Profiling Composable HPC Data Services
WIP@PDSW, 2019

Srinivasan Ramesh
Allen D. Malony
University of Oregon

Philip H. Carns
Robert Ross
Shane Snyder
Argonne National Laboratory
Data Services: Managing Heterogeneity and Change

- Difficult to build custom data services efficiently:
  - Lots of moving parts
  - Need to dynamically adapt to changing application patterns
- Debugging performance problems is hard:
  - Numerous attempts at debugging microservices: Dapper@Google, Stardust, X-Trace, etc
  - We take inspiration from these
Mochi: Composable Data Services

Mochi data services are built by composing **microservices**:  
- RPC for control  
- RDMA for data movement

Mochi’s building blocks:  
- Mercury, Argobots, Margo

Performance Analysis in Mochi:  
- Build performance analysis capability directly into Mochi:  
  - Available out-of-the-box!

Mobject service: An object store

*Image credits: Matthieu Dorier, Argonne National Laboratory*
Mochi: Performance Analysis

MObject service: Call path profiling

Cumulative Time

- We track the time spent in various call paths within the service:
  - A->C->D is a different call path from B->C->D
- **Key idea:** Each microservice stores and forwards RPC call path ancestry
- Time, call count, resource-level usage statistics updated at four instrumentation points: Client send/receive, Server send/receive
- What performance questions do we hope to answer?
Call Path Profiling: Detecting Load Imbalance

- Performance question: For a given call path, what is the distribution of call path times and counts in origin/target entities?

\texttt{mobject\_read\_op}: Raw distribution of call times across all origin (client) entities

\begin{itemize}
  \item Overloaded server: Large variation in response time
  \item Multi-threaded server: Better read perf. and response time
\end{itemize}
Tracing: Detecting Resource-Level Inefficiencies

- Margo servers spawn a new Argobot User-Level-Task (ULT) for every incoming RPC request
  - Size of pool of tasks waiting to run is a measure of load and responsiveness of system
- We perform request tracing at the 4 instrumentation points previously described:
  - We collect Argobot pool size info, memory usage along request path
  - This enables correlation of call path behaviour with resource usage on node

`mobject_read_op`: Max number of pending Argobot ULT’s along request path

Overloaded server: Pending tasks are stacking up

Multi-threaded server: Reduction in number of pending tasks