Coordinated Checkpointing using Vector Timestamp in Grid Computing

Taichi Jinno, Tokimasa Kamiya, Motoyasu Nagata
Osaka Kyoiku University
4-698-1, Asahigaoka, Kashiwara, Osaka, 582-8582, Japan.
TEL / FAX +81-729-78-3672
nagata@cc.osaka-kyoiku.ac.jp

Abstract ***In grid computing, system recovery is carried out using checkpoints recorded at each nodes. The resource manager must recover system with keeping global consistency to prevent Domino effect. Currently, coordinated checkpointing is widely used in which all processes can be synchronized. Considering overhead due to synchronization, we will present a coordinated checkpoint protocol using vector timestamp to reduce overhead. Our proposed protocol aims to reduce idle time of every process by grasping occurred event numbers. We will also evaluate performance of the proposed protocol. Experiment was carried out for parallel computation of eight nodes. As the result of the experiment, we obtained reduction of overhead time with 55 percentages in average at each processes. Thus, we showed effectiveness of our proposed protocol for scalable grid computing.***

Keywords: Grid Computing, Checkpointing, Vector Timestamp, Parallel Processing, Rollback-Recovery

1 Introduction

Along with spread of grid computing technology, scalable computations on geographically distributed computers have become capable. So far, most applications of the grid computing are scientific technology fields. However, from viewpoints of highly flexible processing capability and potentiality of the grid computing, many enterprises began to focus on the grid computation. For more widely spread of the grid, high reliability to cope with faults about processor, memory and network is being required. One of the technologies to cope with system fault, there exists checkpointing at each computing nodes. In checkpointing, state and address space of the process are recorded, which is utilized in the fault occurrence to restart grid computation.

As for design of checkpointing, coordinated checkpointing synchronizing between processes became popular method due to its simplicity rather than asynchronous checkpointing. In early Condor which is a grid job scheduling system, the coordinated checkpointing was implemented in a checkpoint library. However, the coordinated checkpointing became unused. As the grid computing system must be expanded according to change of computer environment, the overhead increases at synchronization in checkpointing. Furthermore, different specifications of each computing nodes and communication delay influences the overhead at synchronization.

This paper presents a coordinated checkpointing using vector timestamp to reduce blocked execution time of the process by grasping occurred event numbers. Our proposal

aims to reduce overhead by synchronization in checkpointing.

We also evaluate performance of our proposed coordinated checkpointing using vector timestamp. Experiment is carried out to compare the proposed protocol with traditional coordinated checkpoint protocol in distributed system environment.

Organization of this paper is described. Section 3 will describe traditional coordinated checkpointing and its problem. Section 4 will propose coordinated checkpointing using vector timestamp. Section 5 will report performance evaluation of our proposed protocol by executing pseudo grid computation. Section 6 concludes.

2 System Model and Definitions

Let us consider grid computing $\mathcal{G}$ consisting of n computing processes and a process playing role of coordinator. In addition, We assume sending message, receiving message and checkpointing as three kinds of events occuring at each processes while checkpointing.

We also use the following notations (see Table 1).

Table 1: Definitions of Notatios

Notatio	Definition
P	Set of processes executing grid computation
$p_i \in P (i = 1, 2, \ldots, n)$	n processes distributed at each nodes
$p_c \in P$	Process playing role of coordinator
$CP(s) \subseteq P (s = 1, 2, 3, \ldots)$	Set of processes executing the s-th checkpointing
$CPF(s) \subseteq P (s = 1, 2, 3, \ldots)$	Set of processes which have finished the s-th checkpointing
$send_i(m)$	Event that p_i sends message m
$receive_i(m)$	Event that p_i receives message m
$send_c(m)$	Event that coordinator p_c sends message m
$receive_c(m)$	Event that coordinator p_c receives message m
$checkpoint_i(s)$	Event that p_i executes the s-th checkpointing
cr	$CHECKPOINT\_REQUEST$ message
cf	$CHECKPOINT\_FINISH$ message
cd	$CHECKPOINT\_DONE$ message

3 Coordinated Checkpointing

3.1 Protocol

1. The coordinator process p_c requests checkpoint to each process p_i, : $send_c(cr) \rightarrow receive_i(cr)$.
2. The process p_i begins checkpointing after receiving cr : $receive_i(cr) \rightarrow checkpoint_i(s)$ holds where $p_i \in CP(s)$.
3. Process p_i sends cf to the coordinator after recording checkpoint: $send_i(cf) \rightarrow receive_c(cf)$ and $\neg(p_i \in CP(s)) \wedge (p_i \in CPF(s))$.

4. Coordinator sends cd to each processes when $(CP(s) = \emptyset) \wedge (CPF(s) = P)$ is satisfied, $send_{\mathrm{c}}(cd) \rightarrow receive_i(cd)$.

5. Once $\neg(p_i \in CPF(s))$ is satisfied, blocked processing restarts.

3.2 Problem of Coordinated Checkpointing

The coordinated checkpoint is the most fundamental checkpoint protocol in the grid computing. However, there exists bottleneck of the checkpointing corresponding to time interval from step 3 to step 4 in the above coordinated checkpointing protocol. During time interval from the time that all the n processes finished checkpointing to the time that the coordinator receives the last message cf, that is $(CP(s) = \emptyset) \wedge (CPF(s) = P)$ is satisfied, each processes are blocked processing. This is the reason that overhead occurs in the coordinated checkpointing protocol.

4 Coordinated Checkpointing using Vector Timestamp

We propose a coordinated checkpointing using vector timestamp. We introduce the vector timestamp to distribute sending of $CHECKPOINT\_DONE$ message by sharing information of occurrence number of events at every process.

4.1 Vector Timestamp

Every process p_i has vector timestamp VT_i of length $n+1$ and holds a message vm_i in which VT_i is attached. The vector timestamp VT_i has the following properties.

- $a \rightarrow b \Leftrightarrow VT[a] < VT[b]$
- $VT_i[i]$ stands for occurrence number of events at process p_i.
- For $VT_i[j] = k$, process p_i recognizes that k events occurred at process p_j.

Once an event occurs at process p_i, value of $VT_i[i]$ is incremented by one. When process p_i sends a message, process p_j can know occurence number of events at p_i by the message vm_i in which vector timestamp VT is attached. At this time, process p_j set $VT_j[k]$ as $\max(VT_j[k], vm_i[k])$.

4.2 Proposed Protocol

Coordinator process p_{c} holds a message vm_{c} in which VT_c is attached.

1. The coordinator process p_{c} executes `sendAction()` and requires checkpointing to each process p_i. $send_{\mathrm{c}}(vm_{\mathrm{c}}) \rightarrow receive_i(vm_{\mathrm{c}})$

2. Once process p_i receives message vm_{c}, process p_i executes `receiveAction()`.

3. Process p_i executes `checkpointAction()` and starts $checkpoint_i(s)$. $receive_i(vm_{\mathrm{c}}) \rightarrow checkpoint_i(s)$ where $p_i \in CP(s)$.

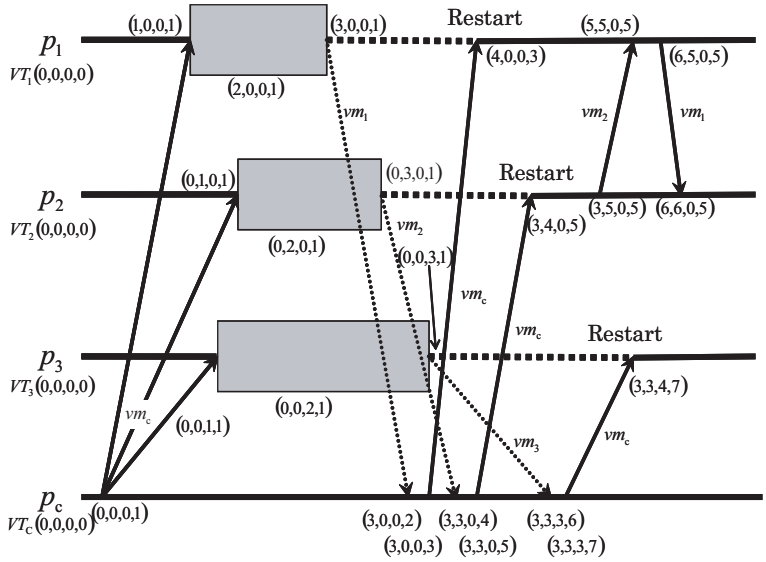

Figure 1: 3 processes of coordinated checkpointing using vector timestamp

4. After process p_i records checkpointing, process p_i executes `sendAction()` and sends message vm_i to the coordinator. $send_i(vm_c) \rightarrow receive_c(vm_c)$ and $\neg(p_i \in CP(s)) \wedge (p_i \in CPF(s))$.

5. As soon as the coordinator executes `receiveAction()`, the coordinator sends message vm_c to the process p_i. $send_c(vm_c) \rightarrow receive_i(vm_c)$

6. Once process p_i receives message vm_c by executing `receiveAction()`, $VT_i[i] = 4$ is satisfied.

7. Process p_i restarts blocked computing. At this time, $send_i(vm_i) \rightarrow receive_k(vm_i)$ is permitted fro processes p_k where $VT_i[k] = 3$.

8. Once $VT_c[4] = 2n + 1$ is satisfied, the s-th checkpointing ends.

Process p_i becomes member of $CPF(s)$, after process p_i receives checkpoint done message. At this time, as shown in Figure 1, $VT_i[i] = 4$ is clear. For remaining member p_k belonging to $CPF(s)$, $VT_i[k] = 3$ is satisfied. Thus, as process p_i can know event numbers occurring at each processes, process p_i has capability to grasp which processes are included in $CPF(s)$. Therefore, process p_i is permitted to restart processing.

Vector Timestamp Action

```
sendAction() {
    VT[myId] ++;
}

receiveAction(int *vm) {
    for (i = 0; i < N; i ++)
        VT[i] = max( VT[i], (vm + i) );
    VT[myId] ++;
}

checkpointAction() {
    VT[myId] ++;
}
```

4.3 The Recovery Line of Proposed Protocol

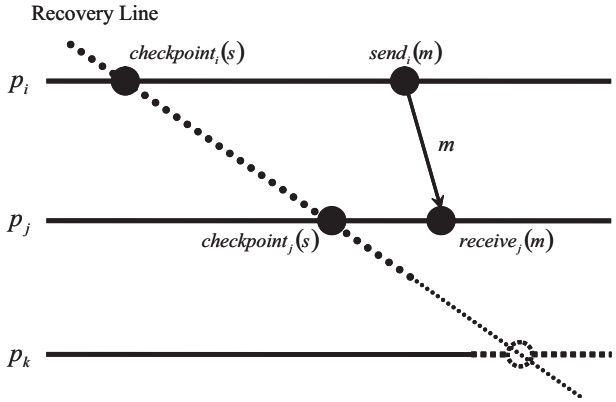

Figure 2: The recovery line of coordinated checkpointing using vector timestamp

The recovery line generated by the coordinated checkpointing using vector timestamp is globally consistent. Because, message communication after the restart can be done between processes satisfying $p_i \in CPF(s)$. Therefore, there exists no message which crosses the recovery line (see Figure 2).

5 Performance Evaluation

We will evaluate performance of our proposed coordinated checkpointing using vector timestamp. Our experiment aims to observe overhead time between our proposed protocol and traditional coordinated checkpointing protocol without vector timestamp. We expermented using two nodes, four nodes and eight nodes, respectively. For performance evaluation, we programmed a software using checkpoint library ckpt developed in Wisconsin University. This library records address space, signal status, and executing environment of all the processes. To recover a system from the fault immediately, the checkpoint file is saved into directory that has been mounted by NFS on node 10. In addition, node 9 played a role as coordinator.

5.1 Evaluation Environment

We implemented pseudo grid computing system of 2, 4 and 8 computing nodes where each processes repeats on writing memory of 128 MB for two times and sending message data of 64 B to the another nodes in random manner. The coordinator sends $CHECKPOINT\_REQUEST$ to each nodes with time interval of 120 seconds. We experimented two kinds of the checkpoint protocols; our proposed coordinated checkpointing protocol with vector timestamp and traditional coordinated checkpointing protocol without vector timestamp.

5.2 Experimental Result

We obtained observation results of overhead time at each process and ratio of computing time of each process to computing time of overall grid computing.

5.2.1 Overhead Time according to Processes

Because machine specifications are heterogeneous, there exists difference in overhead time between processes. Table 2 shows overhead time in each process when grid computing system of 8 computing nodes is working.

Table 2: Overhead time of 8 nodes

	node 1	node 2	node 3	node 4	
without Vector Timestamp (s)	45.5	37.3	2.1	4.9	
using Vector Timestamp (s)	24.5	23.6	0.4	0.8	
Reduceing ratio of Overhead	46.1%	36.7%	82.1%	84.3%	
including Ratio of process execution time	15.9%	9.4%	1.8%	4.5%	
	node 5	node 6	node 7	node 8	Average
without Vector Timestamp (s)	38.6	36.7	38.4	36.5	30.0
using Vector Timestamp (s)	16.4	13.7	12.3	15.8	13.4
Reduceing ratio of Overhead	57.5%	62.7%	67.9%	56.8%	55.2%
including Ratio of process execution time	17.9%	13.5%	15.7%	17.2%	10.6%

Although our proposed coordinated checkpointing protocol needs $CHECKPOINT\_DONE$ message, overhead time decreased in each process. As overhead of the coordinator which needs $CHECKPOINT\_DONE$ message is small, our proposed protocol is effective for reduction of the overhead.

5.2.2 Overhead Time according to Number of Nodes and Ratio of Process Execution Time

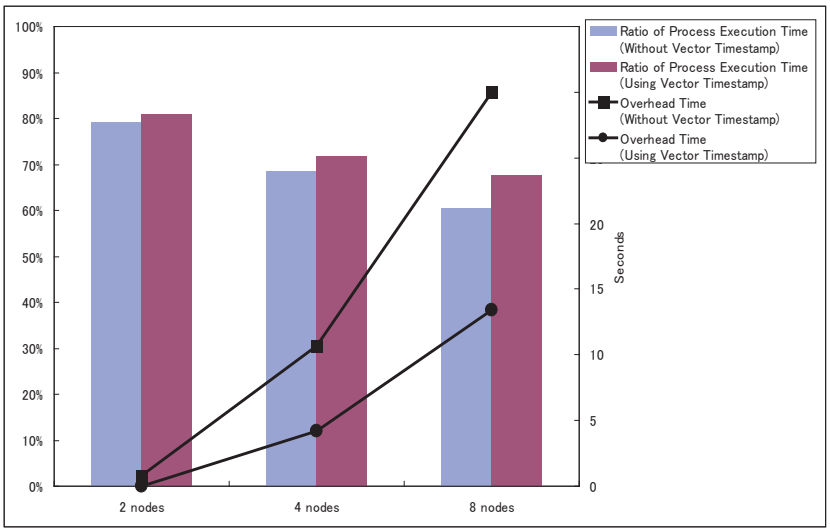

Figure 3: Overhead time and Ratio of process execution time

Figure 3 shows that as the number of the nodes increases, overhead times in both protocols become longer, while ratio of process execution time of the whole system decreases. As the number of the nodes increases, in traditional coordinated checkpointing protocol, ratio of process execution time becomes about 0.87 times, but in proposed protocol becomes about 0.91 times. This shows that proposed protocol is suitable for increase of nodes. We can result that proposed protocol is effective for scalable grid computing.

6 Conclusion and Future Problems

In this paper, we presented coordinated checkpointing using vector timestamp in grid computing. The coordinated checkpointing has been widely used due to restart in consistent state with avoidance of Domino effect for fault occurrence. However, overhead by synchronism is also inevitable in the coordinated checkpointing, we proposed coordinated checkpoint protocol using vector timestamp to reduce overhead. Performance evaluation of the coordinated checkpoint protocol using vector timestamp showed good effectiveness.

References

[1] K. M. Chandy, L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", ACM Transactions on Computer Systems, Vol.3, No.1, pp.63-75, 1985

[2] L. Alvini, K. Marzullo, "Message Logging: Pessimistic, Optimistic, Causal and Optimal", IEEE Transactions on Software Engineering, Vol.24, No.2, pp.149-159, 1998

[3] Y-M. Wang, "Consistent Global Checkpoints that Contain a Given Set of Local Checkpoints", IEEE Transactions on Computers, Vol.46, No.4, pp.456-468, 1997

[4] S. Pickles, "On the Use of Checkpoint/Recovery in Reality Grid", 2004

[5] B. Gupta, S. Rahimi, A. Thakre, N. Mogharreban, "An Efficient Non-Block Synchronous Checkpointing Scheme for Distributed Systems", Proceedings of the 2005 Conference on Parallel and Distributed Processing Techniques and Applications, pp.883-889, 2005

[6] J. Pruyne, M. Livny, "Managing Checkpoints for Parallel Programs", Job Scheduling Strategies for Parallel Processing, 1996

[7] M. Raynal, M. Singhal, "Logical Time: Capturing Causality in Distributed Systems", IEEE Computer, 29(2):49–56, 1996

[8] V. K. Garg , "Concurrent and Distributed Computing in Java", IEEE Press, 2004

[9] A. S. Tanenbaum, M. V. Steen, "Distributed Systems: Principles and Paradigms", Prentice Hall, 2002

[10] ckpt.
http://www.cs.wisc.edu/~zandy/ckpt/

[11] T. Jinno, T. Kamiya, S. Takeda, M. Nagata, "Grid Data Migration Using Service Data", Proceedings of SICE Annual Conference 2005, pp.1377-1380, 2005