SC14

Leveraging naturally distributed data redundancy to optimize collective replication

Bogdan Nicolae, Massimiliano Meneghin, Pierre Lemarinier @ **IBM Research – Ireland**

Context

Data storage at large scale relies heavily on **replication** as a key technique to address two important problems:

- *Data resilience:* At large scale, failures are a common occurence in a datacenter. If a storage device holds the only copy of a dataset and fails, this leads to data loss. By distributing replicas of the dataset on multiple storage devices, even if a device fails, the surviving devices can take over I/O requests.
- *High availability:* When a dataset is replicated in a distributed fashion, I/O requests can also be distributed among the replicas, which dramatically reduces I/O bottlenecks caused by concurrent accesses. This is especially important for data analytics applications that need to process different queries in parallel about the same dataset.

Challenges

Due to everincreasing data sizes, the effectiveness of replication is limited by several issues:

- *Large storage space utilization*: The total capacity necessary to store the data is multplied by the degree of replication, which is particularly important for fast devices of limited capacity (e.g. SSDs, NVMs, etc.).
- *Large I/O bandwidth utilization*: Replication involves sending copies of data pieces to other nodes, which involves multiple parallel data transfers of large sizes.
- *Performance degradation due to I/O contention*: Replication-related data transfers compete for I/O bandwidth both with other replication-related transfers, as well as the communication traffic generated by the application itself.

These issues are particularly prominent for collective I/O access patterns where all processing elements of a distributed application need to simultaneously write and replicate related datasets. We refer to this pattern as **collective replication**. It is often encoundered in situations when numerical simulations and big data analytics applications need to:

- Dump intermediate or final output data
- Checkpoint their state periodically to facilitate fault tolerance, suspend-resume, migration, etc.

Under these circumstances, there is a need for efficient collective replication that achieves high I/O performance and scalability while minimizing I/O bandwidth and storage space utilization.

Key idea in a nutshell

Several studies confirm high redundancy among data stored by HPC applications (e.g., Meister et al. [1] and our own previous work [2]). As a consequence, a straightforward idea is to attempt redundancy elimination (e.g. compression, deduplication) before replicating the datasets. However, this two-phase approach is not optimal: first, an effort is made to eliminate data redundancy, only to reintroduce it later through replication.

This work combines redundancy elimination and replication as a single optimized operation. It proposes high performance algorithms to identify and replicate only those distributed data pieces that are not already duplicated to the desired replication factor, while eliminating extra duplicates for data pieces that exceed the desired replication factor.

Assumptions

Without restricting generality, we assume that a set of distributed processes simultaneously call an I/O collective primitive DUMP_OUTPUT to store the content of (local memory region) $buffer$ on a local storage device and then replicate it to other $rep_factor - 1$ (all processes must use the same replication factor) remote storage devices. This primitive takes the form:

DUMP_OUTPUT($buffer$**,** rep_factor**)**

Design principles

Identify natural redundancy through collective inter-process deduplication. We split each $buffer$ into small fixed sized chunks that are hashed using a cryptographically strong hash function. Then, using a collective reduction we gradually gather the hash values of different processes and keep only the N most frequent. During this process, a maximum of rep_factor ranks are assigned as "owners" for each hash value designated as frequent. All processes learn about the N most frequent hash values and their owners and avoid storing and replicating any chunks for which rep_factor other owners were designated.

Load balancing by means of even rank ownership assignment. How the owners of chunks are designated is important, because if one process needs to store and replicate more chunks than another, it will cause all other processes to wait for it. Thus, a much better idea is to try to assign the ranks in such way that the overall load is evenly distributed, which speeds up the most loaded (and thus slowest) process, effectively leading to a better overall performance. We designed efficient algorithms to dynamically assign and modify ownership during the collective reduction.

Reduce unavoidable imbalance using load-aware partner selection. Chunks that did not make it in the N most frequent hash value set are assumed unqiue and thus there is only a single choice in terms of ownership. They can cause problems, because some processes may hold more such chunks than others. We call this "unavoidable imbalance". This undesirable effect can be partially mitigated by choosing where to replicate the chunks in informed fashion.

Figure 1: Naïve partner selection (left) vs. load aware partner selection (right) for a replication factor of three: first two processes send 100 chunks (illustrated by arrows), the rest send 10 chunks (not illustrated). The value of each node is the total number of received chunks. Using a rank shuffling of (1,3,4,2,5,6) spreads the load more evenly: the maximum number of received chunks is lowered from 200 (left) to 110 (right).

Low-overhead exchanges using single sided communication planning. Once each process has identified its remote partners, the exchanges of chunks between the can begin. A simple solution that gathers chunks as they arrive introduces extra overhead due to buffering and is inefficient. Ideally, each process should be able to put a chunk directly in the memory of other processes at the right offset using low overhead single sided communication channels that take advantage of RDMA. However, the problem of establishing a remote offset for each chunk replica in advance is non-trivial. We introduce a low overhead algorithm to achieve this.

How it works by example

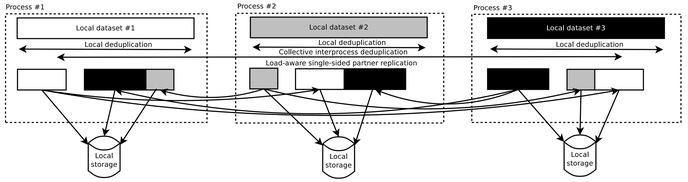

Figure 2: Three processes call the DUMP_OUTPUT primitive with a replication factor $rep_factor = 3$.

Implementation

This effort was integrated in *AC-FTE*, a run-time environment specifically designed to provide scalable checkpoint-restart functionality for tightly coupled applications, both transparently and at application level.

More details: https://github.com/bnicolae/ac-fte

Experimental setup

Platform:

- CPU: Intel Xeon X5670 (12 cores)
- Interconnect: Gigabit Ethernet
- Local storage: 1 TB HDD
- Number of nodes: 34

Applications:

- CM1: a three-dimensional HPC numerical model suitable for idealized studies of atmospheric phenomena.
- HPCCG: a simple conjugate gradient benchmark code for a 3D chimney domain on an arbitrary number of processes.

CM1: Results for 408 MPI processes (12 processes/node)

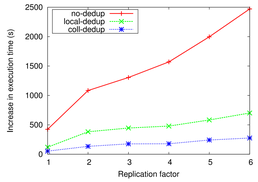

Figure 3: Increase in execution time: lower is better. Baseline is 382s.

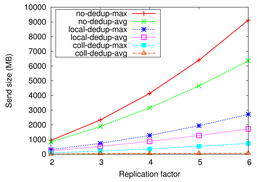

Figure 4: Amount of replicas sent per process: lower is better.

HPCCG: Results for 408 MPI processes (12 processes/node)

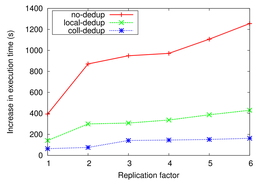

Figure 5: Increase in execution time: lower is better. Baseline is 279s.

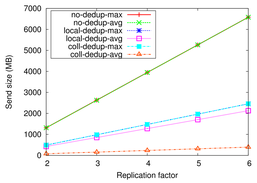

Figure 6: Amount of replicas sent per process: lower is better.

Conclusions

Experiments show:

- Speedup of 7.4x-9.8x compared with full replication
- Speedup of 2.5x-2.8x compared with replication of locally deduplicated data
- Storage space and I/O bandwidth utilization reduced by several orders of magnitude
- Load-aware partner selection reduces maximal receive size per process by 30% compared with static partner selection.

Leveraging natural data redundancy has promising potential to improve the scalability and performance of collective replication while minimizing storage space and bandwidth utilization for real-life large scale distributed applications

References

[1] Dirk Meister, Jurgen Kaiser, Andre Brinkmann, Toni Cortes, Michael Kuhn, and Julian 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.

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

Contact

Bogdan Nicolae
Web: **http://researcher.ibm.com/person/ie-bogdan.nicolae**
E-mail: **bogdan.nicolae@acm.org**