Profiling Composable HPC Data Services

Srinivasan Ramesh* Philip Carns† Robert Ross† Shane Snyder† Allen Malony*
*University of Oregon {sramesh, malony}@cs.uoregon.edu
†Argonne National Laboratory {carns, rross, ssnyder}@mcs.anl.gov

I. Abstract

Storage hardware on today's HPC platforms is multi-layered, where each layer involves different technologies, performance characteristics, cost, latency, and bandwidth constraints. This heterogeneity, coupled with increasing on-node parallelism, complicates the task of managing and optimizing I/O performance. The traditional MPI-based parallelism is increasingly supplemented by large-scale task parallelism involving intensive data-analysis and machine learning routines. It is natural to expect the I/O needs of such applications to be different from traditional HPC simulation workloads.

The Mochi project [1] is an attempt to structure and catalyze the development of customized HPC data services. The key software design paradigm lies in building microservices and composing them to meet applications' needs. The Mochi project focuses on the development of microservice *building blocks* and composing them via Mercury [2], an RPC framework that is optimized for RDMA-based HPC systems. These microservices rely on Argobots [3] for managing concurrency within a node.

A. Performance Tools for Data Services

Data Services built using Mochi have two basic questions that need to be answered as far as performance is concerned:

- What does service performance mean and how can it be measured?
- How can service performance be monitored and analyzed (online) so as to generate a better service configuration in response to changing application needs?

Traditional parallel performance analysis tools for HPC typically rely on the use of a popular parallel performance model such as MPI or OpenMP. RPC-based service requests can span microservices across many nodes, and traditional tools are not designed with this assumption in mind. Thus, there is a need to design and develop new tools that support the RPC paradigm.

B. Breadcrumb Profiling: A Quick Summary of Performance

We have designed a system that collects and presents a profile of the timing of various microservice operations in the data service. A breadcrumb in this context refers to an RPC *callpath profile*. For example, if a Mochi microservice operation C is invoked from two other microservice operations A and B, the breadcrumb $\mathrm{A} \rightarrow \mathrm{C}$ represents a separate callpath from the breadcrumb $\mathrm{B} \rightarrow \mathrm{C}$. Each microservice instance keeps track of its RPC callpath ancestry and forwards this information along the request path (metadata propagation). Breadcrumb timing and count information are collected on both the origin (client) and target (server) side. Further, this information is kept on a per-origin and per-target instance basis. This enables a quick summary of the state of load-balance in the service for various operations. Figure 1 depicts the top-5 breadcrumbs in the execution of the *Mobject* data service, sorted by the descending order of cumulative time across all target (server) and origin (client) instances.

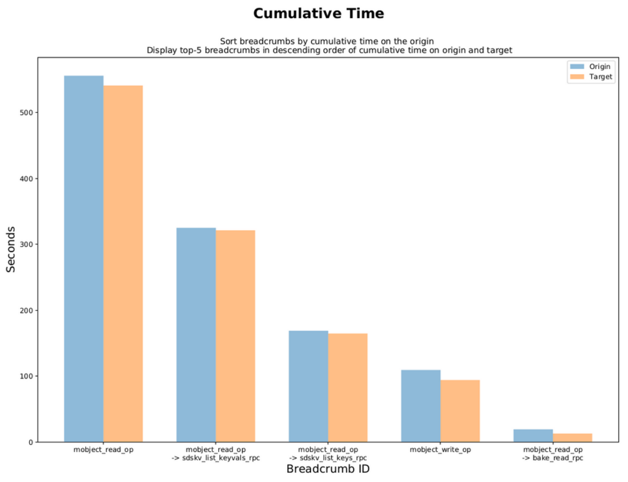

Fig. 1. Breadcrumb Profile: Cumulative Time

At the moment, instrumentation points inside microservices are limited to client send/receive and server send/receive. We intend to extend the level of instrumentation in the future. At these points, the state of Argobot work-queues is collected and their watermarks are summarized in the profile. We are working on a tracing infrastructure that would give us fine-grained information about individual microservice request latencies which we will use to correlate with memory usage, CPU load, and other Mercury-specific metrics.

References

[1] M. Dorier *et al.*, "Methodology for the rapid development of scalable hpc data services," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2018.
[2] J. Soumagne *et al.*, "Mercury: Enabling remote procedure call for high-performance computing," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 2013.
[3] S. Seo *et al.*, "Argobots: A lightweight low-level threading and tasking framework," *IEEE Transactions on Parallel and Distributed Systems*, 2017.