Chiang Mai J. Sci. 2017; 44(2) : 688-698
http://epg.science.cmu.ac.th/ejournal/
Contributed Paper

Design and Analysis of Peer-to-Peer Fault-Tolerance Approach in a Grid Computing System

Thagorn Tangmankhong*, Peerapon Siripongwutikorn and Tiranee Achalakul
Department of Computer Engineering, Faculty of Engineering, King Mongkut's University of Technology Thonburi, 126 Pracha-Uthit Rd., Bangmod, Thungkhru, Bangkok, 10140, Thailand.
*Author for correspondence; e-mail: thagorn.tan@kmutt.ac.th

Received 9 June 2016
Accepted 11 August 2016

ABSTRACT

A grid computing system allows a large complex computing task to efficiently utilize high computing resources by splitting the task into many compute processes to be distributed and executed in parallel at many grid nodes. Under such paradigm, the system fault tolerance is the major issue as the failure of one grid node results in the task failure. Most fault tolerance techniques for a grid computing system are based on periodic savings of checkpoint data, which is used to roll back the system to the last good operating state when the failure occurs. In this paper, the fault tolerance technique based on peer-to-peer replication of checkpoint data is designed and analyzed. The idea is to allow chunks of checkpoint data to be replicated at different backup nodes to facilitate faster recovery time in the failure recovery process. The replication time under the peer-to-peer replication procedure is analyzed to obtain proper choices of chunk size and backup group size. A significant reduction in the recovery time compared to the traditional client-server approach is also gained by using the peer-to-peer replication.

Keywords: fault tolerance, grid computing, peer-to-peer replication, replication time, peer-to-peer fault tolerance

1. INTRODUCTION

A grid computing system is a group of heterogeneous nodes at geographically dispersed sites, which together can provide high performance computing power and large storage space to running applications. For example, an application that involves a large complex computing task may utilize the computing power of several nodes to finish the task in a much smaller time. The original task is divided into many subtasks or *compute processes*. These compute processes are then distributed to a group of nodes, referred to as a *working group* that collaboratively execute the subtasks to accomplish the desired goal.

Under this grid computing paradigm, the node reliability is a major issue because even a single working node failure results in the application failure. The type of failure treated in this paper is the hardware failure, where the node becomes completely dead. Therefore, it is vital that a grid computing system be fault-tolerant, which can be achieved by either *forward recovery* [1, 2, 3] or *backward recovery* [4, 5, 6, 7, 8, 9]. In forward recovery, a process in a working node

is duplicated at many backup nodes and those duplicated process run in parallel. If the working node fails, the system can restore the process from the next good states at one of the backup nodes. The approach results in fast recovery time but requires high resource consumption as the number of compute processes increases. Therefore, forward recovery is not practical for a large number of compute processes.

Backward recovery, also known as *checkpoint/restart model* [6], uses the last good state before the failure occurrence to restore the operation. In this approach, several mechanisms are needed, including Checkpointing, Replication, and Recovery [8, 10, 11, 12, 13–15, 16].

Checkpointing All running processes on the working group are periodically suspended simultaneously and the current states of those running processes in the memory, referred to as *checkpoint data*, are saved to local hard drives. Checkpoint data is essentially a snapshot of the operation states that can be used in the failure recovery process.

Replication For an application requiring high reliability, checkpoint data may be duplicated from the local disk to *backup nodes*. The replication time results from the data transfer from a working node to backup nodes over the network.

Failure Recovery When a working node failure occurs, each node rolls back to the checkpoint data to recover from the failure.

The main focus in this work is the replication technique. Traditional replication techniques in a grid computing system are based on a client-server approach, where the checkpoint data of a working node is replicated as a whole copy to one or more backup nodes depending on the reliability requirement. When a working node fails, its operation state is restored from the checkpoint data in one of the backup nodes. The backup locations can be a local node, a shared file system, or distributed over the network [17]. Using a local host for backup may not produce a high tolerance to failure. If the local node fails, the backup data could be damaged, preventing the full recovery of the system states. In order to increase the level of resiliency, backup data should be stored at the central shared file storage instead of a local host. However, if multiple replications are performed, the bottleneck problem may occur at the central system. To alleviate such a problem, a shared file system along with multiple local storages can be utilized for backups. Backup files can be split into smaller pieces and then distributed to multiple nodes [17, 18]. The method is referred to as "distributed checkpointing" or distributed replication. While this method can eliminate a single point of failure problem, it consumes a higher bandwidth because sending multiple small files to multiple locations generally creates more network traffic than sending one large file to one location. Currently, variations of the distributed replication method were implemented on many systems and many studies have been performed to tackle the overhead problem.

This paper proposes the peer-to-peer fault-tolerance approach in a grid computing system. We substantially extend the work in [19] by refining the protocol details, developing the mathematical analysis of the replication time, and conducting comprehensive performance evaluation. The main concept is to create a group of compute processes that share backup data in such a way that the replication time and the recovery time become smaller. Detailed procedures of how checkpoint data are distributed among backup compute processes as well as the operation under failure recovery are explained. The replication time is also mathematically analyzed to obtain proper choices of the system parameters, and the result is validated against simulation.

The paper is organized as follows. Section 2 presents the system environment, notations, and assumptions on which our work is based, as well as the key components in our proposed

work. It also describes the procedures of group forming, peer-to-peer replication, and failure recovery. The replication time is analyzed in Section 3 under a star topology and the proper choices of system parameter values are identified, and the analytical result is validated against simulation. The conclusion is offered in Section 4.

2. MATERIALS AND METHODS

2.1 Grid Computing Environment

We consider a homogeneous grid computing environment that consists of many grid nodes connected via a TCP/IP network. The system consists of many grid nodes and one front-end node running as the grid proxy. Figure 1 shows how various pieces of software components and compute processes inside a grid site interact. All grid nodes and the front-end node run Globus Toolkit [20]. The front-end node also runs Condor [21] as a grid scheduler and acts as the grid proxy to which a user submits a job. The system works as follows. After accepting a user job, the grid proxy contacts Grid Resources Allocation and Management (GRAM) within Globus toolkit, which is responsible for creating *Compute Processes* (CPs) based on the submitted job and distributing them to the grid nodes (assuming one CP per node). The CPs of a single user job are said to be in the same *working group*, denoted by W. Note that the working group W of each user job is determined by Condor, which handles the data dependency among CPs in the task allocation. Since all CPs in a working group belongs to the same user job, no irrelevant nodes are involved to incur unnecessarily energy consumption.

Each CP contains two components – (i) User application thread and (ii) the Peer-to-Peer fault-tolerant service. The timing at which these threads execute is depicted in Figure 2. The Peer-to-Peer fault-tolerance service comprises the following threads:

Group Forming This thread is executed only once at the beginning to create a backup group for each CP.

Checkpointing This thread is regularly executed in every checkpoint interval. It notifies Condor at the front-end node to invoke the checkpointing process for each user application thread via the grid middleware. In each CP, the user application thread will be suspended while the operating states are saved to the checkpoint data.

Replication After the checkpointing thread finishes, the replication thread replicates the checkpoint data to other CPs in its backup group by using the proposed replication mechanism. All data transfers among CPs are carried out over TCP connections.

Fault Detection During the normal process execution, the fault detection thread monitors if one of the CPs in its backup group fails and notifies the grid proxy.

Fault Recovery The grid proxy contacts M to create the new replacement CP to ıe the operation of the failed CP. The ement CP gets a list of backup CPs and es the checkpoint data from them.

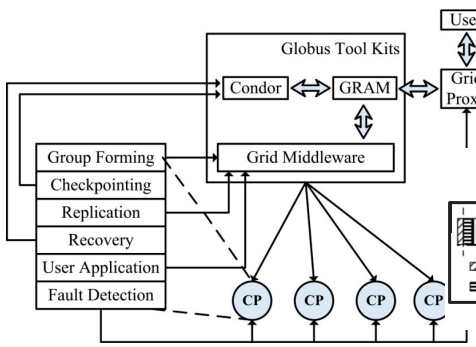


Figure 1. Grid Computing Environment

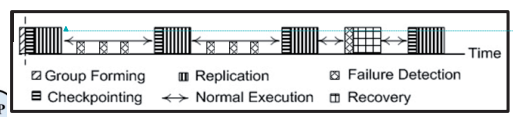


Figure 2. Timing diagram for the thread execution in the fault tolerance service.

2.2 Peer-to-Peer Fault Tolerance Approach

This section describes group forming, replication, and failure recovery procedures, which are the basis of the proposed peer-to-peer fault tolerance approach. Table 1 summarizes the notation used. The set of backup CPs for a given CP is referred to as a *backup group*. The number of CPs in a backup group is called the *backup group size*, denoted by B. To balance the resources among CPs, we enforce that each CP can belong to at most B backup groups at any time. CPs in the same backup group are said to be *peers*. So, there will be $|W|$ backup groups for each working group, all of which having the same size B, with $B < |W|$.

Initially, every CP s creates its own backup group B_s by executing the group forming procedure. Then, in each checkpointing interval, the replication procedure is executed to distribute the checkpoint data to backup CPs. In the context of replication, the CP of interest acts as the source of checkpoint data, referred to as the *source CP*. The group forming and checkpoint data replication procedures are explained below. Because all CPs in the working group perform identical operations, the group forming and replication procedures will be explained from a viewpoint of a single source CP.

2.2.1 Group forming procedure

The group forming procedure is listed in Figure 3. To form a backup group, a source CP sends Group-request messages with a unique ID (its compute process ID assigned by Condor) to all other CPs and waits for responses. Since we restrict that each CP joins at most B backup groups, only CPs that belong to less than B backup groups will acknowledge the Group-Request message together with their ID. The source CP collects the responses and ranks the responding nodes by their response times as candidates for its backup CPs. The source CP sends the Member-Request message to B candidate CPs and waits for the responses. The source CP includes the CP that acknowledges its Member-Request message in its backup group. For each Group-Request and Member-Request message sent, the timeout intervals T_g and T_m are respectively used to trigger the retransmission. In the absence of the acknowledgment from a candidate backup CP for two retries, the source CP chooses the next backup CP in the candidate list to send the Member Request message. Since we restrict that each CP joins at most B backup groups with $B < |W|$, each CP always gets B backup CPs for its backup group.

Table 1. Notations for parameters.

Parameters	Description
W	A set of CPs in the working group with size $\|W\|$
B	Backup group size
B_s	A set of CPs in the backup group of CP s with size $\|B_s\| = B$, $B < \|W\|$
T_g	Timeout interval for Group request message
T_m	Timeout interval for Member request message
Rl	Replication level
N_c	Number of chunks in checkpoint data
Δ	Checkpoint data size

Input
$\mathcal{W}$: A set of CPs in the working group
$\mathcal{C}$: A set of candidate backup CPs
B: Backup group size
T_g: Timeout for Group-Request message
T_m: Timeout for Member-Request message
Output
$\mathcal{B}_s$: Backup group for source CP s

Send a Group-Request message to all nodes in $\mathcal{W}$
while not T_g seconds has passed **do**
    if Receiving an acknowledgment message from node u **then**
        Record the response time of u
        Add u to $\mathcal{C}$
if $|\mathcal{C}| < B$ **then**
    Terminate
else
    Rank nodes in $\mathcal{C}$ ascendantly based on their response times
    Send the Member-request message to B CPs in $\mathcal{C}$
for each $u \in \mathcal{C}$ to which the Member-Request message is sent **do**
    if No acknowledgment is received from u in T_m seconds **then**
        if Two member-request messages have been retransmitted to u **then**
            Remove u from $\mathcal{C}$
            Find another $v \in \mathcal{C}$, $v \notin \mathcal{B}_s$ with smallest response time
            if No such node v exists **then**
                Terminate
            else
                Send the Member-request message to v
        else
            Retransmit the Member-request message to u
    else
        Add u to $\mathcal{B}_s$
    if $|\mathcal{B}_s| = B$ **then**
        Break
Rank CPs in $\mathcal{B}_s$ by their ID.
Send the Group-Confirm message containing the group list member to all CPs in $\mathcal{B}_s$

(a) Source CP operation

for Each message received **do**
    if Already belongs to B backup groups **then**
        break
    if Receive a Group-Request message from node s **then**
        Return acknowledgment message with its process ID to s
    else if Receive a Member-Request message from node s **then**
        Return acknowledgment message to s
    else if Receive a Group-Confirm message **then**
        Keep the backup group member list
        Mark its status as joining a backup group

(b) Backup CP operation

Figure 3. Group forming procedure for source CP and backup CPs.

After the source CP receives the acknowledgments for all its Member-Request messages, it sends the backup group member list to all its backup CPs. The list is ranked by the process IDs so that each backup CP knows both the backup group members as well as its successor in the backup group.

2.2.2 Replication procedure

The replication procedure immediately follows the checkpointing procedure. Each source CP divides the checkpoint data into N_c chunks that are integral multiples of B. That is, $N_c = M \cdot B, M \cdot Z^+$. Each chunk is numbered sequentially.

Denote R_l as the *replication level*, which is the number of copies of checkpoint data in addition to the source copy. At the end of the replication procedure, each chunk will be stored at R_l backup CPs. This means that if the application thread is recoverable if a source CP fails and at most $R_l - 1$ backup CPs fail.

The replication procedure is illustrated by example as follows. Consider a group of four working CPs with three backup CPs per backup group ($|W| = 4$, $B = 3$) as shown Figure 4. Suppose $N_c = 9$ and $R_l = 2$. Without loss of generality, let us denote a source CP of interest by P_0, and the backup CPs by P_1, P_2, and P_3, ranked by their process ID. The replication procedure at the source CP and backup CPs work as follows:

Source CP: The source CP transfers N_c/B chunks to each of its backup CP. The transfer is carried out sequentially for chunks to the same backup CP, and in parallel for chunks to different backup CPs. In the example, P_0 sequentially transfers chunks 1, 4, 7 to P_1, chunks 2, 5, 8 to P_2, and chunks 3, 6, 9 to P_3. Generally, P_0 transfers chunks $i + (j - 1)B$ to CP i, where $i \in \{1, 2, \ldots, B\}$ and $j \in \{1, 2, \ldots, M\}$.

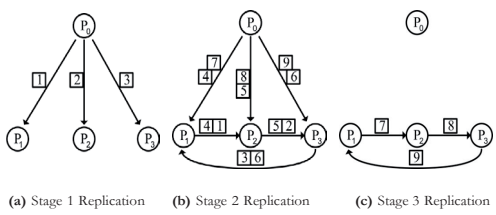


Figure 4. Replication mechanism.

Backup CP: For chunks received from the source CP, the backup CP is responsible for distributing them to other $R_l - 1$ backup CPs. For $R_l = 1$, the backup CP only keeps chunks to itself. For $R_l = 2$, the backup CP transfers each chunk to its successor. In general, there exists R_l copies of each chunk at R_l backup CPs, which is done by each backup CP iteratively transferring a received chunk to its successor until R_l copies exist in R_l backup CPs. Thus, each backup CP eventually holds exactly $M \cdot R_l$ chunks. In this example of $R_l = 2$, P_1 will replicate chunks 1, 4, 7 received from the source CP to P_2, and likewise for P_2 and P_3. At the end of the replication procedure, $P1$ holds chunks 1, 3, 4, 6, 7, 9, P_2 holds chunks 1, 2, 4, 5, 7, 8, and P_3 holds chunks 2, 3, 5, 6, 8, 9.

The above replication procedure enforces that $R_l \leq B$. For $R_l = B$, all backup CPs will store the complete checkpoint data. In practice, $R_l = 2$ is commonly used (data is replicated at two sites), and higher values of R_l is rare.

2.2.3 Failure recovery procedure

During the user application execution period, every CP monitors the liveliness of its backup CPs (e.g., by periodically issuing the ping command). Consider the case when a particular CP p fails. All CPs that has p as its backup will detect the failure and send a fault detection message to the grid proxy, and hence many fault detection messages can be received by the grid proxy. After the grid proxy has verified the node failure, it requests GRAM to create a replacement CP at another grid node not in the current working group W. Then, the grid proxy notifies the monitoring CPs of the replacement CP. Because the replacement

CP needs to get the backup checkpoint data for the recovery, only monitoring CPs that are in the backup group of p are relevant. Those CPs will send the backup group member list to the replacement CP, which in turn asks for chunks from backup CPs in the list. Once the complete checkpoint data has been acquired, the replacement CP calls Condor restore function to resume the user application execution thread.

3. RESULTS AND DISCUSSION

3.1 Analysis of Replication Time

This section analyzes the replication time under a TCP/IP network as a function of checkpoint data size, chunk size, and the backup group size. The major difference between our proposed Peer-to-Peer replication approach and the client-server replication approach lies in the replication time, which is the time taken for the checkpoint data of a source CP to be completely replicated at all of its backup CPs. In the client-server replication, the whole checkpoint data is transferred from the source CP to backup CPs while in the Peer-to-Peer replication, different chunks from the source CP are transferred to different backup CPs, and those chunks are also replicated among backup CP. Because chunks are transferred over the network, the replication time strongly depends on the network topology of the grid nodes and the flow pattern, i.e., how many flows on each network link at a given time. To enable tractable analysis, we assume that the grid nodes are interconnected in a star topology as shown in Figure 5. For a more complex topology, the replication time could be determined by manually constructing the flow pattern in the network. The performance on more complex topologies is left for future work.

We denote the replication time by T, the checkpoint data size by D (in MB), the chunk size by Δ (in MB). The replication level R_l = 2 is assumed. Recall that in the replication procedure, the checkpoint data is divided into chunks and individual chunks are then transferred among backup peers over TCP connections. Consequently, the replication time can be calculated based on the TCP connection throughput and how chunks are transferred among the peers. To analyze the replication time, consider the scenario of one source CP (P_0) with three backup CPs (P_1, P_2, and P_3) in Figure 4 with one CP per grid node. The corresponding network topology for this scenario is shown in Figure 5. The source CP divides the checkpoint data into nine chunks, chunks 1, 4, 6 for P_1, chunks 2, 5, 7 for P_2, and chunks 3, 6, 9 for P_3. The replication time can be analyzed in three stages as follows:

1st stage: The source CP transfers the first chunk of each backup CP simultaneously over three TCP flows as shown in Figure 6(a), which takes T_1 to finish. For n_b backup CPs, B parallel flows exist in the link between the source CP and the switch, and one ow on all the other links. Because the transfer time is dictated by the link with largest number of flows, we have

$$T_1 = \frac{\Delta}{R(n_b)}$$

where $R(n_b)$ is the per-flow TCP throughput (in Mbps) over a single link having n_b flows.

2nd stage: In the second stage, the source CP continues to send the remaining chunks to the backup CPs, while each backup CP starts

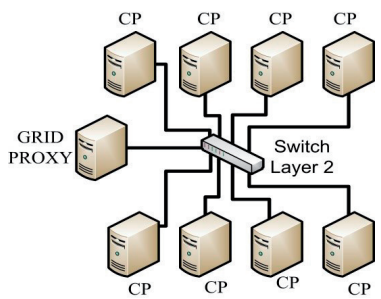


Figure 5. Network topology used for the analysis of replication time.

replicating the completely received chunks to its successor. As shown in Figure 6(b), while receiving chunk 4 from the source CP, P_1 shares chunk 1 with P_2. Then, it shares chunk 4 with P_2 while receiving chunk 7, and so on. This stage lasts for the amount of time for the source CP to transfer the remaining chunks to individual backup CPs. Denote the time taken in the second stage by T_2. With n_b CPs per backup group, the number of chunks left to be sent from the source CP to each backup CP is (D/N_b) - Δ because the first chunk of each backup CP has already been transferred in the first stage. For each CP i, let $N_i = \{n_{out}, n_{in}\}$ be a tuple representing the number of outgoing parallel sessions to the switch and the number of incoming parallel sessions from the switch. From Figure 6(b), at the source CPs, $N_s = \{3, 0\}$, and all the other backup CPs have $N_b = \{1, 2\}$. In general, the source CP will have $N_s = \{n_b, 0\}$ while each backup CP will have $N_s = \{1, 2\}$ because it has to send its chunks to its successor while receiving one chunk from the source CP and one chunk from its predecessor. Since there are $|W|$ CPs in the working group, there must be $|W|$ backup groups, one for each CP. As each CP belongs to N_b backup groups, we have that for each CP i, the number of parallel flows to and from the switch is given by

$$N_i = \left\{ n_i + \sum_{j=1}^{n_b} 1, 2 \sum_{j=1}^{n_b} 1 \right\}$$
$$= \{2n_b, 2n_b\}$$

Therefore, there exists $2n_b$ parallel flows between each grid node and the switch in both directions, and it follows that

$$T_2 = \frac{(D/n_b) - \Delta}{R(2n_b)}$$

3rd stage: The third stage starts when the source CP finishes transferring the remaining chunks to its backup CPs, and each backup CP replicates the last chunk with its successor. As shown in Figure 6(c), the source CP will have $N_s = \{0, 0\}$, while each backup CP will have $N_b = \{1, 1\}$, because it has to share its chunks to the successor while receiving one chunk from its predecessor. Therefore, there exists n_b parallel flows between each node and the switch in both directions. It follows that the time taken in this 3rd stage is given by

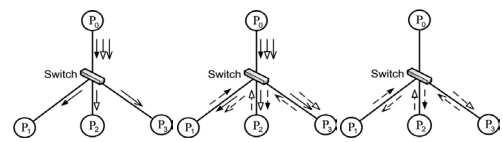
$$T_1 = \frac{\Delta}{R(n_b)}$$

Combining the replication time from the three stages above, we have

$$T(D, C, p) = T_1 + T_2 + T_3$$

$$T(D, \Delta, n_b) = \left(\frac{\Delta}{R(n_b)}\right) + \left(\frac{\left(D/N_b\right) - \Delta}{R(2n_b)}\right) + \left(\frac{\Delta}{R(n_b)}\right)$$
$$= \left(\frac{2\Delta}{R(n_b)}\right) + \left(\frac{\left(D/N_b\right) - \Delta}{R(2n_b)}\right) \quad (1)$$

where, $\mathrm{p} \leq \frac{\mathrm{D}}{\mathrm{C}}$ and $1 \leq n_b \leq min\left(|W|, \left|\frac{D}{\Delta}\right|\right)$



(a) Stage 1 Replication **(b)** Stage 2 Replication **(c)** Stage 3 Replication

Figure 6. Flow patterns in different stages of the replication procedure.

3.2 Choice of Backup Group Size

From (1), our goal is to find appropriate values of n_b and Δ that minimizes the replication time for a given checkpoint data size *D*. To accomplish so, we first determine the expression of TCP per-flow throughput $R(.)$ Essentially, the key to TCP throughput is that TCP uses the end-to-end congestion control mechanism that keeps increasing the window size every round-trip time until reaching the maximum window size and flows sharing the same bottleneck link get approximately equal throughputs. The TCP throughput is

proportional to the window size divided by the round-trip time. So, only after the window size reaches its maximum (65,535 bytes), we will get the maximum TCP throughput. If the amount of data transferred is too small, the TCP session may finish before reaching its maximum throughput.

The behavior of TCP per-flow throughput over a single 1 Gbps link is investigated by using ns-2 simulation. Figure 7(a) plots the TCP per-flow throughput against the data transfer size (i.e., chunk size) at different numbers of TCP flows (n). For a given number of flows, TCP per-flow throughput increases with the data transfer size and starts to converge to a constant after the data size goes beyond MB regardless of the number of parallel sessions. The chunk size Δ should thus be at least 1 MB to achieve the maximum TCP throughput in the transfer, and the TCP throughput becomes independent of the data transfer size. To derive the form of the TCP throughput per session, $R(n)$, we simulate the TCP throughput per session as shown in Figure 7(b) under 1 MB data size. The fitted regression model of the throughput curve is given by

$$R(n) = \frac{\alpha}{\beta + n} \text{ Mbps}, \quad (2)$$

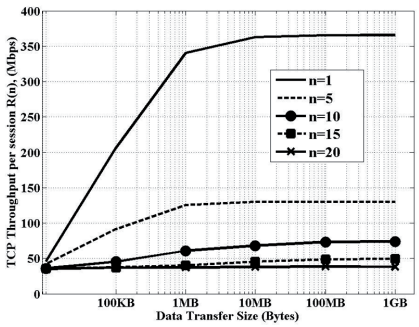
$$\alpha = 776.15, \beta = 0.92$$

Substituting (2) in (3) and given D and Δ, we have

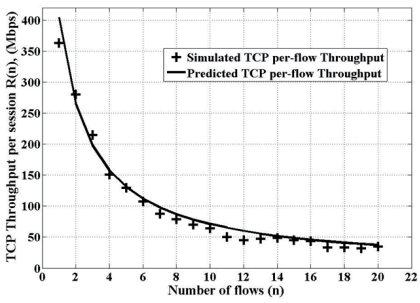
$$T(n_b) = c_1 + \left(\frac{c_2}{n_b}\right), \quad (3)$$

where $c_1 = \frac{\beta\Delta + 2D}{\alpha}$, $c_2 = \frac{D\beta}{\alpha}$

Another important observation from Figure 7(b) is that the aggregate throughput on the link increases with the number of flows. For example, for two flows, the aggregate throughput is about 560 Mbps while for three flows, the aggregate throughput is about 645 Mbps. This behavior benefits the recovery time under Peer-to-Peer replication, which will be discussed later in Section 4.



(a) TCP per-flow throughput vs. transfer data size at different number of flows.



(b) TCP per-flow throughput as a function of the number of flows

Figure 7. TCP per-flow throughput under a single 1 Gbps link.

From (3), we see that the replication is a decreasing function of n_b that converges to $2\Delta/R(n_b)$ at $n_b = \lfloor D/\Delta \rfloor$. Because using a larger backup group size means more resources, we select n_b at which $T(n_b)$ no longer significantly decreases, say by some small ratio δ, as n_b increases. Therefore, our goal is to find n^* such that

$$\frac{T(n^*+1) - T(n^*)}{T(n^*)} = \frac{c_2\left(\frac{1}{n^*+1} - \frac{1}{n^*}\right),}{c_1 + \left(\frac{c_2}{n^*}\right),} = \delta$$

$$\delta c_1 n^2 + \delta(c_1 + c_2)n - (1-\delta)c_2 = 0 \quad (4)$$

Note that (4) has two roots with opposite signs. It follows that n^* is the positive root of (4), which is given by

$$n^* = \frac{-\delta c_1 + \sqrt{\delta(c_1+c_2)^2 - 4\delta(1-\delta)c_1c_2}}{2\delta c_1} \quad (5)$$

and the choice of backup group size is

$$B = min\left(\lceil n^* \rceil, |W|, \left\lfloor \frac{D}{\Delta} \right\rfloor\right) \quad (6)$$

As an example, with 1-GB checkpoint data, 1-MB chunk size (D = 1000 MB and Δ = 1 MB) and a large working group, we have c_1 = 2.5780, c_2 = 1.1853. For δ = 0.02, the backup group size calculated from (6) is B = 4.

Observe from (3) that if $\Delta \ll D$, c_1 and c_2 are approximately D scaled by constants, and the solution of n^* in (5) becomes independent of D. Therefore, the choice of backup group size can be set to four.

3.3 Simulation Results

This section evaluates the replication time obtained from the analysis in Section 3.2 against ns-2 simulation. The recovery time under Peer-to-Peer replication and client-server replication are also compared. We simulated a grid environment with ten grid nodes under the star network topology as in Figure 5 with 1-Gbps links and a 200 microseconds of the end-to-end propagation delay.

Figure 8 shows the replication time as a function of the backup group size (B). The results from analysis and simulation are consistent and clearly validate the analysis that as the backup group size increases beyond four, the replication time starts to converge.

For the recovery process, the time to recover the checkpoint data from backup CPs to a replacement CP is observed after a node failure. Because the replacement CP needs to be created by GRAM, the recovery overhead occurs when backup CPs of the failed CP send backup chunks to the replacement CP. We measure the recovery time from the start of backup transfer. Figure 9 shows the recovery times under Peer-to-Peer replication compared to the client-server replication at different backup group sizes. As expected, the recovery time increases linearly with the checkpoint data size. However, the recovery time under Peer-to-Peer replication are always smaller because the aggregate throughput on the link increases with the number of flows as shown earlier in Figure 7(b). The results suggest that our proposed Peer-to-Peer fault tolerance approach can reduce both replication time and recovery time compared to the traditional client-server approach.

4. CONCLUSIONS

A peer-to-peer fault tolerance approach in a grid computing system is proposed to reduce the replication time and the recovery time in the backup process. The backup process is performed by nodes in a working group to replicate checkpoint data in parallel. The main

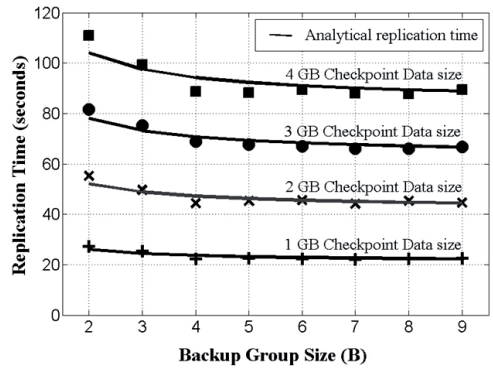


Figure 8. Replication time as a function of the backup group size at different check- point data sizes (Δ = 1 MB, |W| = 10 nodes).

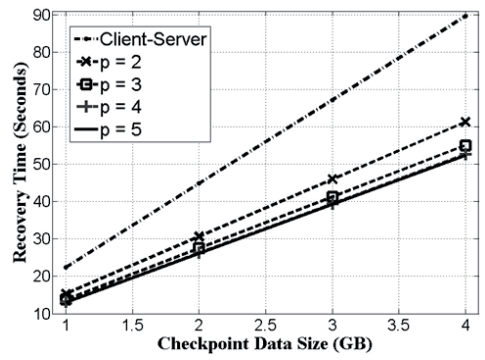


Figure 9. Comparison of the recovery time.

procedures including group forming, replication, and failure recovery are described and the replication time is analyzed under a basic star topology to obtain suitable parameter values. Our mathematical analysis, also validated against simulation, reveals that negligible reduction in the replication time is gained once the backup group size is beyond four. Furthermore, the recovery time under the peer-to-peer approach is always better than that of the client-server approach due to simultaneous transfers of data among compute processes. Our replication time analysis assumes the fixed redundancy level of two and a simple star topology. However, there may be a need of a higher redundancy level in some cases and the system may have a more complicated network topology. The study of how our peer-to-peer fault tolerance approach performs under more generic circumstances will be left for the future work.

ACKNOWLEDGEMENTS

This work is supported by the Thailand Research Fund and King Mongkut's University of Technology Thonburi through the Royal Golden Jubilee Ph.D. Program under Grant No. PHD/0247/2549.

REFERENCES

[1] Agbaria A. and Friedman R., *J. Clust. Comput.*, 1999; **6**: 167-176.

[2] Charoenchaiamornkij P. and Achalakul T., *Proceeding of Technology and Innovation for Sustainable Development Conference 2008*, 2008.

[3] Lee J., Chapin S. and Taylor S., *J. Qual. Reliab. Eng. Int.,* 2002; **18**.

[4] Abewajy J., *Proceeding of International Conference on Computational Science and Its Applications*, 2004; 107-115.

[5] Budhiraja N., Marzullo K., Schneider F.B. and Toueg S., *Proceeding of the 6th International Workshop on Distributed Algorithms*, 1993.

[6] Grabriel E., Fagg G., Bukovsky A., Angskun T. and Dongarra J., *Proceeding of 17th Annual ACM International Conference on Supercomputing, 2003*, 2003.

[7] Ouyang J. and Maheshwari P., *Proceedings of IEEE Second International Conference on Algorithms and Architectures for Parallel Processing*, 1996.

[8] Ozaki T., Dohi T. and Okamura H., *IEEE T. Depend. Secure Comput.*, 2006; **2**: 130-140.

[9] Zhang X., Zagorodnov D., Hiltunen M., Marzullo K. and Schlichting R., *Proceeding of IEEE International Conference on Cluster Computing*, 2004; 105-114.

[10] Garg R. and Singh A., *J. Comput. Sci. Eng. Survey*, 2011; DOI 10.5121/ijcses.2011.2107.

[11] Gottumukkala N., *Failure Analysis and Reliability-Aware Resource Allocation of Parallel Applications in High Performance Computing Systems*, Ph D Thesis, Louisiana Tech University, USA, 2008.

[12] Gottumukkala N., Liu Y., Leangsuksun C., Nassar R. and Scott S., *Proceeding of High Availability and Performance Workshop 2006 in conjunction with Los Alamos Computer Science Institute Symposium*, 2006.

[13] Luckow A., *J. Future Gener. Comp. Sy.*, 2008; **24**: 142-152.

[14] Naksinehaboon N., Paun M., Nassar R., Leangsuksun C. and Scott S., *Int. J. Comput. Commun. Control*, 2009; **4**: 386-400.

[15] Nandagopal M. and Uthariaraj v., *Int. J. Eng.Sci. Technol.*, 2010; **2**: 4361- 4372.

[16] Oliner A., Rudolph L. and Sahoo, *Proceeding of the 20th Annual International Conference on Supercomputing*, 2006; 14-23.

[17] Al-Kiswany S., Ripeanu M., Vazhkudai S. and Gharaibeh A., *Proceeding of The 28th International Conference on Distributed Computing Systems 2008*, 2008; 613-624.

[18] Song U., Gil J. and Hong S., Checkpoint Sharing-Based Replication Scheme in Desktop Grid Computing, *Embedded and Multimedia Computing Technology and Service Lecture Notes in Electrical Engineering*, 2012; 477-484.

[19] Tangmankhong T., Siripongwutikorn P. and Achalakul T., *Proceeding of the 9th International Joint Conference on Computer Science and Software Engineering (JCSSE2012)*, Bangkok, Thailand, 30 May 2012 – 1 Jun 2012.

[20] The Globus Alliance, About the Globus Toolkit, Available at: http://About the Globus Toolkit.

[21] UW-Madison Research, HTCondor, Available at: http://research.cs.wisc.edu/htcondor/.