VELOC: Very Low Overhead Checkpointing System

Bogdan Nicolae, Franck Cappello
Argonne National Laboratory
USA
{bnicolae,cappello}@anl.gov

Adam Moody, Elsa Gonsiorowski, Kathryn Mohror
Lawrence Livermore National Laboratory
USA
{moody20,gonsiorowski1,mohror1}@llnl.gov

KEYWORDS

HPC, resilience, checkpoint-restart

1 INTRODUCTION

High performance computing (HPC) applications produce massive amounts of checkpointing data during their runtime, which is often used for *defensive* purposes, i.e., to employ a *checkpoint-restart* resilience strategy in case of failures. In addition, *productive* uses of checkpointing are becoming increasingly popular to facilitate analytics (offline or in-situ) or computational models that need to revisit previous states (e.g., adjoint computations).

Checkpointing is a challenging I/O pattern, because the data coming from a large number of distributed processes typically needs to be coordinated to form a globally consistent checkpoint. This generates high write concurrency that overwhelms the I/O bandwidth of the system, leading to large performance overheads and poor scalability.

In the quest to reach Exascale, many architectural trade-offs are necessary, including a high degree of parallelism and a growing gap between the compute capacity and I/O capabilities, which means less I/O bandwidth will be available per compute unit [1]. Therefore, checkpointing will be even more challenging: it needs to be performed more frequently (increasing failure rates) and it puts more pressure on the I/O bandwidth.

To avoid this issue, many applications have switched from writing checkpoints directly to a parallel file system to more advanced approaches, such as *multi-level checkpointing*. The idea is to use the faster and less reliable local storage of the compute nodes (or that of neighboring nodes) to implement "lighter" resilience levels that hold the checkpoints. This enables applications to survive a majority of failures, thereby decreasing the need to write the checkpoints as often to a parallel file system.

Despite promising potential, multi-level checkpointing faces several challenges at Exascale, for which state of art approaches provide no support or only limited support. First, the performance overhead is still significant. Much of this overhead can be masked by running resilience strategies asynchronously in the background (e.g., copy checkpoints to partner nodes or the parallel file system without blocking). However, this generates interference with the application runtime that is not well understood. To avoid interference, state of art simply uses dedicated resources (e.g., reserved cores) to implement asynchronous strategies but this may be wasteful for certain applications. Second, the storage stack is increasingly heterogeneous (deep memory hierarchies on compute nodes, burst buffers, key-value stores, etc.). Many users are not aware of the various options and, even if they are, they report difficulties in using multiple vendor-specific APIs and understanding the performance characteristics. Therefore, it is necessary to hide the complexity of heterogeneous storage from the users, yet state-of-art approaches have limited support in this regard (e.g., they consider a two-level hierarchy with node local-storage and a parallel file system). Third, as applications grow in complexity, so does the need to configure the resilience strategies and perform additional custom processing (e.g., filtering, compression). However, state of art techniques implement the multi-level checkpointing as a monolithic runtime that is not easy to change and/or extend.

SC'18, 2018, Dallas, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 VELOC: AN OVERVIEW

This poster introduces *VELOC* (Very Low Overhead Checkpointing System), an Exascale-ready checkpointing runtime that is specifically designed to address the challenges mentioned above. It aims to deliver high performance, scalability and flexibility without sacrificing ease of use.

2.1 Key features

Hidden Complexity of Storage Interaction. To address the issue of complexity of heterogeneous storage, we propose a simple API that enables each application process to declare "critical" memory regions that need to be part of a global checkpoint. When a global checkpoint needs to be taken, all processes call a collective checkpointing primitive that handles all details transparently. Using this approach, users never have to worry what types of storage are available and how to use them. Furthermore, the separation between fine-grained declarations of critical memory regions and the actual checkpoint request opens several optimization opportunities compared with writing the critical data directly into a file, such as: the flexibility of placing different memory regions on different types of storage, gathering information about memory layout and access patterns in advance to help subsequent resilience strategies, pro-active copy-on-write, etc.

Asynchronous Dynamic Multi-Level Resilience. We propose and advanced multi-level approach that is based on two key ideas. First, we take advantage of idle resources if present during application runtime to advance asynchronous resilience strategies in the background using work stealing. Using this approach, it is not necessary to dedicate resources specifically for resilience. Second, we group multiple storage options of similar resilience properties but different performance characteristics together to create dynamic levels

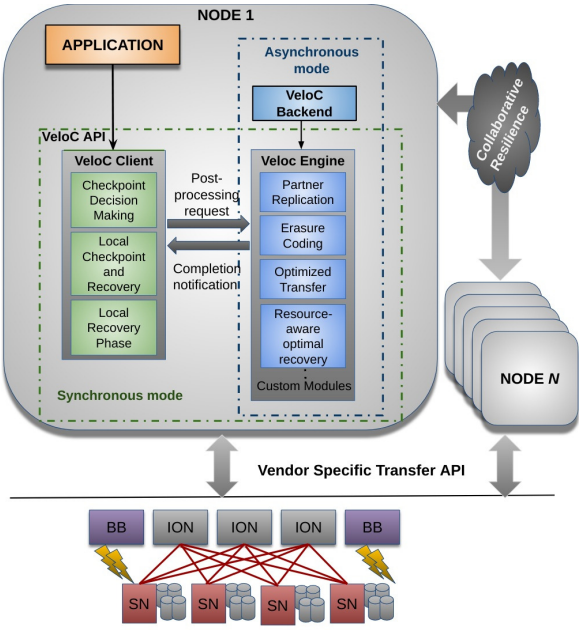


Figure 1: Architecture of VELOC (Very Low Overhead Checkpointing System)

that combine the storage space (e.g., scavenge spare memory and local storage on the nodes) and exploit synergies with other levels (e.g., apply a different I/O strategy to manage local checkpoints if they need to be flushed to external storage at the same time).

Flexibility through Modular Design. We propose a modular design that encapsulates each resilience strategy as an independent module that is part of a pipeline. Whenever a checkpoint request is issued, the pipeline will trigger each module one after another based on a pre-defined priority. Each module is individually responsible to react to a checkpoint request and can do so (or simply pass) based on its own internal state and/or the outcome of the other modules invoked earlier in the pipeline. Using this approach, a module can be activated or deactivated at runtime as needed using a simple switch. Furthermore, users can add their own modules in the pipeline and control when they are triggered by customizing their priority with respect to the other modules.

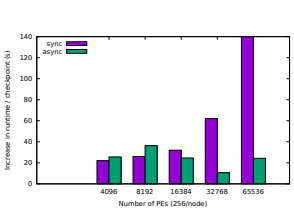
2.2 Architecture

The architecture of VELOC is depicted in Figure 1. The *client* is responsible to expose the API to the application and to save the checkpoints to local storage. The *engine* is responsible to run the module pipeline. The default modules perform post-processing on the local checkpoints, which includes both collaborative resilience (e.g., replication and erasure coding using partner compute nodes) and optimized transfer support to heterogeneous external storage (e.g., parallel file systems, burst buffers, key-value stores, etc.) using vendor APIs (where applicable). Users can freely add their own modules.

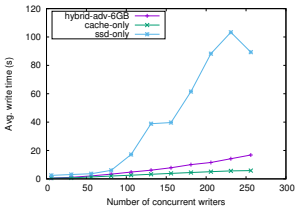
Two modes of operation are supported: *synchronous* and *asynchronous*. In the synchronous mode, the client and the engine are linked together using a trivial control plane in the same frontend library used by the application. Both the local checkpointing and the post-processing are blocking operations. In the asynchronous mode, the engine instance is created only once per node and lives in a separate active backend. All clients connect to the same active backend using a configurable control plane that is based on shared-memory or RPC libraries. In this mode, the clients do not need to wait for the engine to finish and can resume the application immediately after the local checkpoints were written and the engine was notified about their existence.

3 PRELIMINARY RESULTS

We ran preliminary experiments on *Theta*, a 11.69 petaflops pre-Exascale Cray XC40 system based on the second-generation KNL Intel Xeon Phi 7230 SKU with 512 parallel processing elements (PEs) per node. This experiments involved a custom benchmark and an HPC application (that models the heat distribution in a medium).



(a) Sync vs. Async mode: increase in application runtime per checkpoint (Lower is better).



(b) Dynamic Levels: avg. write time to local storage while flushing to the parallel file system (Lower is better).

Figure 2: Experimental results focusing on the advantages of asynchronous checkpointing and dynamic multi-level resilience

Figure 2(a) show the overhead (increase in runtime for one global checkpoint) at scale (up to 64k PEs) when saving checkpoints to local storage and flushing them asynchronously to a parallel file system (Lustre in this case) vs. the case when writing the checkpoints directly to Lustre in synchronous mode. Given a 7x speed-up, the asynchronous mode shows promising potential.

Furthermore, we also considered a scenario with many cores per node that put high I/O pressure on the local storage while the checkpoints are flushed in the background to Lustre. The results are depicted in Figure 2(b). Using as little as 6% spare RAM in addition to a local SSD, our advanced resilience strategy that employs dynamic levels is close to the case when using RAM only as local storage and up to 80% faster than using the SSD only.

REFERENCES

[1] Richard Gerber, James Hack, Katherine Riley, Katie Antypas, Richard Coffey, Eli Dart, Tjerk Straatsma, Jack Wells, Deborah Bard, Sudip Dosanjh, Inder Monga, Michael E. Papka, and Lauren Rotman. 2018. *Crosscut report: Exascale Requirements Reviews, March 9-10, 2017 - Tysons Corner, Virginia. An Office of Science review sponsored by: Advanced Scientific Computing Research, Basic Energy Sciences, Biological and Environmental Research, Fusion Energy Sciences, High Energy Physics, Nuclear Physics.* Technical Report. DoE.