A Checkpoint-Optimized Storage System for HPC Applications

Sudharshan S. Vazhkudai [1], Matei Ripeanu [2], Samer Al Kiswany [2], Xiaosong Ma [3] and Greg Bronevetsky [4]
[1] *Oak Ridge National Laboratory, vazhkudaiss@ornl.gov*
[2] *University of British Columbia, {matei, samera}@ece.ubc.ca*
[3] *North Carolina State University, ma@cs.ncsu.edu*
[4] *Lawrence Livermore National Laboratory, bronevetsky1@llnl.gov*

Problem Statement: Checkpointing is an indispensable fault tolerance tool, commonly used by high-throughput applications that run continuously for hours or days at a time. Applications periodically save their own state and in the event of a failure recover by rolling-back the execution state to some previous timestep. Checkpointing, however critical it may be for applications, is pure overhead from an application standpoint, as it is time spent away from useful computation. Consider a massively parallel application running on tens of thousands of computing cores (such as the Cray XT3/4 leadership computing facility, LCF, at ORNL). Thousands of processes, performing intensive I/O, simultaneously at regular intervals can create millions of files, amounting to TBs of data and huge I/O workloads. This I/O wait time required to save the checkpoint images is the key driver for the checkpoint time and can overwhelm any storage system despite high-speed interconnects. Even at the lower end, in most clusters (hundreds of nodes) or a desktop grid (loosely connected CPU cycle aggregation system), checkpointing is cumbersome. *Thus, what is needed is a way to alleviate this I/O bandwidth bottleneck.*

Current State-of-the-art: It is common practice for jobs running on the individual nodes in a desktop grid or a cluster to checkpoint to a shared, central file server. Shared parallel file systems are usually crowded with I/O requests, have limited space and are stymied by the server-side bottleneck. Further, the hundreds of nodes in the desktop grid or a cluster—on which processes of the parallel job are running—can flood the central server with their simultaneous checkpointing I/O operations. Alternatively, cnodes can also checkpoint to their respective node-local storage. Node-local storage is dedicated and is not vulnerable to contention from network file system I/O traffic. However, node-local storage, and consequently the checkpoint data, is bound to the volatility of the compute node itself. For large-scale supercomputers without local disks on the compute nodes (such as the LCF at ORNL), local memories present an alternative to node-local storage. However, this is unreliable and does not tolerate a global failure.

An Opportunity: In light of the limitations of current checkpoint storage techniques, most prior work has focused on reducing the I/O needs of checkpointing [1-4]. This article focuses on the complimentary problem of improving storage systems to provide better support for this operation. The storage hierarchy in supercomputing centers presents a unique opportunity, making available vast amounts of under-utilized storage resources, and consequently, untapped residual bandwidth. For instance, the LCF at ORNL will potentially have hundreds of TBs of aggregate main memory. An exascale machine could have on the order of hundreds of PBs of memory (as discussed at the recent Exascale Townhall meeting). A survey of Tier 1 applications (Fusion, Climate, Astrophysics and Combustion) for the LCF suggests that most applications seldom use all the memory per core, leaving significant amounts of memory available for checkpoint storage. Furthermore, most clusters have hundreds of gigabytes of node-local storage (order of tens of TBs in aggregate) and in desktop grids similarly large amounts of disk space remain idle on the individual computers [5]. In addition, the interprocessor, cluster interconnect, LAN and WAN connectivity to these resources is rapidly increasing. Moreover, current checkpointing solutions for a desktop grid, a cluster or a massive supercomputer are already using node-local storage or local memories (as noted earlier), albeit not collectively but in isolation. *What is needed is a novel solution that brings to bear these storage resources and their collective bandwidth potential on the bandwidth crisis faced in checkpointing HPC applications.*

A Checkpoint Storage System: We propose that storage resources at all levels of the storage hierarchy be aggregated and presented as a dedicated checkpoint storage system for parallel applications (Figure 1). For instance, in a desktop grid, the storage system can even be built atop unreliable resources, much like how a computational desktop grid itself is based on an unreliable substrate. Similarly, node-local storage in clusters and local memories in a massive supercomputer can be aggregated. Aggregation of storage virtualizes access to a set of storage devices and in addition allows us to expose their collective potential in terms of capacity, I/O and network bandwidth, all of which are obtained at no additional hardware cost.

We have built a storage system based on FreeLoader [6] that is optimized for checkpointing in a desktop grid environment [7]. It stores large, immutable datasets by fragmenting them into smaller, equal-sized chunks, which are then striped onto multiple storage nodes to enhance data access rates. This enables applications to access data as if they reside on a high-performance shared file system. A dedicated manager node maintains metadata such as node status, chunk distribution map, and dataset attributes. Actual transfer of data chunks occurs directly between the storage nodes and the client in parallel. Our results indicate that a Gb/sec client can easily saturate its read/write throughput from this storage system. *We propose to extend this baseline implementation to aggregate node-local storage and local memories, while presenting it as a dedicated checkpoint storage. Such a storage can offer on the order of GBs/sec of I/O bandwidth.*

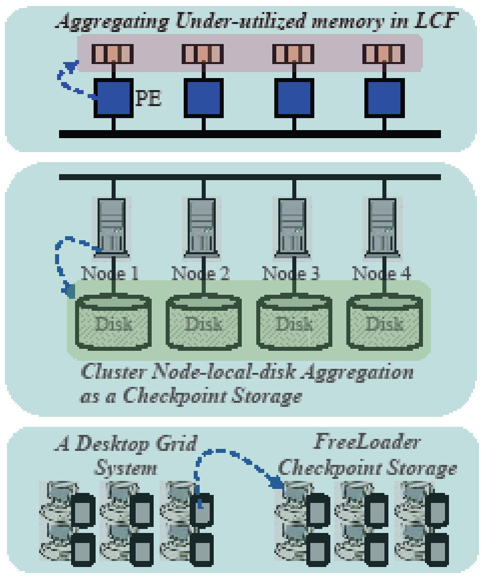
Aggregating Under-utilized memory in LCF
PE
Node 1
Node 2
Node 3
Node 4
Disk
Disk
Disk
Disk
Cluster Node-local-disk Aggregation as a Checkpoint Storage
A Desktop Grid System
FreeLoader Checkpoint Storage

Figure 1: Checkpoint storage at each level of the storage hierarchy

A checkpoint storage system should support the following desired functionality.

High-speed writes: The checkpoint storage system should provide excellent write performance and several asynchronous write optimizations so applications can expeditiously perform the checkpoint operation. As mentioned before, checkpointing is a necessary—but time consuming—operation, the results of which will only be used in the event of failure. Thus, it is desirable to provide high-speed write performance so the application can swiftly return to performing useful computation. Our aggregation approach performs parallel I/O across a network of storage devices and is thus well-positioned to offer high write bandwidth. Further, we propose to optimize the system by reducing file system overhead associated with large writes, metadata management and synchronization, thereby relaxing POSIX semantics. Thus, an application can checkpoint at a significantly higher rate than with extant solutions. Checkpointing to our prototype in a desktop grid offered an application release bandwidth of 135MB/sec [7].

Cache-like behavior: Checkpoint data is transient in nature and is usually not maintained beyond the lifetime of an application run. Unlike a regular file system, a checkpoint storage system can be aware of this characteristic and purge files based on usage or aging. This improves application performance by making available additional storage and reducing overheads due to reduced availability of RAM.

Incremental checkpointing: Often, in a checkpointing environment, subsequent snapshots of the same process generate partially similar files. This property of incremental change can be used to improve the write throughput of our aggregate storage system: blocks that do not change from one file version to another need not be written to disk again. The problem, however, is identifying common data blocks. The solution is *content addressability* in the file system. One way to implement this is to name data blocks using a *sha1* hash of the block's data, thus uniquely tying the block content to its name.

Easy to use interfaces: The storage system should provide interfaces, enabling easy integration with applications. Specialized libraries and interfaces, however optimized, cannot match the simplicity of file system interfaces. Our prototype used FUSE [8], a Linux kernel module, to provide file system-like interface.

Impact control: A key issue to address is the impact on competing workloads. For instance, amassing node-local storage or memory resources can sometimes hinder the processing of another application, using the corresponding nodes. In our previous work [9] in desktop grid storage, we throttled data access on donor nodes to control the impact on native processes. In the future we propose to investigate other schemes, such as striping data only onto the storage or memory of relatively free processing elements.

Summary: A checkpoint application-specific storage system can be constructed at all levels of the storage hierarchy and can help alleviate the I/O bandwidth bottleneck of HPC applications. The storage system can offer a competitive, low-cost alternative to checkpointing to a traditional file system, which can be freed to cater to other user needs.

References

1. Janakiraman, G.J., Santos, J.R., Subhraveti, D., and Turner, Y. *Cruz: Application-Transparent Distributed Checkpoint-Restart on Standard Operating Systems*. in *DSN*. 2005.
2. Laadan, O., Phung, D., and Nieh, J. *Transparent Checkpoint-Restart of Distributed Applications on Commodity Clusters*. in *IEEE International Conference on Cluster Computing (Cluster 2005)*. 2005. Boston, MA.
3. Bronevetsky, G., Marques, D., and Pingali, K. *Application-level Checkpointing for Shared Memory Programs*. in *ASPLOS*. 2004.
4. Bronevetsky, G., Marques, D., Pingali, K., and Stodghill, P. *Automated Application-level Checkpointing of MPI Programs*. in *PPoPP*. 2003.
5. Agrawal, N., Bolosky, W.J., Douceur, J.R., and Lorch, J.R. *A Five-Year Study of File-System Metadata*. in *5th USENIX Conference on File and Storage Technologies (FAST'07)*. 2007.
6. Vazhkudai, S.S., Ma, X., Freeh, V.W., Strickland, J.W., et al., *Constructing collaborative desktop storage caches for large scientific datasets.* 2006. **2**(3): p. 221 - 254.
7. Al-Kiswany, S., Ripeanu, M., and Vazhkudai, S.S. *A Checkpoint Storage System for Desktop Grid Computing*. in *submitted*. 2007.
8. *FUSE, Filesystem in Userspace, http://fuse.sourceforge.net/*. 2007.
9. Strickland, J.W., Freeh, V.W., Ma, X., and Vazhkudai, S.S. *Governor: Autonomic Throttling for Aggressive Idle Resource Scavenging*. in *Proceedings of 2nd IEEE International Conference on Autonomic Computing (ICAC 2005)*. 2005. Seattle, WA.