Leveraging naturally distributed data redundancy to optimize collective replication

Bogdan Nicolae
IBM Research, Ireland
bogdan.nicolae@ie.ibm.com

Massimiliano Meneghin
IBM Research, Ireland
massimim@ie.ibm.com

Pierre Lemarinier
IBM Research, Ireland
pierrele@ie.ibm.com

***Abstract*—Dumping large amounts of related data simultaneously to local storage devices instead of a parallel file system is a frequent I/O pattern of HPC applications running at large scale. Since local storage resources are prone to failures and have limited potential to serve multiple requests in parallel, techniques such as replication are often used to enable resilience and high availability. However, replication introduces overhead, both in terms of network traffic necessary to distribute replicas, as well as extra storage space requirements. To reduce this overhead, state-of-art techniques often apply redundancy elimination (e.g. compression or de-duplication) before replication, ignoring the natural redundancy that is already present. By contrast, this paper proposes a novel scheme that treats redundancy elimination and replication as a single co-optimized phase: remotely duplicated data is detected and directly leveraged to maintain a desired replication factor by keeping only as many replicas as needed and adding more if necessary. In this context, we introduce a series of high performance algorithms specifically designed to operate under tight and controllable constrains at large scale. We present how this idea can be leveraged in practice and demonstrate its viability for two real-life HPC applications.**

***Index Terms*—data resilience; high availability; replication; deduplication; collective I/O scalability; redundancy management**

I. Introduction

Scientific and data-intensive computing have matured over the last couple of years in all fields of science and industry. Their rapid increase in complexity and scale has prompted ongoing efforts dedicated to reach exascale infrastructure capability by the end of the decade. However, advances in this context are not homogeneous: I/O capabilities in terms of networking and storage are lagging behind computational power and are often considered a major limitation that that persists even at petascale [1].

A particularly difficult challenge in this context are collective I/O access patterns (which we henceforth refer to as *collective dump*) where all processes simultaneously dump large amounts of related data simultaneously to persistent storage. This pattern is often exhibited by large-scale, bulk-synchronous applications in a variety of circumstances, e.g., when they use checkpoint-restart fault tolerance techniques to save intermediate computational states at regular time intervals [2] or when intermediate, globally synchronized results are needed during the lifetime of the computation (e.g. to understand how a simulation progresses during key phases). Under such circumstances, a decoupled storage system (e.g. a parallel file system such as *GPFS* [3] or a specialized storage system such as *BlobSeer* [4]) does not provide sufficient I/O bandwidth to handle the explosion of data sizes: for example, Jones et al. [5] predict dump times in the order of several hours.

In order to overcome the I/O bandwidth limitation, one potential solution is to equip the compute nodes with local storage (i.e., HDDs, SSDs, NVMs, etc.). Using this approach, a large part of the data can be dumped locally, which completely avoids the need to consume and compete for the I/O bandwidth of a decoupled storage system. However, this is not without drawbacks: the local storage devices are prone to failures and as such the data they hold is volatile. Furthermore, the availability of the data may also suffer under concurrency due to the limited I/O bandwidth of the local storage device and/or network link.

Partner replication is a technique often used to mitigate the limitations of using local storage devices: instead of storing only one local copy of the dataset, a predefined number of extra copies are send remotely to the local storage devices of other compute nodes. Using this approach, resilience and high availability of the data can be achieved in a scalable fashion by leveraging the network bandwidth allocated to the compute nodes for communication, which is often orders of magnitude higher than the I/O bandwidth of a decoupled storage system.

However, with increasing scale, partner replication quickly hits on an important limitation: due to an increasing failure rate and an increasing number of processes potentially interested in a dataset, it is necessary to increase the replication factor in order to guarantee the same level of resilience and/or availability. As a consequence, the processes need to send more data to each other: this increases the network bandwidth contention because of larger data transfers, as well as the space utilization and I/O pressure on the local storage devices because more data is received from other processes. Ultimately, both aspects introduce an overhead that not only negatively impacts performance, but also increases operational costs (e.g. need to buy larger local storage devices).

Thus, it is important to be able to achieve a high replication factor with minimal overhead. A common strategy in this context is to apply some form of redundancy elimination (i.e., compression or deduplication) before the

replication, under the assumption that it leads to a significant reduction of replication overhead, which improves overall performance and reduces resource utilization. However, although straightforward, this two-phase approach is not optimal: first, an effort is made to eliminate data redundancy, only to reintroduce it later through replication.

II. Contribution

This poster focuses on co-optimizing redundancy elimination and partner replication at large scale in the context collective I/O operations that involve parallel writes of distributed related datasets. Inspired by several studies that confirm high data redundancy for HPC workloads (such as Meister et al. [6] and our own previous work [7]), we propose to identify any data redundancy that already exists across distributed processes and group together duplicated data into natural replicas. Using this approach, redundancy elimination and partner replication are only selectively needed when a data piece is duplicated by more and, respectively, by less than the desired replication factor. We summarize our contributions as follows:

- We introduce a series of design principles that facilitate efficient deduplication of distributed chunks, eliminating those that are remotely duplicated beyond a fixed replication factor and evenly distributing the partner replication workload for the remaining chunks among the processes to achieve load balancing. Furthermore, all processes closely coordinate to help each other out and minimize the overhead of network traffic using single-sided communication.
- We materialize these design principles in practice through a series of algorithmic descriptions that are applied to implement an I/O library that exposes a dedicated collective I/O write primitive at application level. This library is then integrated with the *AC-FTE* [8] fault tolerance runtime, which leverages the collective I/O write capabilities of the library in the context of checkpoint-restart.
- We evaluate our approach in a series of experiments conducted on the Shamrock testbed, using two representative real-life HPC applications that exhibit a high degree of redundancy in the context of checkpoint-restart. Our experiments demonstrate a large reduction of performance overhead and resource utilization compared to techniques that are not aware of naturally distributed duplicates.

III. Preliminary results

We illustrate the benefits of our approach through experiments that involve dozens of nodes equipped with local storage and a high core count, using two real-life HPC applications that need to collectively dump large amounts of data as part of checkpointing. For these two scenarios, our approach demonstrates significantly lower performance overhead and resource utilization (in terms of storage space and network traffic) both with respect to weak scalability and replication scalability. More specifically, in terms of weak scalability, for a constant replication factor of three, we demonstrate a speedup between 2.5x-2.8x compared with replication of locally deduplicated data and a speedup of 7.4x-9.8x compared with full replication. For the maximal problem size (408 processes distributed on 34 nodes) and a variable replication factor, the speedups are up to 2.3x and 8x when compared to the same approaches as above. Furthermore, as the replication factor increases, our approach can conserve more storage space and bandwidth (up to several orders of magnitude!) when compared to the other two approaches, which has important implications: reduced energy consumption, more available network bandwidth for application communications, the ability to use fast, low-capacity local storage without running out of space, etc. Also important to note is the impact of the partner selection strategy: compared to a naive solution, our load-aware strategy manages to obtain a much better load balancing, with maximal receive size per process reduced by up to 30%.

Encouraged by these results, we plan to broaden the scope of our work in future efforts. One interesting direction is to combine our approach with other redundancy mechanisms, in particular erasure codes, which would act as a replacement for replication. Furthermore, we proposed a load-aware partner selection strategy that is based on the amount of send data only. What would be interesting to explore in this context are other partner selection criteria, such as rack-awareness or topology.

Acknowledgments

The experiments presented in this paper were carried out using the Shamrock cluster of IBM Research, Ireland.

References

[1] J. Dongarra *et al.*, "The international exascale software project roadmap," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 1, pp. 3–60, Feb. 2011.

[2] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, vol. 34, pp. 375–408, September 2002.

[3] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Monterey, USA, 2002.

[4] B. Nicolae, G. Antoniu, L. Bougé, D. Moise, and A. Carpen-Amarie, "BlobSeer: Next-generation data management for large scale infrastructures," *J. Parallel Distrib. Comput.*, vol. 71, pp. 169–184, February 2011.

[5] W. M. Jones, J. T. Daly, and N. DeBardeleben, "Application Monitoring and Checkpointing in HPC : Looking Towards Exascale Systems," in *ACM-SE '12: Proceedings of the 50th Annual Southeast Regional Conference*, Tuscaloosa, USA, 2012, pp. 262–267.

[6] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A Study on Data Deduplication in HPC Storage Systems," in *SC '12: 25th International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, USA, 2012.

[7] B. Nicolae, "Towards Scalable Checkpoint Restart: A Collective Inline Memory Contents Deduplication Proposal," in *IPDPS '13: The 27th IEEE International Parallel and Distributed Processing Symposium*, Boston, USA, 2013, pp. 19–28.

[8] "Ac-fte," http://github.com/bnicolae/ac-fte.