Group-based Coordinated Checkpointing for MPI: A Case Study on InfiniBand

Qi Gao   Wei Huang   Matthew Koop   Dhabaleswar K. Panda

Technical Report
OSU-CISRC-1/07-TR07

Group-based Coordinated Checkpointing for MPI: A Case Study on InfiniBand [*]

Qi Gao Wei Huang Matthew Koop Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{*gaoq, huanwei, koop, panda*}*@cse.ohio-state.edu*

Abstract

With the rapid development in High End Computing (HEC) systems, more and more computer clusters with thousands nodes are being deployed. Fault tolerance in cluster environments has become a critical requirement to guarantee the completion of application execution. Checkpointing and rollback recovery is a common approach to achieve fault tolerance. Although widely used in practice, coordinated checkpointing has known limitation on scalability especially when a central storage system is used to store checkpoint files. In this paper, we propose a new design of group-based checkpointing to alleviate the scalability limitation of coordinated checkpointing. By carefully scheduling the MPI processes to take checkpoints in smaller groups, group-based checkpointing reduces the effect of the storage bottleneck and allows the rest of processes to make progress while one group of processes are taking checkpoints. Detailed evaluations are conducted through micro-benchmarks and High Performance Linpack (HPL). The experimental results show that the group-based checkpointing can reduce the effective delay for checkpointing significantly, up to 78% for HPL on 32 nodes.

1 Introduction

With the rapid development in High End Computing (HEC) systems, more and more computer clusters with thousands nodes are being deployed. The failure rates of those systems, however, also increases along with their size and complexity. As a result, the running times of many scientific applications are becoming longer than the mean-time-between-failure (MTBF) of the cluster systems. Therefore, fault tolerance in cluster environments has become a critical requirement to guarantee the completion of application execution. A commonly used approach to achieve fault tolerance is checkpointing and rollback recovery, where the intermediate states of running parallel applications can be saved and used for restart upon failures.

The Message Passing Interface (MPI) [21] is the *de facto* programming model on which parallel applications are typically written. However, since the MPI has no specification regarding fault tolerance, critical MPI applications have to rely on their own checkpointing mechanism, which increases the design time for new applications and potentially leads to rewriting code of legacy applications. In addition, it is difficult for application designers to predict the failure rates of different systems on which the application will run and adjust the checkpointing algorithm accordingly. Therefore, providing checkpointing support in the MPI library is desirable in many cases to achieve application transparency.

There are two main categories of checkpointing protocols: coordinated checkpointing and uncoordinated checkpointing, which use different approaches to guarantee global consistency. Uncoordinated checkpointing allows processes to save their states independently of each other, but requires message logging to maintain consistency and avoid cascading rollback propagation, a.k.a. *domino effect* [23]. Since there is a clear trend for large clusters to be equipped with high performance networks, such as InfiniBand [18], the overhead of message logging would be prohibitive considering the very high bandwidth and message rate the networks can provide. Coordinated checkpointing guarantees the consistency by global coordination, and usually requires all processes to save their states at relatively the same time. Therefore, it has inherent limitations on scalability. In a real-world scenario, checkpoint files are usually stored on a reliable central storage system, which tends to be a bottleneck for checkpointing of large parallel jobs with

[*]This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco systems, Linux Networx and Sun Microsystems; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, Dell, Microway, PathScale, IBM, SilverStorm and Sun Microsystems.

many processes.

In this paper, we propose group-based checkpointing as an extension to the coordinated checkpointing protocol to alleviate the scalability limitation caused by the storage bottleneck. The key idea of group-based checkpointing is that when taking a consistent global checkpoint, all the processes are carefully scheduled to take individual checkpoints so that they do not access the stable storage at same time. By checkpointing a smaller group of processes at one time, the processes in the checkpointing group can obtain a larger share of the bandwidth to storage and thus reduce their checkpointing delay greatly. While at the same time, the design allows processes in other groups to proceed with their execution as much as possible, and only communications which can introduce inconsistency of the global checkpoint is deferred. On one hand, this design limits the number of processes concurrently accessing central storage to reduce the checkpoint delay for large MPI jobs, on the other hand it avoids the message logging overhead to provide a very low overhead in failure free case.

We have implemented our design of group-based checkpointing on MVAPICH2 [20, 22], a high performance MPI implementation over InfiniBand, and evaluated our design on an InfiniBand cluster. The experimental results indicate that our new design reduces the effective delay for checkpointing significantly, up to 78% for HPL benchmark on 32 nodes. Though our current implementation is based on InfiniBand and MVAPICH2, the design can be readily applicable to other coordination checkpointing protocols for other MPI implementations.

The rest of the paper is organized as follows: In Section 2 we briefly discuss the background. In Section 3 we present the design of group-based checkpointing and discuss some key design issues. In Section 4, we describe the metrics of checkpoint delay and the evaluation results. In Section 5, we discuss related works. Finally, we conclude this paper and briefly mention future directions in Section 6.

2 Background

2.1 Checkpointing Technologies

Checkpointing and rollback recovery is a commonly used approach for failure recovery. By periodically saving the running state of applications, the checkpointing system allows applications to restart from a time point in the middle of their executions upon failure instead of from the beginning. A detailed survey on rollback recovery techniques can be found in [13].

As discussed in Section 1, uncoordinated checkpointing has larger overhead in failure free cases because of message logging, but coordinated checkpointing has scalability limitation largely due to the storage bottleneck.

Several technologies are available to make use of local disk or spare memory on peer nodes as temporary buffer to hold checkpoint files. However, there are three facts making these approaches less practical: a) Scientific applications tend to use as much memory as possible for computation, which makes the node memory to be a scarce resource and not able to be used to store checkpoints. b) The local disk may not be available on the computing nodes in new large clusters, e.g. the compute nodes in the Peloton clusters in LLNL, are diskless [19]. c) Buffering the checkpoint files in local disk may undermine the ability to recover from failure, because if the node crashes before the checkpoint files are completely transferred to reliable storage, the checkpoint files may be lost.

Note that in this paper we refer the blocking design of coordinated checkpointing protocol as 'coordinated checkpointing', which does not require any message logging. Although the non-blocking design of Chandy-Lamport coordinated checkpointing protocol [10] allows processes to make progress when other processes are taking checkpoints, it does not explicitly schedule them globally. Therefore, in an MPI application, all the processes are still likely to take checkpoints at the same time and hit the storage bottleneck. In addition, the non-blocking protocol also requires logging some messages in the channels during checkpointing. The message logging overhead for this protocol on high speed interconnect is reported in [11].

2.2 Checkpointing on OS-bypass Networks

In recent years, high speed networks have been deployed on clusters to improve the overall performance. These interconnects usually use intelligent network interface cards (NIC) and provide user-level protocols to allow OS-bypass communications for low latency and high bandwidth. However, as compared to Ethernet, these high performance interconnects introduce additional challenges for system-level checkpointing. These challenges are discussed in our previous work [15]. Essentially, since communication context is cached on NIC and also memory mapped in process address space for user-level access, network connections are usually required to be explicitly torn down before the checkpoint and reconnected afterwards.

InfiniBand [18] is a typical and widely used high performance interconnects, which uses connection-oriented model for best performance and advanced features, such as Remote Direct Memory Access (RDMA). And most MPI implementations on InfiniBand use connection-oriented model as default. To establish connections on InfiniBand, however, two peers have to exchange connection parameters using other communication channels. As a result, the cost for connection management are much higher as compared to using the TCP/IP protocol. And, even for a local checkpoint on one node to take place, all the remote processes it connects to need to participate in the costly procedure for connection tear-down and rebuild. This requirement makes the checkpointing protocols like uncoordinated checkpointing and non-blocking coordinated checkpointing much more expensive to use on InfiniBand.

3 Group-based Coordinated Checkpointing

3.1 Motivation

In coordinated checkpointing, the overall checkpointing delay consists of two parts, coordination delay and storage accessing delay. With the order-of-magnitude improvement in network speed but limited improvement in disk speed in recent years, the current dominating delay for checkpointing is the storage accessing delay [13]. For large-scale parallel applications, the central storage system is likely to be a bottleneck of checkpointing. Usually the throughput of a storage system is limited, and thus the more number of processes accessing the storage concurrently, the less bandwidth each process obtains.

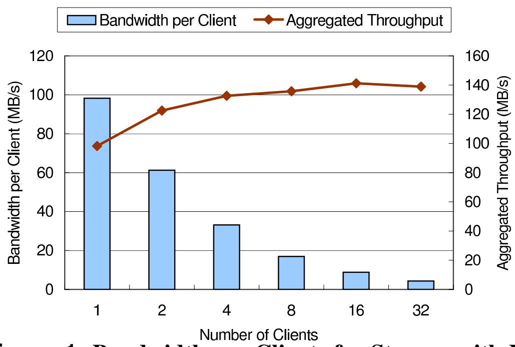

Figure 1. Bandwidth per Clients for Storage with Different Number of Clients

Figure 1 demonstrates that using 4 PVFS2 servers with a total of about 140MB/s throughput as the storage, the bandwidth each clients obtains decreases significantly when the number of clients increases. The case where 32 clients shares 140MB/s reflects a typical ratio of the number of CPUs to storage bandwidth in real cluster deployment. e.g. 1024-node cluster of Peloton clusters in LLNL has 8192 CPUs and the storage throughput is 25.6 GB/s [19]. From the experiment, we can see that it is desirable to have a coordination protocol to limit the number of processes taking checkpoints at the same time.

In addition, previous studies [25] indicate that in many parallel applications each process only communicates to a limited number of peer processes. Therefore the MPI processes can potentially make reasonable progress while some other MPI processes in the MPI job take their checkpoints.

Based on these observations, we design group-based checkpointing to explore the potential opportunities to reduce the checkpointing delay for large MPI jobs.

3.2 The Proposed Design

The main goal of group-based coordinated checkpointing is that when checkpointing a large MPI job with many processes, all the MPI processes should be carefully scheduled to take individual checkpoints at slightly different time to avoid the storage bottleneck, and meanwhile the global consistency should be maintained by a coordination protocol without message logging to avoid extra overhead.

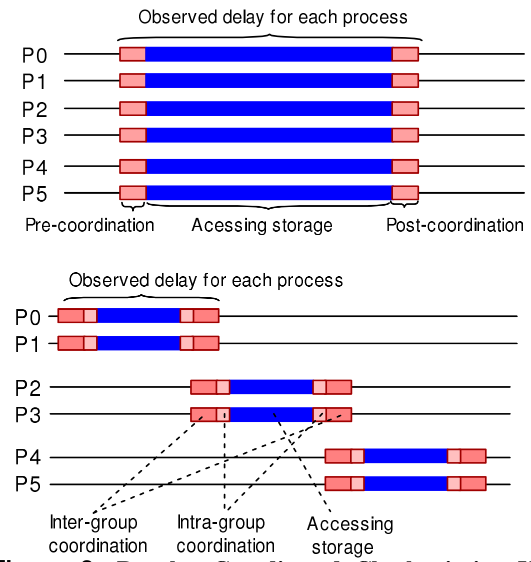

Figure 2. Regular Coordinated Checkpointing V.S. Group-based Coordinated Checkpointing

As illustrated in Figure 2, in our design, instead of all processes trying to take checkpoints at the same time and suffering a large checkpointing delay, processes are into smaller groups and scheduled to take their own checkpoints group by group to form a consistent global checkpoint. Therefore, each processes in the checkpointing group can obtain a much larger bandwidth to the storage system and thus reduce their checkpointing delay greatly.

The group-based design of coordinated checkpointing certainly involves more coordination overhead. However, even on Gigabit Ethernet, the coordination delay is very small as compared to storage accessing delay, and it can be further reduced in the order of magnitude by utilizing high speed networks. Therefore, when considering the trade-offs between coordination overhead and storage accessing overhead of checkpointing in HPC environment, it is preferable to optimize the storage accessing delay at the cost of increasing the coordination overhead to a certain extent.

Group-based coordinated checkpointing involves two levels of coordination protocols. The inter-group coordination protocol is used to schedule the progress groups to take checkpoints in turn and guarantee the consistency among different groups, and the intra-group coordination protocol is used to guarantee the consistency among processes within a group. For intra-group coordination, any existing coordinated checkpointing protocol can be applied. For inter-group coordination, to guarantee the consistency, the protocol needs to ensure that the groups which have already taken

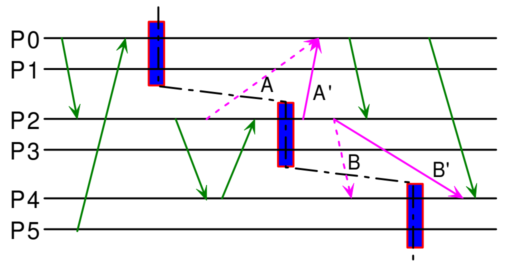

Figure 3. **Consistency in Inter-group Communication during Checkpointing**

the checkpoint do not communicate with the groups which have not yet taken the checkpoint.

As shown in Figure 3, checkpoints in three groups form a recovery line. Within a group no message passing can cause inconsistency, but between different groups, message passing across the recovery line can potentially lead to an inconsistent state. For example, at restart time, message A will become a lost message, and message B will become an orphan message. Therefore, message A and B need to be deferred to A' and B' respectively so that no message is passed across the recovery line. Under this condition, a consistent recovery line is formed without logging any message, and later each process can restart from its own checkpoint with a consistent view of a global state.

Another alternative to handle messages crossing the recovery line is to log these messages and resolve the potential inconsistency at restart time. However, there are performance trade-offs involved for that approach especially on high speed networks. To enable message logging, zero-copy rendezvous protocols, which are commonly used in high speed networks to improve performance, can not be used, which will affect the performance in failure-free cases negatively. Therefore, in our current design, we choose to defer the message passing across the recovery line to resolve inconsistency.

An important issue in group-based checkpointing is how to form checkpoint groups. To achieve maximum benefits, checkpoint groups should be formed matching with the communication pattern of the application so that most frequent communication happens within groups. In addition, the size of the group also plays an important role in checkpointing delay since it is related to not only the communication pattern but also the efficiency for accessing storage. We will discuss these issues in more detail in Section 4.

3.3 Detailed Design Issues

We have implemented our design of group-based checkpointing on MVAPICH2 [20], a high performance MPI-2 implementation over InfiniBand based on MPICH2 [2].

Several key design issues are discussed here and our solutions based on MVAPICH2 are also described. However, these issues can be addressed similarly on different MPI implementations.

3.3.1 Coordination Framework

Our coordination framework to support group-based checkpointing is based on our previous work [15], in which a framework has been designed to coordinate all MPI processes to take global consistent checkpoints concurrently, upon user's request or periodically with a certain time interval. In this design, we extend the functionality of some key components of the framework including the central Checkpoint/Restart (C/R) coordinator in the MPI job console, and the local C/R controller in individual MPI processes. The central C/R coordinator is redesigned to manage group information, including the members' ranks and the checkpointing states of group, so that it keeps track of which groups have taken the checkpoint already and which have not. And one more state called 'passive coordination' is added in the state machine maintained by the local C/R controller. This state indicates that another group of processes are taking the checkpoints, and the process may be required to perform necessary coordination, such as connection teardown or rebuild.

Our current coordination framework utilizes both the off-band channel and the in-band channel. The off-band channel is based on our extended version of the Multi Purpose Daemon (MPD) [9], the default process manager for MPICH2 and MVAPICH2. Since this is the only channel available between the MPI job console and individual MPI processes, the off-band channel is used for propagating checkpoint requests and group states, as well as scheduling process groups to take checkpoints in turn. The in-band channel is mainly used for connection management, which is discussed next.

3.3.2 Connection Management

As mentioned in Section 2, connection management for InfiniBand is more complex and more costly. In the regular coordinated checkpointing case, since all processes participate in checkpointing at the same time, they can tear-down and rebuild connections collectively. In group-based checkpointing, however, the connections between processes must be controlled dynamically on a per connection base, so that: a) Each MPI process can disconnect/reconnect to only a specific subset of processes. b) Any side can initiate a connect/disconnect procedure without the active participation from remote side.

We have designed a connection manager for group-based checkpointing using Unreliable Datagram (UD) provided by InfiniBand. The UD-based connection manager has better performance and scalability as compared to connection manager using TCP/IP, which makes it more suitable for checkpointing since connection establishment and teardown happens much more frequently here.

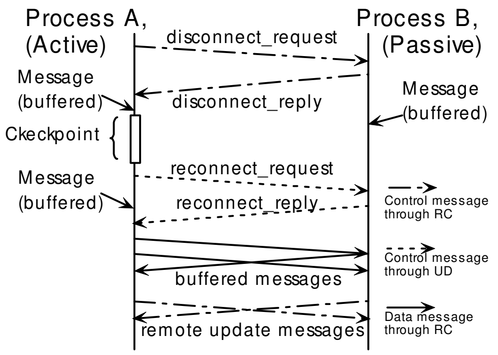

Figure 4. Connection Management in Group-based Checkpointing

The working scenario of the connection management is illustrated in Figure 4. Suppose process A is about to take a checkpoint and process B is not in the checkpointing group. The process A will serve as the active side of the protocol to first disconnect from process B before the checkpoint, and reconnect to process B as needed after the checkpoint. The detailed discussion on the channel consistency between MPI processes during checkpointing can be found in our previous paper [15].

3.3.3 Message Buffering and Request Buffering

To defer passing the messages across the recovery line, two buffering techniques, message buffering and request buffering, are used under different circumstances.

Message buffering is to temporarily hold the messages which are to be sent across the recovery line. It is used only for the small messages which have already been copied into communication buffers but have not been posted to the network. These messages are buffered in message queues according to their destinations, and will be sent in the original order when the communication to the destination is allowed.

However, buffering all the message content is very expensive. Therefore, in our design, the request buffering is used in all possible cases, i.e. for large messages and the small messages which have not yet been copied to a communication buffer. In most MPI implementation, there are internal data structures to carry the communication requests from user applications to the bottom layer (network driver). Request buffering is to keep the communication requests in ‘incomplete’ state and buffer them in queues so that they can be processed by bottom layer later. But for zero-copy rendezvous protocol, since we allow the checkpoint can happen in middle of the protocol, the progress information is also kept with the request.

3.3.4 Asynchronous Progress

Asynchronous progress is very important for inter-group coordination. Unlike the processes in the checkpointing group, which are focusing on coordination and checkpointing, processes in other groups are potentially busy with computation. If they do not make progress in inter-group coordination a timely manner, the checkpointing processes will suffer a large coordination delay.

Therefore, when a process is entered the passive coordination state, meaning that another group is taking the checkpoint, it will temporarily activate a helper thread which will call progress engine to make progress if the progress engine has not been called for a certain amount of time. The helper thread has negligible overhead but can guarantee the timely progress of inter-group coordination.

4 Performance Evaluation

In this section, we analyze the benefits of group-based checkpointing in terms of reduction in checkpoint delay. First, we introduce the metrics and characterize two important parameters which can affect the performance impact for checkpointing to MPI applications. Then, we analyze the effects of these parameters using micro-benchmarks. At last, we describe the experimental results from High Performance Linpack (HPL) [3] benchmark.

4.1 Experimental Platform

We implemented group-based checkpointing based on MVAPICH2-0.9.8. The experiments are conducted on an InfiniBand cluster of 36 nodes, with 32 compute nodes and 4 storage nodes. Each compute node is equipped with dual Intel 64-bit Xeon 3.6GHz CPUs, 2GB memory, and Mellanox MT25208 InfiniBand HCA. Each storage node is equipped with dual AMD Opteron 2.8 GHz CPUs, 4GB memory, and Mellanox InfiniBand MT25208 HCA. The operating system used is Redhat AS4 with kernel 2.6.17.7. The file system used is PVFS2 on top of local SATA disks. The network protocol used by PVFS2 is TCP on top of IP over IB (IPoIB). To avoid the contention of file accessing through the network to get stable performance, we use one process per node configuration. The base performance of the file system is shown in Figure 1.

4.2 Performance Metrics

We use three metrics in this paper to measure the performance of group-based checkpointing, defined as follows:

- **Effective Ckpt Delay**: The delay in execution time caused by taking one checkpoint during the application’s execution. It is the most important metric, which reflects the effective performance impact of a checkpoint to the application.
- **Individual Ckpt Time**: The delay observed by each individual MPI process when taking a checkpoint, which reflects the down time of each individual process in the middle of execution.
- **Total Ckpt Time**: The total time from the point when a checkpoint request is issued to the point when all processes have finished taking their checkpoints.

For regular coordinated checkpointing, since all the processes take their checkpoints at the same time, we have:

Effective Delay = Individual Time = Total Time

However for group-based coordinated checkpointing, since processes take checkpoints group by group, we have:

Individual Time = Total Time / Number of Groups

Ideally, the effective checkpoint delay should be the individual checkpoint time. Due to the synchronization and dependencies, however, MPI processes can not always make full progress when some process group is taking a checkpoint. Therefore, in practice:

Individual Time $\leq$ Effective Delay $\leq$ Total Time

Note that in group-based checkpointing the total time only reflects the *worst case* application impact.

There are several parameters affecting the effective checkpoint delay. In the remaining parts of this section, we focus on two most important parameters: checkpoint group size and issuance time of the checkpoint request.

4.3 Evaluation with Micro-benchmarks

To evaluate the benefits of group-based checkpointing, we design a set of micro-benchmarks to simulate communication patterns of MPI applications. Since we are mainly concerned with how processes are synchronized with each other and how they make progress, we use an abstracted model. As depicted in Figure 5, in our benchmarks, processes are organized in smaller groups, and during execution, processes communicate within group more frequently and synchronize globally less frequently. Note that different frequencies are used in different experiments to measure different cases, and the checkpoint file size in micro-benchmarks is configured to be 180MB per process, which is sufficiently large to show the trend.

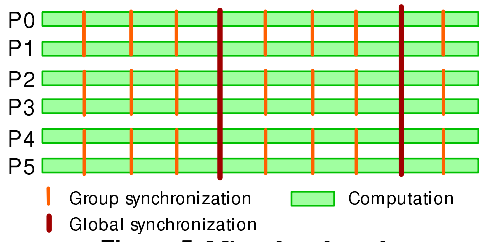

Figure 5. Micro-benchmarks

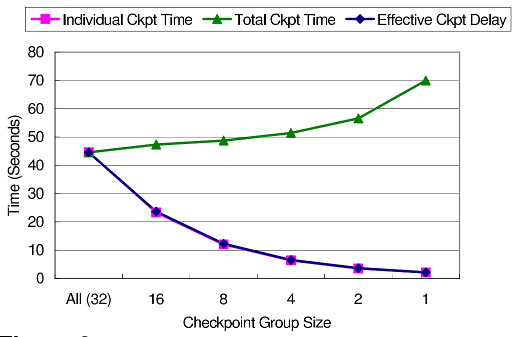

Figure 6. Ideal Case of Group-based Checkpointing

Figure 6 shows the ideal case of group-based checkpointing where the application is embarrassingly parallel, such that all processes only compute individually and make progress by themselves without synchronization. It is clear that the Effective Delay is almost same as the Individual Ckpt Time. Since the file accessing time is the dominant factor here, well above 90%, the Individual Ckpt Time approximately reduces by half every time when the checkpoint group size is reduced by half.

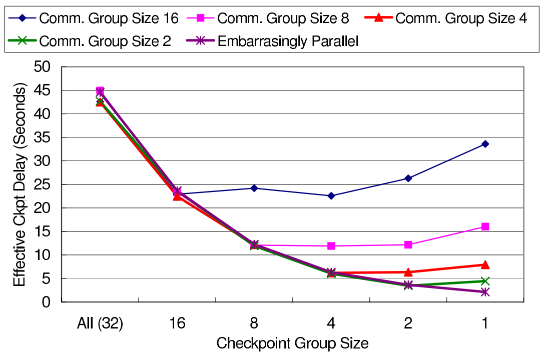

Figure 7. Checkpoint Group Size and Effective Delay

Figure 7 shows the impact of different checkpoint group sizes on the effective checkpoint delay. To highlight the impact of checkpoint group size, in this experiment, MPI processes communicate only within a group using blocking MPI calls continuously, which effectively synchronizes them in groups. The communication group size varies from 16 to 2. From the results, we observe that when the checkpoint group covers one or more communication group, the Effective Ckpt Delay is still same as the Individual Ckpt Time, but when the checkpoint group size reduces to smaller than the communication group size, Effective Ckpt Delay remains at the same level, or even increases when the checkpoint group size is very small, such as 2 or 1. This is mainly because that the aggregated bandwidth for a small number clients is not optimal, and the total storage accessing time for a communication group increases.

These results indicate that the appropriate checkpoint group size should be chosen according to the communication group size to better realize the benefits of group-based checkpointing, however, noticeable benefits still can be achieved even with sub-optimal checkpoint group sizes.

The issuance time of checkpoint request, i.e. checkpoint placement time, is also an important parameter. Although there are many issues related to checkpoint placement, here we only focus on its effect on group-based checkpointing. In this experiment, we set both the checkpoint group size and the communication group size to be 8, and enforce a global synchronization using MPI_Barrier every minute. As shown in Figure 8, the Effective Ckpt Delay lies in between the Individual Ckpt Time and Total Ckpt Time. When the checkpoint is placed closer to the synchronization line, the Effective Ckpt Delay is larger, closer to the Total Ckpt

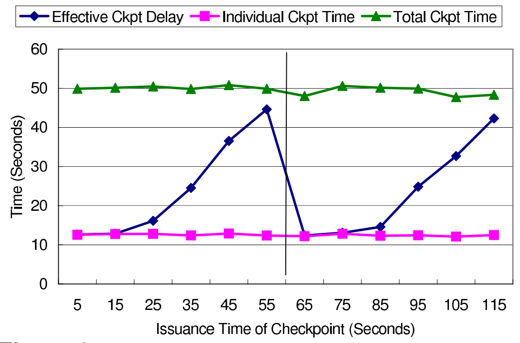

Figure 8. Checkpoint Placement and Effective Delay

Time. It is because that in this case, the process groups which finish their checkpoint earlier can not progress across the global barrier to the next phase of execution without violating the semantics of barrier, as illustrated in Figure 9. However, this negative effect can be avoided if the application is designed to be skew-tolerant.

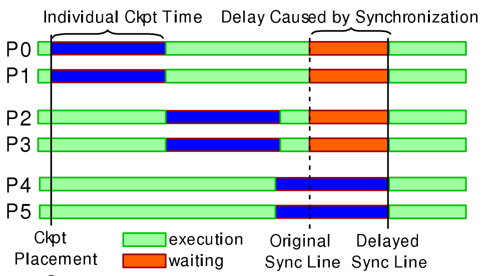

Figure 9. Additional Checkpoint Delay Caused by Synchronization

In Figure 10, the effects of both checkpoint group size and issuance time of checkpoint are combined. In this experiment, MPI processes communicate in groups of 4 processes for every 30 seconds, and synchronize globally for every 2 minutes. And we take checkpoints using different checkpoint group sizes for three representative time points:

- A (155 Second): Far from both global synchronization and far group synchronization.
- B (95 Second): Relatively close to global synchronization.
- C (145 Second): Far from global synchronization, but very close to group synchronization.

We observe that for time point A, significant reductions are shown in Effective Ckpt Delay for using smaller group size. For the time point C, the benefits are also very large, but reducing the checkpoint group size to smaller than communication group size does not result in less delay. And for the time point B, the overall benefits are only limited by the global synchronization before the effect of checkpoint group size is shown.

From the results micro-benchmarks, we can see that both checkpoint group size and issuance time of the checkpoint

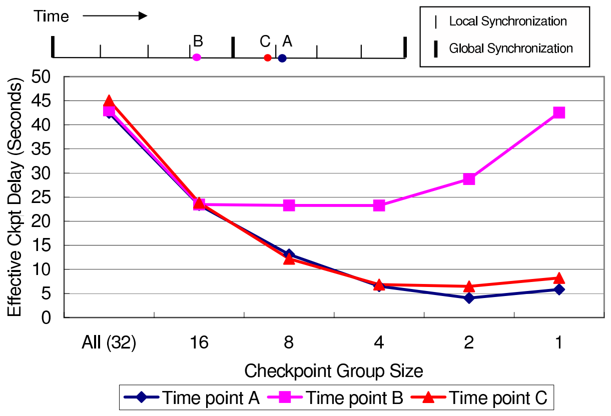

Figure 10. Effective Delay in Combined Cases

request are important parameters, but they affect the Effective Ckpt Delay in different ways. And in practice, they should be chosen according to applications communication characteristics.

4.4 Evaluation using High Performance Linpack

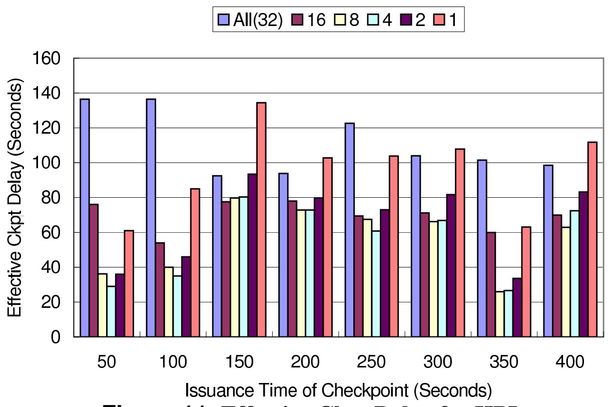

Figure 11. Effective Ckpt Delay for HPL

For most applications, the communication patterns are not as simple as in the micro-benchmarks described above. However, communication patterns of many applications can be characterized similarly. In this section, we describe the experimental results for High Performance Linpack (HPL) [3] benchmark, which is to solve a dense linear system on distributed-memory computers. The matrix data are distributed to a two-dimensional grid of processes, and processes mostly only communicate in the same row or same column. In our experiments, we choose a 8×4 configuration with a larger block size. Thus the communication group size is effectively 4.

In the experiment, we choose 8 time points evenly distributed across the execution time, and measure the Effective Ckpt Delay for different checkpoint group sizes. Figure 11 shows the detailed experimental results, from which we observe that in general the Effective Ckpt Delay in the cases with group size 2, 4, 8, or 16 is noticeably less than the regular checkpointing case, where the group size is 32, with up to 78% reduction. Average reductions for all time points with group sizes 2, 4, 8, and 16 are 37%, 46%, 46%, and 35% respectively. However, with group size 1, the Effective Ckpt Delay is not reduced much, and in some cases becomes even more than the regular checkpointing case. That is expected because the processes are not able to make progress individually and the full bandwidth provided by parallel file system is not fully utilized. Note that the delays are different for different time points even in regular checkpointing case. That is because during the execution, the total size of temporary buffers is different, and that results in a relative large variation of checkpoint file size.

Figure 12 shows the average checkpoint delay with respect to different checkpoint group sizes, with a vertical line indicating the maximum and minimum delay for each group size. We can clearly see that the checkpoint group sizes 4 and 8 give best performance. These results match the configuration, 8 $\times$ 4 processes, which we use in HPL experiments.

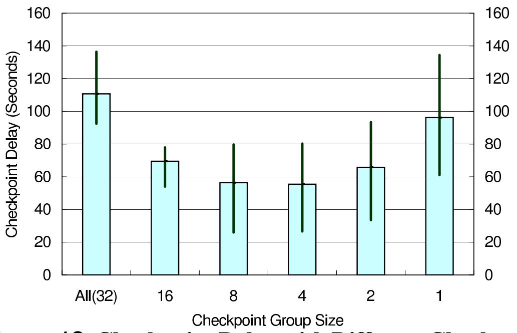

Figure 12. Checkpoint Delay with Different Checkpoint Group Sizes

5 Related Works

Many efforts have been carried out to provide fault tolerance to MPI programs. In recent years, the MPICH-V team [1] has developed and/or evaluated several roll-back recovery protocols, including both uncoordinated checkpointing with message-logging protocols, such as V1 [5], V2 [6], Vcausal [7], and coordinated checkpointing protocols, such as Vcl [8] based on Chandy-Lamport Algorithm [10], and Pcl [11] based on the blocking coordinated checkpointing protocol. FT-MPI [14] has extended the MPI specification to provide support to applications to achieve fault tolerance on application level. LA-MPI [4] enables data reliability checking and provides network fault tolerant by using multiple network devices. LAM/MPI [24] has incorporated checkpoint/restart capabilities based on Berkeley Lab's Checkpoint/Restart (BLCR) [12] to checkpoint MPI program, which also uses the blocking coordinated checkpointing protocol. Their efforts have been recently incorporated in OpenMPI project with an extended framework [17]. In our earlier work [15], we have proposed a framework to checkpoint MPI programs over InfiniBand using a similar blocking coordinated checkpointing protocol.

In this paper, we extended the blocking coordinated checkpointing design to take group-based checkpoints, which reduces the effective checkpoint delay and improves the scalability of checkpointing for MPI. The group-based checkpointing differs from uncoordinated checkpointing and non-blocking Chandy-Lamport coordinated checkpointing in the sense that a consistent global checkpoint is formed by the individual checkpoints without message-logging at any time. This is a very critical feature for clusters using high performance interconnect such as InfiniBand, where message logging can potentially impose a large overhead. Observations for the noticeable message logging overhead in checkpointing MPI on high speed network has also been reported recently by in [11].

Another approach to deal with the storage bottleneck is incremental checkpointing. Recently, a low-overhead, kernel-level checkpointer called TICK [16] has been designed for parallel computers with incremental checkpointing support. We believe that group-based checkpointing mechanism can be combined with incremental checkpointing techniques to further reduce the checkpointing overhead.

6 Conclusions and Future Work

In this paper, we have presented a design of group-based checkpointing as an extension to the coordinated checkpointing protocol to improve scalability. By carefully scheduling the processes in an MPI job to take individual checkpoints group by group, the group-based checkpointing alleviates the storage bottleneck and reduces the checkpointing delay observed by every process. A prototype implementation has been developed based on MVAPICH2, and a detailed performance evaluation has been carried out using micro-benchmarks and HPL on an InfiniBand cluster. The experimental results demonstrate that group-based checkpointing can reduce the effective delay for checkpointing significantly, up to 78% for HPL benchmark on 32 nodes.

For future research, we plan to study in more depth on two important factors affecting the checkpoint delay, checkpoint group selection and checkpoint placement, on larger platform. We also plan to study how to combine the incremental checkpointing techniques to further reduce the checkpoint delay.

Acknowledgements

We would like to express our sincere gratefulness to Dr. Paul Hargrove from LBNL for helpful advices and discussions about BLCR.

References

[1] MPICH-V Project. http://mpich-v.lri.fr.

[2] MPICH2, Argonne. http://www-unix.mcs.anl.gov/mpi/mpich2/.

[3] A. Petitet and R. C. Whaley and J. Dongarra and A. Cleary. http://www.netlib.org/benchmark/hpl/.

[4] R. T. Aulwes, D. J. Daniel, N. N. Desai, R. L. Graham, L. D. Risinger, and M. W. Sukalski M. A. Taylor, T. S. Woodall. Architecture of la-mpi, a network-fault-tolerant mpi. In *Proceedings of Int'l Parallel and Distributed Processing Symposium*, Santa Fe, NM, April 2004.

[5] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Magniette, V. Néri, and A. Selikhov. MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes. In *IEEE/ACM SuperComputing 2002*, Baltimore, MD, November 2002.

[6] A. Bouteiller, F. Cappello, T. Hérault, G. Krawezik, P. Lemarinier, and F. Magniette. MPICH-V2: a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In *IEEE/ACM SuperComputing 2003*, Phoenix, AZ, November 2003.

[7] A. Bouteiller, B. Collin, T. Hérault, P. Lemarinier, and F. Cappello. Impact of event logger on causal message logging protocols for fault tolerant MPI. In *Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS)*, Denver, CO, April 2005.

[8] A. Bouteiller, P. Lemarinier, T. Hérault, G. Krawezik, and F. Cappello. Improved message logging versus improved coordinated checkpointing for fault tolerant MPI. In *Proceedings of Cluster 2004*, San Diego, CA, September 2004.

[9] R. Butler, W. Gropp, and E. Lusk. Components and Interfaces of a Process Management System for Parallel Programs. *Parallel Computing*, 27(11):1417–1429, 2001.

[10] M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. In *ACM Trans. Comput. Syst. 31*, 1985.

[11] Camille Coti, Thomas Herault, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Rodriguez, and Franck Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI. In *Proceedings of ACM/IEEE SC'2006*, Tampa, FL, 11 2006.

[12] J. Duell, P. Hargrove, and E. Roman. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Technical Report LBNL-54941, Berkeley Lab, 2002.

[13] E. N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3), 2002.

[14] G. E. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pjesivac-Grbovic, K. London, and J. J. Dongarra. Extending the MPI Specification for Process Fault Tolerance on High Performance Computing Systems. In *Proceeding of International Supercomputer Conference (ICS)*, Heidelberg, Germany, 2003.

[15] Q. Gao, W. Yu, W. Huang, and D. K. Panda. Application-Transparent Checkpoint/Restart for MPI Programs over InfiniBand. In *Int'l Conference on Parallel Processing (ICPP '06)*, Columbus, OH, August 2006.

[16] R. Gioiosa, J. C. Sancho, S. Jiang, and F. Petrini. Transparent incremental checkpointing at kernel level: A foundation for fault tolerance for parallel computers. In *ACM/IEEE SuperComputing 2005*, Seattle, WA, November 2005.

[17] Joshua Hursey, Jeffrey M. Squyres, and Andrew Lumsdaine. A checkpoint and restart service specification for Open MPI. Technical Report TR635, Indiana University, July 2006.

[18] InfiniBand Trade Association. http://www.infinibandta.org.

[19] Matt Leininger. InfiniBand and OpenFabrics Successes and Future Requirements. http://www.infinibandta.org/events/DevCon2006presentations, 09 2006.

[20] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over InfiniBand with RDMA Support. In *Int'l Parallel and Distributed Processing Symposium (IPDPS '04)*, April 2004.

[21] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 1994.

[22] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/.

[23] B. Randell. Systems structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2):220–232, 1975.

[24] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing. *International Journal of High Performance Computing Applications*, pages 479–493, 2005.

[25] J. S. Vetter and F. Mueller. Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS) 2002*, Fort Lauderdale, FL, April 2002.