A Quantitative Analysis of Fault Tolerance Mechanisms for Parallel Machine Learning Systems with Parameter Servers

Mingxi LI
University of Tsukuba
Tennoudai 1–1–1, Tsukuba, Japan
rei-meigi@aist.go.jp

Yusuke TANIMURA
National Institute of Advanced Industrial Science and Technology
Umezono 1–1–1, Tsukuba, Japan
yusuke.tanimura@aist.go.jp

Hidemoto NAKADA
National Institute of Advanced Industrial Science and Technology
Umezono 1–1–1, Tsukuba, Japan
hide-nakada@aist.go.jp

ABSTRACT

Parallel computation is essential for machine learning systems that handle large amount of data for training. One popular form of parallel machine learning is the one called 'Data parallel' which use large number of parameter servers that manage computational workers. Fault tolerance is always a crucial issue on large scale computation systems in general, and parameter server based machine learning systems are no exception. However, there are many discussions on the fault tolerance of large scale computation systems in general, there are no discussions on parallel machine learning systems, in spite of their unique characteristics. In this paper, we discuss the fault tolerance of parallel machine learning systems which use parameter servers that provides extra redundancy to the system and could double as the checkpoint server. We also quantitatively evaluate several fault tolerance method using parallel environment simulator SimGrid, and demonstrate the effectiveness of the proposed method.

CCS Concepts

•**Computing methodologies** → *Distributed computing methodologies;* Machine learning; •**Computer systems organization** → *Availability;*

Keywords

Parameter Server, Fault Tolerance, Machine Learning

1. INTRODUCTION

Since modern large-scale machine learning systems require huge amount of computational power to handle huge amount of data, it is essential to parallelize them to achieve practical performance. There are two techniques to parallelize machine learning systems; namely, **data parallel method** where multiple machine learning modules learn independently synchronizing the model parameters periodically, and **model parallel method** where multiple threads / processes update model parameters inside a single machine learning module. We focus on the data parallel method that uses **parameter servers** for synchronizing model parameters.

Data parallel machine learning systems tend to have hundreds of computation nodes, and the larger a system is, the larger the possibility of fault occurs. Hence, the systems require some kind of fault tolerance mechanism. Fault tolerance mechanism is well investigated in High-performance computing area.[13]

Machine learning systems with parameter servers have unique characteristics, in terms of data redundancy. Where, the parameters are basically shared with all the computational servers and the parameter servers, proving data redundancy that could be leveraged for fault recovery. However, such kinds of mechanisms are independently deployed in practical machine leaning systems, theoretical and quantitative analysis has not been sufficiently studied.

We discuss fault tolerance mechanism that leverage data redundancy which is unique to the parameter server based systems, and perform quantitative analysis using SimGrid [6], the distributed processing environment simulator. We compared with conventional periodic checkpointing mechanism with two methods that leverage data redundancy in parameter server based systems.

The contributions of this paper are the followings; 1) formalize two methods that is specific to parameter server based systems, 2) quantitatively evaluate their performance impact comparing conventional mechanism.

The rest of paper is organized as follows. Section 2 gives background of this work. In section 3, we formalize the two fault tolerance methods for parameter server based systems. Section 4 gives simulation settings and the result of the simulation. we present related work in section 5 and summary and future work in section 6.

IMCOM '17, January 05-07, 2017, Beppu, Japan
© 2017 ACM. ISBN 978-1-4503-4888-1/17/01. . . $15.00
DOI: http://dx.doi.org/10.1145/3022227.3022295

2. BACKGROUND

2.1 Parameter Server Based Data Parallel Machine Learning Systems

For cutting-edge machine learning researches, large-scale parallel systems are essential to handle huge amount of data. For example, in [10], authors used 1,000 nodes with 16 processing cores each for three days, to train their model.

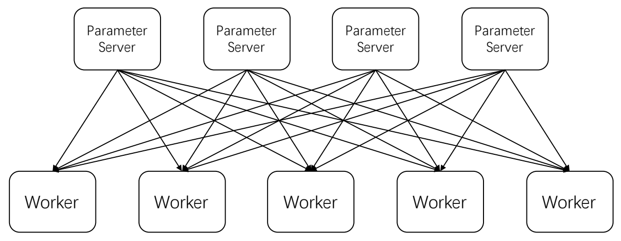

Figure 1: Typical Structure of a Parallel Machine Learning System with Parameter Servers.

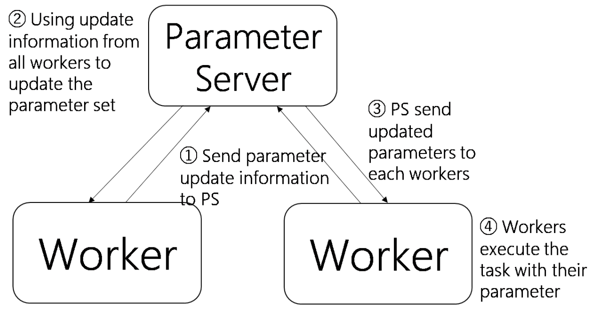
② Using update information from all workers to update the parameter set
Parameter Server
③ PS send updated parameters to each workers
① Send parameter update information to PS
Worker
Worker
④ Workers execute the task with their parameter

Figure 2: Communication between a Parameter Server and Computation Workers.

Generally Speaking, machine learning systems could be parallelized in two ways. One way is called **Model Parallel**, where one machine learning module processes data in parallel fashion. The other way is called **Data Parallel**, where multiple machine learning modules independently learn on different subset of dataset. Each module independently updates its model parameters periodically synching the parameter each other.

The data parallel method is used in [10][8]. The servers that are responsible for parameter synchronization are called **Parameter Servers**[1][11]. We have implemented master-worker based parameter server prototype [12] to speed up restricted Baysean-network based model called BESOM[9].

Figure. 1 shows a parallel machine learning system with parameter servers, which consists of parameter servers and machine learning workers. As shown in the figure, the system forms bipartite graph; each parameter server talks to all the workers, and each worker talks to parameter server.

Parameter servers manage latest shared parameter set as a whole. Each parameter server is responsible for a specific portion of the parameter set. The reason to have large number of parameter servers is to avoid communication bottleneck.

The workers periodically send parameter update information to the parameter servers and receive latest parameters from the parameter servers. Parameter servers receive parameter update information and update its internal parameter set, and send them to the workers, as shown in Figure. 2.

Figure. 3 demonstrates network communication among parameter servers and workers, along the time axis. All the parameter servers communicate all the workers. There are no direct communication within parameter servers and

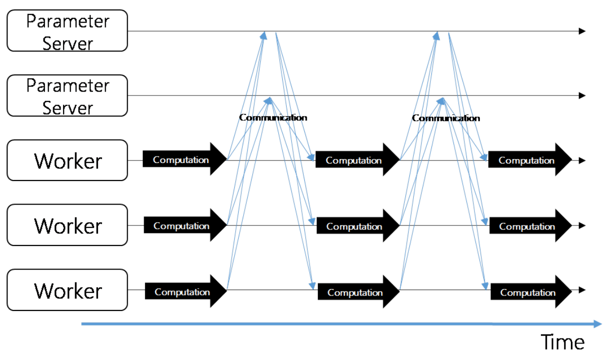

Figure 3: Time line of Communication among Parameter Servers and Computation Workers.

workers. The parallel machine learning process could be considered as a BSP(Bulk Synchronous Parallel[14]). Workers synchronize each other via parameter servers.

In large-scale machine learning system, it is common to have hundreds of parameter servers, to avoid network congestion on parameter servers. We divide parameters into chunks and assign each of them to a parameter server. Since parameters could be updated independently, no communication required among parameter servers.

2.2 SimGrid: a Distributed Environment Simulator

SimGrid[6][2] is a simulation framework for distributed parallel applications. Simulators for parallel systems could be divided into four categories; experimental platforms, emulators, network emulators, and application simulators. SimGrid is categorized into application simulator.

SimGrid is based on a discrete event simulation; it does not perform any real computation / communication. It just estimates times to perform computation / communication based on given parameters and records events like 'start / end of computation / communication'. The advantage of this type of simulator is that the simulation cost is relatively small. Even with single node computer, SimGrid can handle several thousands of communicating nodes.

To simulate a distributed system in SimGrid, users have to describe platform description and deployment description in XML, and the simulation code in C++. We show some samples here. [1] Figure. 4 shows an example of platform description. The platform described here is a quite simple one; `host1` and `host1` are connected with `link1`. Hosts definitions include computational power and Link definition includes network bandwidth and latency. The description also defines routing information.

Figure. 5 shows an example of deployment description for SimGrid. This description denotes that `host1` executes process `master` and `host2` executes process `worker`, specifying input arguments for the processes.

Figure. 6 shows an example of process code. Typical SimGrid process repeats followings; it receives message from some other nodes, performs some computation based on the received message, and sends out messages to other nodes.

[1]The samples below are taken from http://simgrid.gforge.inria.fr/tutorials/simgrid-use-101.pdf

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
 <AS id="AS0" routing="Full">
  <host name="host1" power="1E8"/>
  <host name="host2" power="1E8"/>
  <link name="link1" bandwidth="1E6"
                     latency="1E-2" />
  <route src="host1" dst="host2">
    <link:ctn id="link1"/>
  </route>
...
 </AS>
</platform>
```

Figure 4: An Example of Platform Description in SimGrid.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM
"http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
 <!-- The master process -->
 <process host="host1" function="master">
  <argument value="10"/><!--argv[1]:#tasks-->
  <argument value="1"/><!--argv[2]:#workers-->
 </process>
 <!-- The workers -->
 <process host="host2" function="worker">
  <argument value="0"/>
 </process>
</platform>
```

Figure 5: An Example of Deployment Description in SimGrid.

SimGrid allows defining high level computer network components, like “Cluster”, adding to the basic components like “hosts” or “links”. This feature greatly helps to describe large-scaled distributed systems.

3. FAULT TOLERANCE METHOD FOR PARAMETER SERVER BASED SYSTEMS

Faults Tolerance mechanisms on distributed systems are extensively studied mainly in high performance computing area[13]. While distributed systems could have numerous kinds of faults, such as fault of nodes or fault of network, we only study the fault of worker nodes in this paper. This is because of the fact that the worker nodes dominate the system in numbers, and hence most likely to have fault.

In this paper, we assume the existence of backup nodes which could be used to replace the failed worker nodes. When a worker node has a fault, the system will start up the backup node to inherit the status of the worker node which had the fault and continue the interrupted task.

In this paper, we studied 3 kinds of fault tolerance mechanisms.

3.1 Using checkpoint recovery

The fault tolerance mechanism which uses checkpoint files is commonly-used by distributed systems in general. All the nodes save their status as checkpoint periodically in coordination. and when a node got a fault, All the nodes revert to the state when the checkpoint had taken with the failed node replaced by the substitution node. We call this mechanism **CKPT**.

Checkpoint method could be implemented in two ways. One is in the application layer, and the other is in the system layer, such as MPI. The former put extra implementation burden on the application programmer, but tends to be more effective since the application programmer knows exactly which portion of the data should be saved to recover the state, and proper timing to take checkpoint where the state footprint is minimum. The latter approach is easier for the application programmer who do not have to do extra work to take checkpoint, but tends to be less effective. Since the system does not have no idea where is essential portion, it simply dumps whole the process memory area. There are several MPI implementations that supports automatic checkpoint/restart that require no modification for the application programs.[4][5]

```
int worker(int argc, char *argv[ ]) {
  msg_task_t task; int errcode;
  int id = atoi(argv[1]);
  char mailbox[80];
  sprintf(mailbox,"worker-%d",id);
  while(1) {
    errcode = MSG_task_receive(&task, mailbox);
    xbt_assert(errcode ==
               MSG_OK, "MSG_task_get failed");
    if (!strcmp(MSG_task_get_name(task),
                "finalize")) {
      MSG_task_destroy(task);
      break;
    }
    XBT INFO("Processing '%s'",
             MSG_task_get_name(task));
    MSG_task_execute(task);
    XBT INFO("'%s' done",
             MSG_task_get_name(task));
    MSG_task_destroy(task);
  }
  XBT INFO("I'm done. See you!");
  return 0;
}
```

Figure 6: An Example of Simulation code in SimGrid.

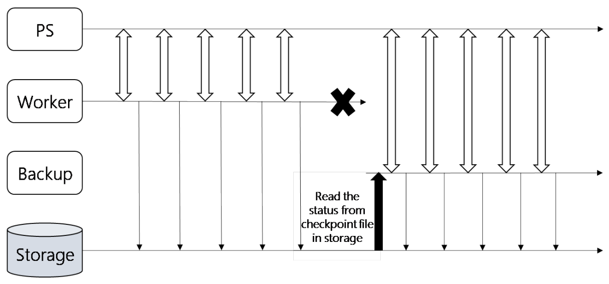

Figure 7: The mechanism uses checkpoint (CKPT method).

While CKPT could be used any kind of distributed systems, it suffered from several overheads. One overhead comes from coordination of nodes on checkpoint. To have globally consistent checkpoint, costly coordination among all the nodes is essential.

Saving checkpoint file itself also causes overhead. Saving checkpoint inherently involves storage I/O access. Therefore, saving the checkpoint frequently will increase the learning time significantly. On the other hand, if the interval of checkpoints is too long, the damage of a fault will become large, since all the nodes have to roll back the latest checkpoint.

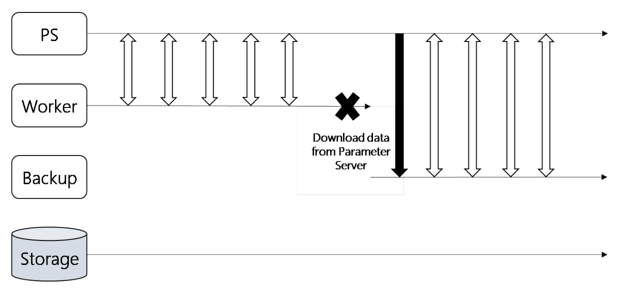

Figure 8: The mechanism gets data from Parameter Server (PS method).

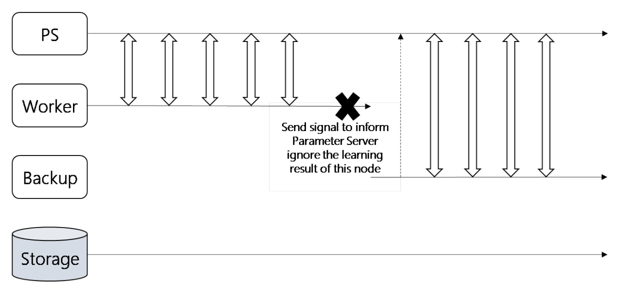
PS
Worker
Backup
Storage
Send signal to inform Parameter Server ignore the learning result of this node

Figure 9: The mechanism discards the failed task (IGNR method).

When the system detects a fault on a worker node, it will inform the fact to all the other worker nodes. All the worker nodes will discard their task and roll back to the status stored in the checkpoint file.

3.2 Getting data from Parameter Server

In most distributed parallel systems, duplication of information is not common. In other words, there is no information redundancy in the system. This is why thus, they have to use checkpoint to have information redundancy. In contrast, Parameter Server based parallel machine learning systems have high information redundancy, since basically all the worker nodes are synchronized periodically and the Parameter Servers reserve the same parameter information.

Here, we define a fault tolerance mechanism which uses Parameter Servers as checkpoint servers. We call the mechanism **PS**. Backup node will load the parameters from Parameter Servers to continue the interrupted job.

The detailed steps are shown in Figure. 8. System will start up the backup server after detecting the fault. Backup node retrieves latest parameters from Parameter Servers, and continues the work as a worker node. In this mechanism, the worker nodes which did not have faults do not have to modify their behavior, although they might have to wait long for the recovery of the backup node to catch up.

3.3 Discard the failed task

Parameter Server will receive update information from a large number of worker nodes to update the parameter. Since this process is stochastic, it is conceivable that losing one worker contribution in one iteration will not affect the whole learning process much. The following mechanism is based on this idea.

The detailed steps are described in Figure. 9. When a fault in a worker node is detected, Parameter Server will update the parameters without the update information from the worker node. During this time, system will start up the backup node. The backup node will receive lasted parameters from Parameter Servers and recover as a worker node in next learning repetition. We call this mechanism **IGNR**.

Compared to the previous two mechanisms, this mechanism does not involve I/O access, and hence, the recovery cost is expected to be the lowest in this three mechanisms. However, there is a risk that accuracy of learning will decrease, since some portion of parameter update information is lost.

4. QUANTITATIVE ANALYSIS WITH SIMULATION

We evaluated the three methods using a distributed environment simulator, called SimGrid. Simulators allow us to setup virtual evaluation environment arbitrary, making deep understanding of characteristics of the methods.

4.1 Interval between failures

The probability of computer failure is said to obey Poisson distribution. If a series of events obeys a Poisson distribution with parameter λ, the interval of the events obeys an exponential distribution with the parameter λ. The expected value of exponential distribution is $1/\lambda$, thus, the mean interval of failure is the reciprocal of λ. This mean interval is called MTBF(Mean time between failures). We determined the MTBF of worker nodes and derived the time when to fail using exponential distribution.

4.2 Fault Implementation with SimGrid

Processes in SimGrid takes one of the three states: data-sending, data-receiving, and task-executing. In this experiment, we only studied the failure during task execution, since it dominates the time spent.

We generate random numbers with exponential distribution and determine the time of failure before a worker executes a task. If the determined time is during the task executing, we make the failure happen. If not, the worker finishes normal task execution.

4.3 Simulation Setup

We setup a virtual distributed system with one Parameter Server cluster, 100 Worker clusters and a dedicated storage cluster for checkpoint files which is connected worker node clusters with independent dedicated network, for the evaluation. Each cluster has 100 nodes. There is a backbone link in the cluster. When a node communicates via a router, the information will go through the normal link between the backbone link and the node, and then reach the router via the backbone link. The structure of the system is shown in Figure. 10. Detailed parameters of nodes and learning model is shown in table 1.

We change the checkpoint size and the bandwidth to storage cluster in CKPT method, performed 30 simulations each which have 200 times of learning iterations on the three fault tolerance mechanisms, in 2 types of network environment (table 2,table 3). We logged elapsed time for whole learning and number of failures which happens during the simulation.

Table 1: Nodes and Model Settings

Item	Parameter
Time of 1 learning task executing	100s
Size of learning model	1GB
I/O speed of checkpoint storage	10GB/s
Starting time of nodes	5s
MTBF of 1 computer	3Years = 94608000s

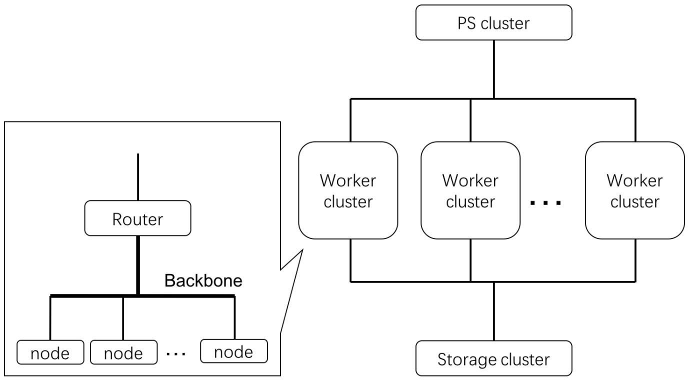

Figure 10: Configuration of the system.

4.4 Evaluation Results

Mean of elapsed time for each method is shown in Figure. 11 and Figure. 12, and the standard deviation of elapsed time is shown in Figure. 13 and Figure. 14.

We also logged the mean times of fault occurs with PS method, IGNR method, and CKPT method that checkpoint size is 4GB, bandwidth to storage is 0.5TB/s, simulated with the network environment table 2.

As shown in table 4, the mean times of fault occurrence is around three. The learning time is about 28700s for PS and IGNR method, therefore, the probability of failure during execution can be calculated with Poisson distribution which has parameter λ = 28700 / 94608000, k = 1. The derived probability is 0.03%, thus, for 10000 worker nodes, around 3 times of failure is expected. This ensured us that the simulation setting is correct and the failure mechanism described in 4.2 is working as expected.

CKPT method tends to have more fault than the other 2 methods. This is because CKPT method consume more computation time due to rollback on failure. More computation time makes possibility of fault higher.

As shown in Figure. 11 and Figure. 12, CKPT method took the longest. This is because, as the elapsed time that CKPT method with faults shows, the method not only spend time for writing checkpoint file repeatedly, but also waste computation time for loading checkpoint file, and for the data loss caused by roll back.

Both checkpoint size and the bandwidth to checkpoint storage affect the execution time. The overhead caused by checkpointing is proportional to the size of checkpoint. [2]

In addition, as shown in Figure. 13 and Figure. 14, the standard deviation of elapsed time for CKPT method is always larger than the other methods. This is because the cost of fault recovery depends on the run length from the last checkpoint and varies time to time.

[2]Note that the x-axis of Figure. 11 and Figure. 12 are logarithmic.

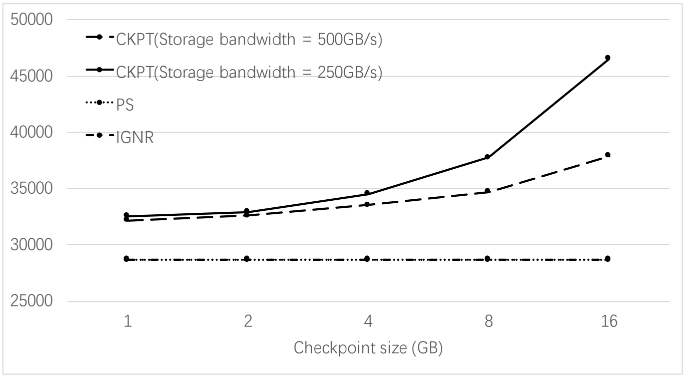

Figure 11: Mean time of learning(s) (with environment table 2).

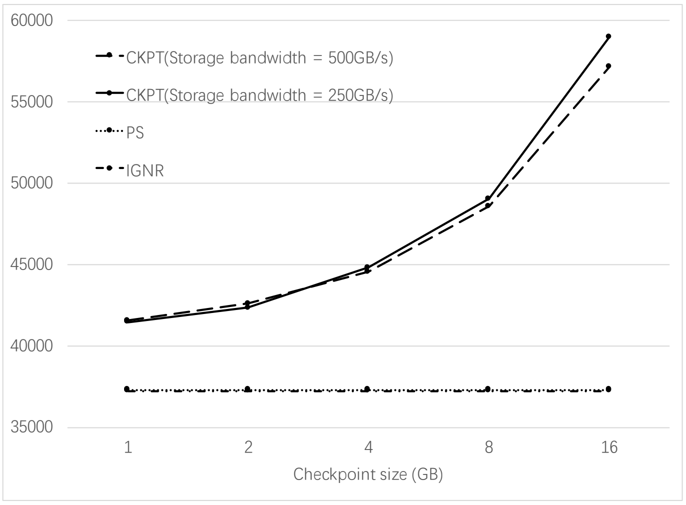

Figure 12: Mean time of learning(s) (with environment table 3).

As shown in the result, with IGNR method, the total execution time had not been affected by the failure. Furthermore, there are some cases that the learning time even decreases when the failure happened. This is because there are no 'recovery' procedure in this method; backup node effectively skips the rest of the task just informing Parameter Servers that the failure has been happened. No further data transfer is required. However, from the performance impact point of view this method is ideal, there might be negative impact on learning progress and accuracy of the learned model.

5. RELATED WORKS

There are several efforts on overhead analysis of fault tolerance mechanisms with simulation.

Arunagiri et al. analyzes checkpoint I/O overhead of the

Table 2: Network Environment of Simulation (1).

	Link	Bandwidth	Latency
Worker cluster	Link between node and backbone	125MB/s	50μs
	Backbone link	5GB/s	500μs
PS cluster	Link between node and backbone	5GB/s	500μs
	Backbone link	0.5TB/s	500μs
Storage cluster	Link between node and backbone	5GB/s	500μs
	Backbone link	0.5TB/s	500μs

Table 3: Network Environment of Simulation (2).

	Link	Bandwidth	Latency
Worker cluster	Link between node and backbone	125MB/s	50μs
	Backbone link	2.5GB/s	500μs
PS cluster	Link between node and backbone	2.5GB/s	500μs
	Backbone link	250GB/s	500μs
Storage cluster	Link between node and backbone	2.5GB/s	500μs
	Backbone link	250GB/s	500μs

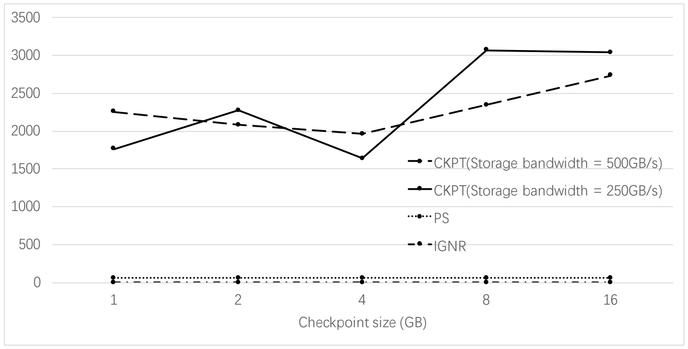

Figure 13: Standard deviation of learning time(s) (with environment table 2).

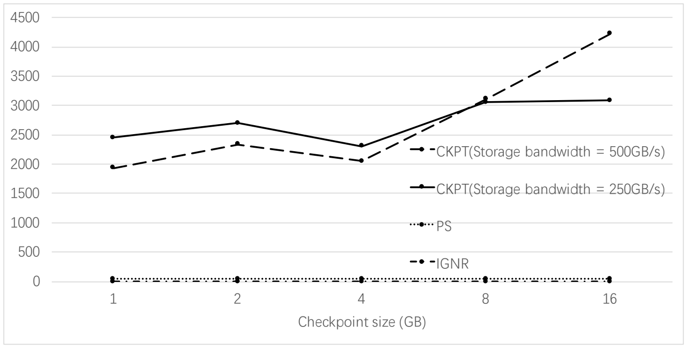

Figure 14: Standard deviation of learning time(s) (with environment table 3).

Table 4: Mean times of fault occurrence.

Mechanism	CKPT	PS	IGNR
times of fault occurrence	3.655	3.1	3.1

massively parallel processing (MPP) systems using simulator.[3] Their focus is in the low-level I/O traffic while we are interested in macroscopic behavior of the system.

Wang et al. use trace-driven simulation for worst-case analyze of coordinated checkpoint to find proper interval between checkpoints.[15] While their goal is rather close to ours, we are interested in parameter-server specific fault tolerant algorithms.

6. CONCLUSIONS AND FUTURE WORK

We have discussed fault tolerance methods that leverage unique characteristics of parameter server based parallel machine learning systems, and performed quantitative analysis with simulations. We proved that we could achieve fault tolerance with lower cost by leveraging the unique data redundancy provided by parameter server based systems, compared with conventional checkpoint-recover method. We have described two methods called PS and IGNR, and showed that IGNR is slightly better in terms of time cost, with some potential performance drawback in terms of learning accuracy. However, the time differences between this two methods is very small, considering the possibility that IGNR will delay the learning process, PS method maybe better in a practical learning.

Our future work includes the followings.

- In this paper we focused on worker node fault only and ignored parameter servers and network faults. parameter servers also have good chance to have failure and some parameter servers do implement fault tolerance through data replication. [7][11] We will simulate these behavior and evaluate the performance impacts.

- The IGNR method might delay whole learning process since it throws away once iteration of computation that is assigned to the failed node. We have to quantitatively evaluate the delay.

Acknowledgment

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO). This work was supported by JSPS KAKENHI Grant Number JP16K00116.

7. REFERENCES

[1] Parameter server: http://parameterserver.org/. Accessed: 2015-06-20.

[2] Simgrid: Versatile simulation of distributed systems: http://simgrid.gforge.inria.fr/index.php. Accessed: 2016-07-11.

[3] S. Arunagiri, J. T. Daly, and P. J. Teller. *Modeling and Analysis of Checkpoint I/O Operations*, pages 386–400. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[4] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. Mpich-v: Toward a scalable fault tolerant mpi for volatile nodes. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 29–29, Nov 2002.

[5] G. Bronevetsky, D. Marques, K. Pingali, and P. Stodghill. Automated application-level checkpointing of mpi programs. In *Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '03, pages 84–94, New York, NY, USA, 2003. ACM.

[6] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[7] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, Broomfield, CO, Oct. 2014. USENIX Association.

[8] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS 2012: Neural Information Processing Systems*, 2012.

[9] Y. Ichisugi and N. Takahashi. An efficient recognition algorithm for restricted bayesian networks. In *Proc. of 2015 International Joint Conference on Neural Networks (IJCNN 2015).*, 2015.

[10] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

[11] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO, Oct. 2014. USENIX Association.

[12] M. Li, Y. Tanimura, Y. Ichisugi, and H. Nakada. An implementation of the master-worker based parameter server and its application to besom (in japanese). In *IEICE technical report, vol. 115, no. 174, CPSY2015-33*, pages 179–184, 2015.

[13] I. P.Egwutuoha, D. Levy, B. Selic, and S. Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. In *The Journal of Supercomputing*, pages 1302–1326, Sept. 2013.

[14] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, Aug. 1990.

[15] Y. M. Wang and W. K. Fuchs. Lazy checkpoint coordination for bounding rollback propagation. In *Reliable Distributed Systems, 1993. Proceedings., 12th Symposium on*, pages 78–85, Oct 1993.