IO Services on the Aurora Supercomputer - CPPR

Christopher Holguin
Intel Federal, LLC
4100 Monument Corner Drive
Suite 540
Fairfax, VA 22030
+1 720 652 5789
christopher.a.holguin@intel.com

Kalyana Chadalavada
Intel Corp.
2200 Mission College Blvd
Santa Clara, CA 95054
+1 916 356 8730
kalyana.chadalavada@intel.com

Jeffrey Olivier
Intel Federal, LLC
+1 720 652 5767
jeffrey.v.olivier@intel.com

John Carrier
Intel Corp
+1 206 407 3718
john.carrier@intel.com

ABSTRACT

Common Persistent-memory POSIX Runtime (CPPR) enables applications to leverage the fast persistent memory available on Aurora compute nodes (CNs) for I/O which enables more efficient use of compute cores. CPPR comprises 3 components - 1) sync API for developers to utilize CPPR services 2) Compute node session services (CNSS) daemon 3) file movement utilities. Using CPPR, applications can 1) take advantage of the node-local (NL) file system backed by persistent memory hardware to reduce interaction with the global file system, 2) move files into and out of the NL file system asynchronously, and 3) create fault-tolerant checkpoint data with SCR and FTI, which utilize CPPR services.

1. INTRODUCTION

CPPR is a project funded by a Non-Recurring Engineering (NRE) contract with CORAL (Collaboration of Oak Ridge, Argonne, and Lawrence Livermore National Labs). In this document, an overview of the architecture of the CPPR project will be presented, including why CPPR is needed/relevant, descriptions of the required deliverables and proposed solutions which are currently being developed, and will close with potential future uses for CPPR.

2. Problem/Opportunity

There are several problems to be addressed with the Aurora IO stack: 1) there is a new tier in the memory hierarchy. 2) Aurora will have 50k+ compute nodes. This means that any data stored on a given local node has a higher likelihood of loss due to decreased Mean Time To Interrupt (MTTI). This is simply the nature of increasing hardware complexity: as the Compute Node (CN) and component count goes up or becomes more complex, MTTI worsens. 3) The standard memory hierarchy constraints are still in place (e.g. accessing the global file system is still orders of magnitude slower than local storage). CPPR has been designed to address each of these problems.

3. Architecture

CPPR consists of three essential components: the client side library and file movement utilities which the end users interact with, the CPPR Daemon, which is being called the Compute Node Session Services Daemon (CNSS), and finally some pre-existing projects that will be ported to the CPPR APIs.

Beyond the essential CPPR components is the architecture of Aurora itself. Aurora will contain >7PB of DRAM and node local persistent memory combined. This huge combination of fast, node local memory is what CPPR is designed to take advantage of: CPPR will provide a node local namespace to access this memory, and all of the APIs, utilities, and services which CPPR implements are at their core designed to enable application developers to take full advantage of this memory.

3.1 File movement APIs and Utilities

CPPR will implement a client-side library in C and FORTRAN. These APIs can be grouped into 3 classes: 1) basic file movement, 2) group file movement, 3) cooperative get.

All client side API calls will be function-shipped via Mercury RPCs [2] to the CPPR Daemon. This enables any interaction with the CPPR file movement services and APIs to be asynchronous with respect to the application.

3.2 CPPR Daemon (CNSS)

The CNSS is the service which will implement the CPPR APIs and actually performs the file movement operations. This process will be running on each CN and can be pinned to a specific core in order to reduce jitter to the application.

3.2.1 Cooperative Get and Cooperative Cache

Cooperative Get and the Cooperative Cache are new terms coined by CPPR. Essentially, these leverage ideas from SPINDLE [3] to reduce interaction with the global file system when loading such files as share libraries.

3.3 SCR/FTI

SCR and FTI are libraries developed at LLNL and ANL, respectively, and they enable application developers to manage checkpoint data on multiple storage tiers and harden data against loss. CPPR is tasked with porting both of these projects to the CPPR APIs to take advantage of the performance benefit that comes with asynchronous file transfers, which CPPR enables.

4. Extended Scope/Beyond Aurora

CPPR has both extended scope work written into the contract, and potential value beyond Aurora.

The extended scope features include: a relatively basic shared-file API and semantic in which the file stripes are distributed among the nodes in a given session and are stored on node local storage. Additionally, CPPR has proposed exposing a redundancy API which would allow SCR and FTI to further offload computation to

the CNSS asynchronously, thus gaining a further performance benefit.

Finally, the potential for CPPR beyond Aurora is in Intel's Scalable Systems Framework (SSF). Intel wants to enable its customers to receive a tailored software package for any level of HPC environment, whether it be top 500, or small-medium business. CPPR may become a component in the SSF after Aurora.

5. Acknowledgment

A special thanks to Eric Barton for his architectural contributions to CPPR.

6. References

[1] NERSC, "File Storage and IO on Cori," [Online]. Available: http://www.nersc.gov/users/computational-systems/cori/file-storage-and-i-o/.

[2] Argonne National Laboratory and The HDF Group, "Mercury," [Online]. Available: https://mercury-hpc.github.io/.

[3] "spindle," [Online]. Available: https://github.com/hpc/Spindle.

[4] Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303-312, 2006.

[5] "SCR," [Online]. Available: https://github.com/LLNL/scr.

[6] "FTI," [Online]. Available: https://github.com/leobago/fti.