by the lineage.

5. Experimental and evaluation

5.1. Experiment environment

This section will be compared and evaluated by experiments, which verify the validity of the checkpoint automatic selection algorithm and the recovery algorithm.

The experiments perform with a master server and eight computing nodes as the master and Hadoop namenodes of Spark, and the nodes configuration is shown in Table 1.

5.2. The optimization strategy

5.2.1. Different failure rate. Figure 1 indicates that the iteration time is 1–10, when the failure rate is $fr = 0.375$ and 0.5, the algorithm uses different data sets, and compared the failure recovery algorithm.

As shown in Fig. 1, with the increase of failure rate, task execution time also increases. This is because of the failure rate is high, which means the more node failure. Therefore, in order to restore the RDD, it is need more time overhead to recalculate the corresponding. Comparison of different data sets, and Web-Google Wiki-Talk under different algorithms, Wiki-Talk has bigger time overhead. This is

due to the size difference computation.

Table 1. Configuration parameters

Parameters	Values
CPU	Intel CORE i7/2.2 GHz
RAM	1 GB
Hard Disk	200 GB/SATA3.0
OS	CentOS 6.4
Spark	Apache Spark 1.4.1
Hadoop	Apache Hadoop 2.6
Scala	Scala-2.10.4
JDK	OpenJDK 1.8.0 25

Comparing the number of iteration, it can be seen that when the iteration number is 1, the task execution time of failure recovery algorithm or system recovery algorithm is basically the same. This is because the check pointing algorithm is set to complete the checkpoint, so the failure recovery algorithm does not affect task. And with the increasing in the number of iterations, the execution speed of failure recovery is obviously better than that of system recovery algorithm. The system recovery algorithm recomputed RDD from the beginning. Then the bigger the iteration time is, the longer recovery time is. Therefore, if the number of iterations is small, the user can use the lineage to recover and does not need to set up checkpoints to improve the efficiency.

5.2.2. Accelerate the recovery ratio. Figure 2 indicates that the algorithm uses different data sets, the average recovery time and recovery speedup ratio of the failure recovery algorithm.

Figure 2 shows that the comparison of different iterations, with the increasing in the number of iterations, the original Spark by lineage to recover the time cost is larger. When the node fails, it will result in the loss of RDD partitions. Spark task will perform these tasks concurrent on other machines. The task reread the input data, and reconstruction RDD based on the lineage. The longer the iteration and computing time is, the greater recovery time overhead is. Comparing the different data sets, the recovery of Wiki-Talk speedup more, because it needs more computation, The execution time is longer, so the overhead of the recovery is larger, and the recovery cost can be significantly reduced by using checkpoint recovery algorithm.

Then there were compared Fig. 1 and Fig. 2 comprehensively, analyzed task execution time, recovery time and recovery speedup ratio. Although the algorithm may increases the amount of time overhead, compared to the traditional policy uncertainty and even abnormal risk, the extra overhead is worth it. On the basis of automatic checkpoint algorithm, failure recovery algorithm considers not only the RDD lineage length, but also the computation cost and operation complexity. The greater the weight of RDD is, the higher checkpoint priority is. It will make the task to minimum the overall computational cost, so as to improve the recovery efficiency

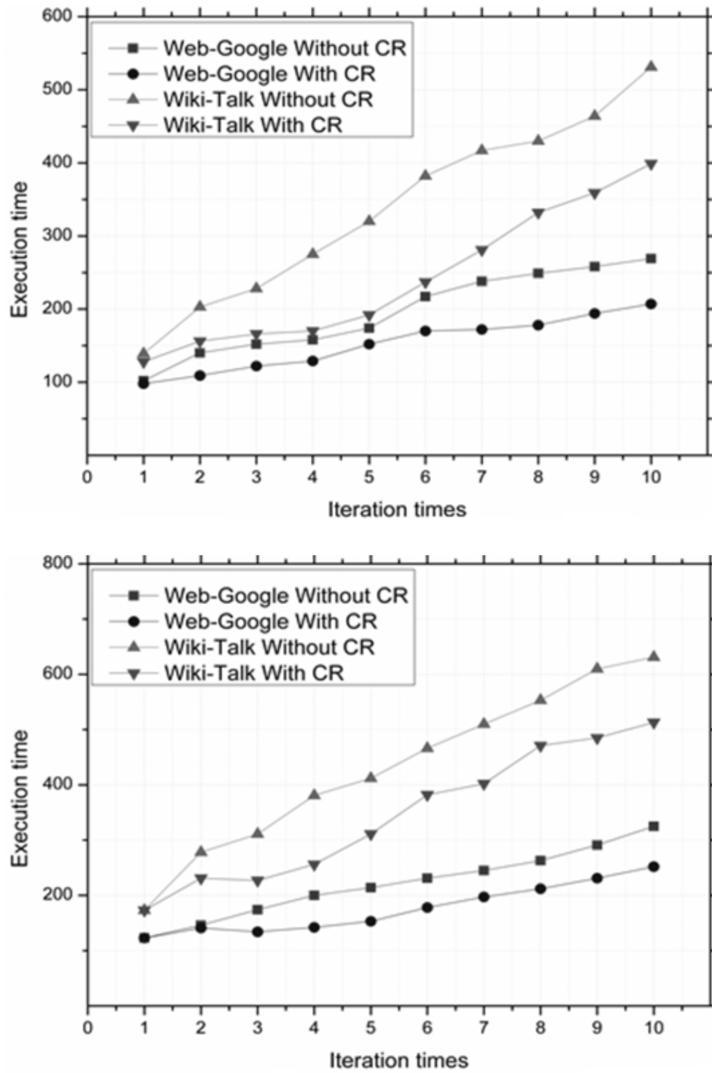

Fig. 1. Execution time (s) of CR(s): up–$fr = 0.375$, bottom–$fr = 0.5$

of tasks. Therefore, checkpoint selection algorithm cannot significantly affect the performance of the Spark system under the condition of enhancing system stability and reliability.

6. Conclusion

The traditional network failure recovery strategies set the checkpoints depending on the experiences of the programmers, which may cause larger execution time and

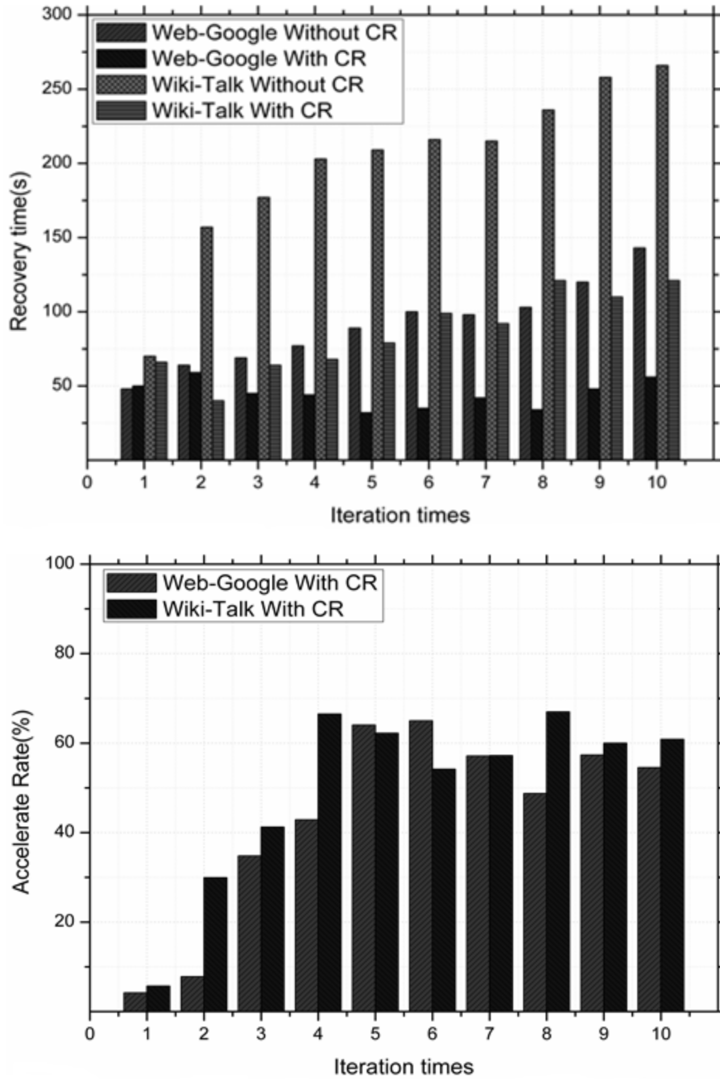

Fig. 2. Average recovery efficiency of CR: up–recovery time (s), bottom–accelerate rate (%)

lower efficiency. To address the issue, we analyzed task execution mechanism, and established the task execution efficiency model. Then put forward the RDD weight model, which provided a theoretical basis for the strategy presented. Experiments were conducted with different data sets and failure rate. And the results demonstrated that the strategy can improve the recovery efficiency and the utilization of system resources at the same time. Especially with the rising in large data analysis, network optimization has become increasingly prominent, and network failure recovery is one of the key problems to be addressed.

However, due to the limited capacities, the research should further be conducted in the future. Our work will mainly focus in the following aspects:

(1) Analyze in multiple checkpoints failure and different recovery strategy for network efficiency.

(2) With the decreasing cost of network devices, using new medium to enhance recovery efficiency becomes feasible.

(3) By constructing a multi-level failure tolerance network to improve the performance of the system is a future research direction.

References

[1] Z. Zheng, P. Wang, J. Liu, S. Sun: *Real-time big data processing framework: challenges and solutions.* Applied Mathematics & Information Sciences *9* (2015), No. 6, 3169–3190.

[2] O. Shamir, N. Srebro, T. Zhang: *Communication efficient distributed optimization using an approximate Newton-type method.* International conference on machine learning (ICML), 21–26 June 2014, Beijing, China (2014), 1000–1008, arXiv:1312.7853v4 [cs.LG] 13 May 2014.

[3] E. Yildirim, T. Kosar: *End-to-end data-flow parallelism for throughput optimization in high-speed networks.* Journal of Grid Computing *10* (2012), No. 3, 395–418.

[4] Z. Kozhirbayev, R. O. Sinnott: *A performance comparison of container-based technologies for the cloud.* Future Generation Computer Systems *68* (2017), 175–182.

[5] Z. Xiao, W. Song, Q. Chen: *Dynamic resource allocation using virtual machines for cloud computing environment.* IEEE Transactions on Parallel and Distributed Systems *24* (2013), No. 6, 1107–1117.

[6] H. Shen, G. Liu: *An efficient and trustworthy resource sharing platform for collaborative cloud computing.* IEEE Transactions on Parallel and Distributed Systems *25* (2014), No. 4, 862–875.

[7] I. P. Egwutuoha, D. Levy, B. Selic, S. Chen: *A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems.* The Journal of Supercomputing *65* (2013), No. 3, 1302–1326.

[8] I. Cores, G. Rodríguez, M. J. Martín, P. González, R. R. Osorio: *Improving scalability of application-level checkpoint-recovery by reducing checkpoint sizes.* New Generation Computing *31* (2013), No. 3, 163–185.

[9] S. H. Lim, S. Lee, B. H. Lee, S. Lee, H. W. Lee: *Stochastic method for power-aware checkpoint intervals in wireless environments: Theory and application.* Journal of Industrial and Management Optimization *8* (2012), No. 4, 969–986.

[10] K. B. Ferreira, R. Riesen, P. Bridges, D. Arnold, R. Brightwell: *Accelerating incremental checkpointing for extreme-scale computing.* Future Generation Computer Systems *30*, (2014), 66–77.

Received July 12, 2017