INFLOW 2016

Argonne NATIONAL LABORATORY

ENABLING NVM FOR DATA-INTENSIVE SCIENTIFIC SERVICES

PHILIP CARNS
John Jenkins
Sangmin Seo
Shane Snyder
Robert Ross
(Argonne National Laboratory)

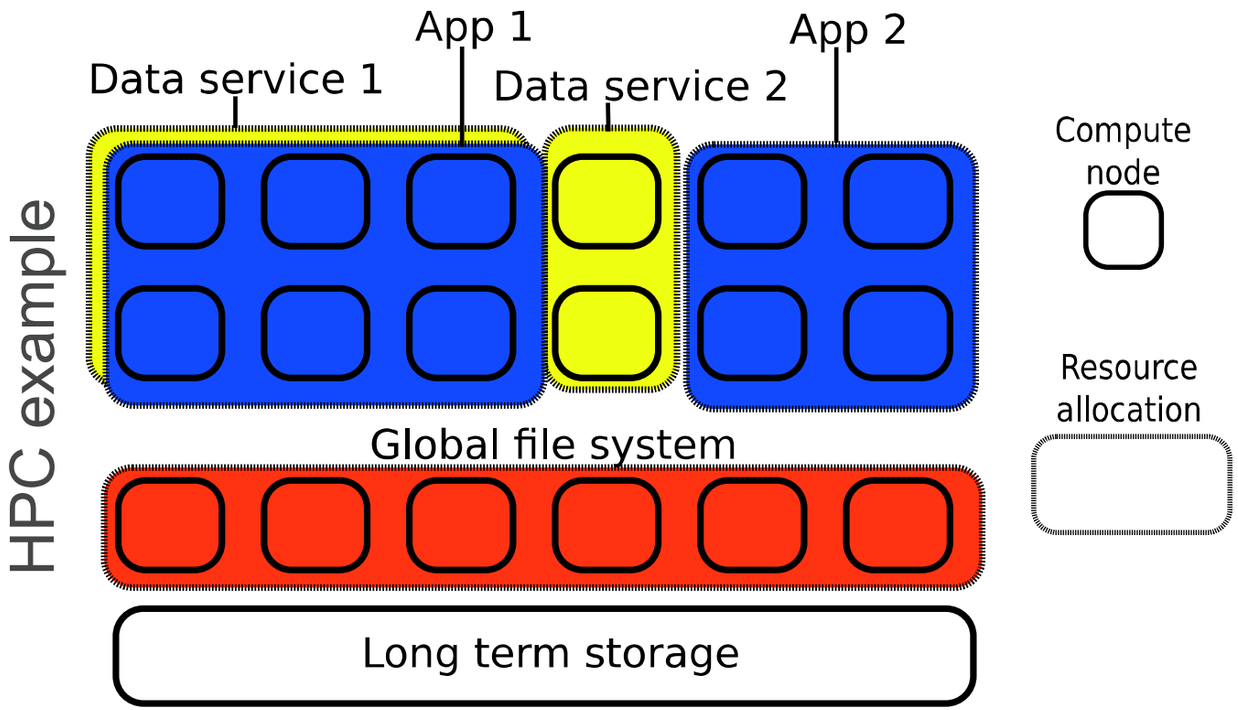
Chuck Cranor
(Carnegie Mellon University)

Scott Atchley
(Oak Ridge National Laboratory)

Torsten Hoefler
(ETH Zurich)

November 1, 2016 Savannah GA

DATA-INTENSIVE SCIENTIFIC SERVICES



- Blue: applications
- Yellow: transient data services
- Red: persistent global file system

Transient data services allocated on demand to provide domain-specific functionality and semantics

- Use in-system storage devices
- Overlap with application resources in some cases
- Provision appropriate resources for the task at hand
- Supplement, not replace, conventional “global” file system

Specialized data services are already here!

Examples from HPC and cloud environments

	Provisioning	Comm.	Local Storage	Fault Mgmt. and Group Membership	Security
DataSpaces *Data store and pub/sub.*	Indep. job	Dart	RAM (SSD)	Under devel.	N/A
DataWarp *Burst buffer mgmt.*	Admin./ sched.	DVS/ Inet	SSD	Ext. monitor	Kernel, Inet
FTI *Checkpoint/restart mgmt.*	MPI ranks	MPI	RAM, SSD	N/A	N/A
Kelpie *Dist. in-mem. key/val store*	MPI ranks	Nessie	RAM (Object)	N/A	Obfusc. IDs
HDFS * *Dist. file system for MapReduce*	EMR or similar	TCP/IP	Disk, SSD	Zookeeper	Key encryption
Spark * *Data processing*	EMR or similar	TCP/IP	RAM	Zookeeper	Key encryption

* Persistent services such as HDFS, Spark, and others can be provisioned on-demand in cloud environments

WHAT'S NEXT?

The evolution of data-intensive scientific services

- Support emerging hardware
  - NVM (byte-addressible non-volatile memory) to accelerate data access
  - See Aurora (ANL), Summit (ORNL), and others
- Lower the barrier to entry
  - Avoid building new services from the ground up
  - Reuse existing toolkit/framework/library for critical functionality
  - "Microservice" model: lightweight *composable* building block services
  - Performance portability
- Example microservices: key/value storage, object storage, group membership, replication, namespace management
- Example composed scientific service: "Genomics Query Service"

CASE STUDY:
AN OBJECT STORAGE MICROSERVICE

Argonne NATIONAL LABORATORY

GOAL

Low-latency, high-throughput access to *distributed* NVM, using an object storage API, that embodies the microservice concept for data-intensive science

- Why not just use local NVM devices through local APIs?
  - Ability to share data across tasks or ensembles
  - Support imbalanced workloads
  - NVM devices may not be present on every node
- Challenges for remotely accessible object storage service:
  - **High concurrency** is the norm (thousands of application processes)
  - **Network portability** for big data and HPC environments
  - **Latency** (software overheads won't be masked by slow disks)
  - Being a **"friendly neighbor" to co-located applications** and services

COMPONENTS USED IN PROTOTYPE*

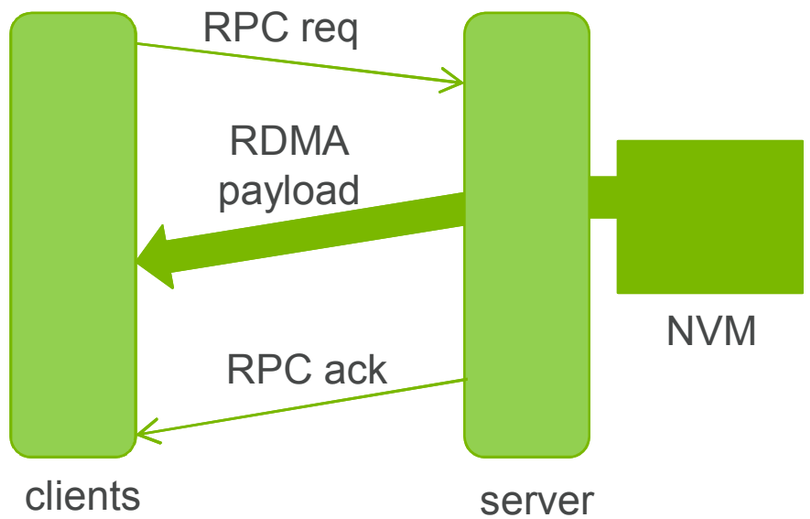
- Local NVM access: **libpmemobj** (Intel)
  - Object abstraction for local NVM access
- Network abstraction: **Common Communication Interface** (ORNL)
  - High-performance network abstraction with concise primitives
- RPCs: **Mercury** (HDF Group and ANL)
  - HPC-oriented RPC framework with RDMA path for data
- Concurrency: **Argobots** (ANL)
  - User-level threading and tasking
- Bindings: **Margo** and **abt-snoozer** (ANL)
  - Maps communication to threads with custom scheduling

*See URLs for source code at end of presentation

CHALLENGE: HIGH CONCURRENCY

How do we avoid overwhelming services under large-scale, highly-concurrent workloads?

- Put services in charge of coordinating the following at shared resource (contention) points:
  - Flow control and memory consumption
  - Coherency
  - Staging to/from slower storage tiers
  - Access rights
- To accomplish this:
  - Clients initiate access using explicit RPCs
  - Servers drive payload transfers with RDMA
  - Can we still get reasonably low latency with this model?
- Trade off latency for ability to optimize concurrent access



CHALLENGE: NETWORK PORTABILITY

Can we support a variety of networks effectively?

- Anecdote: Argonne (ALCF) machine room has production systems with the following transports and fabric:
  - TCP/IP (Ethernet)
  - Verbs (multistage InfiniBand)
  - uGNI (dragonfly)
  - PAMI (torus)

- Can't afford to tune all the way to metal while supporting wide range of big data and HPC deployments
- Mercury RPC API is network agnostic, provides RDMA path for bulk data transfer
  - RDMA is emulated on networks that don't support it
- CCI communication library (used within Mercury as an optional transport) supports TCP, Verbs, uGNI, shared memory, and others

CHALLENGE: LATENCY

How do we prevent software from becoming the bottleneck for high-performance storage devices?

- All threads are **user-level threads** implemented by Argobots
  - Spawn with low overhead (for incoming requests)
  - Schedule and context switch cooperatively (when blocked on I/O resources)
  - No OS-level context switching
- Make path from network to storage as direct as possible
  - Server drives RDMA directly to libpmemobj regions when appropriate
  - Memory copy is more efficient than memory registration in some cases, though. More on this later.
- Avoid extraneous serialization by eschewing POSIX semantics and name-space rules when they aren't needed

CHALLENGE: FRIENDLY CO-LOCATION

How do we minimize interference when microservices run on the same node as an application?

- No busy-spinning (at least by default) for network events; requires considerations within multiple components:
  - CCI exposes file descriptor to block callers until events are ready
  - Mercury decouples event management from network “progress”
  - Abt-snoozer scheduler for Argobots gracefully idles (but resumes promptly) when all threads are blocked on I/O
- Constrain CPU core usage
  - Argobots allows fine grained control of which threads and taskets execute on which cores (execution streams)

EVALUATION:

WHAT KIND OF PERFORMANCE CAN WE ACHIEVE UNDER THESE CONSTRAINTS?

PERFORMANCE MEASUREMENT

- Prototype API model is similar to libpmemobj, but tailored for remote access:
  - Explicit reads and writes rather than load/store access
  - Allow concurrent access to objects from many clients simultaneously
  - No transaction grouping
- Benchmarks:
  - Create new plugin for IOR (HPC concurrent I/O benchmark) to read and write to distributed objects rather than files
  - Create custom microbenchmark for access latency
    - Measure latency of a sequential set of write or read operations
    - Does not persist on write

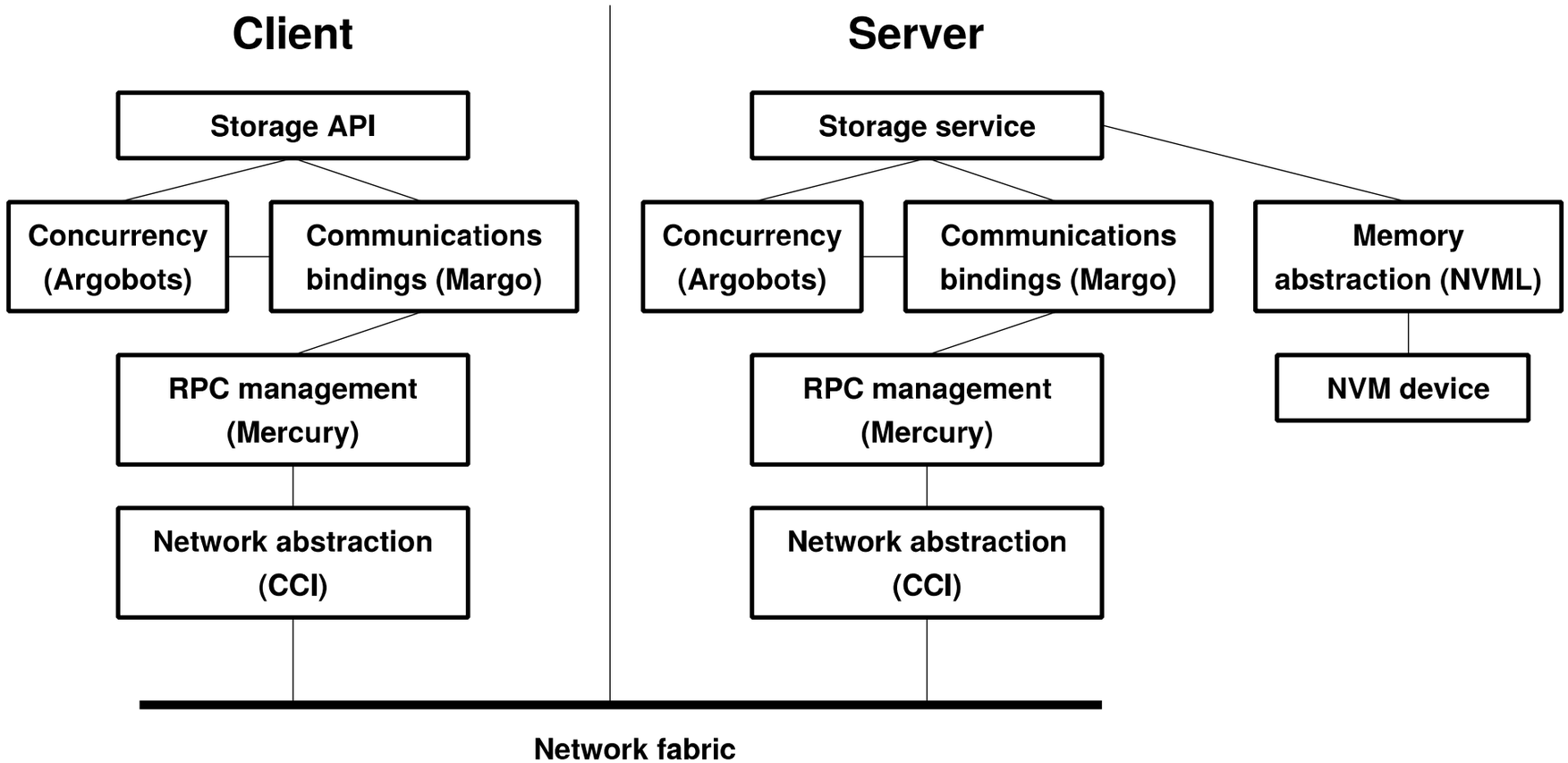
THE TESTBED

Cooley cluster at ALCF

- 126 Linux nodes, each with 12 Haswell CPU cores
- FDR InfiniBand fabric with multistage switch
- 384 GiB RAM per node (used to emulate NVM in these experiments)

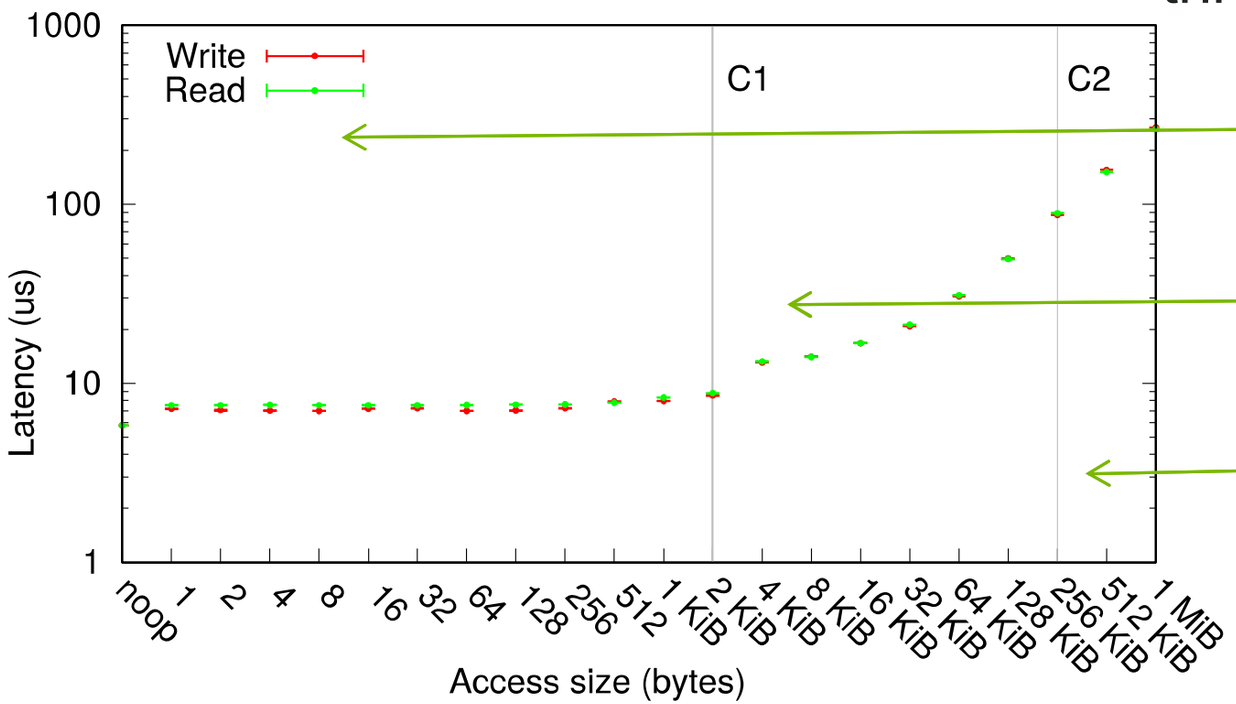
PUTTING THE COMPONENTS TOGETHER

A prototype object storage service architecture



ACCESS LATENCY

How much latency do those software layers add?



- Components are “thin”: no context switches or extra memory copies
- Recall that there is no busy polling
- Each access is at least one network round trip, one libpmem access, and one new thread

Protocol modes:

- Eager mode, data is packed into RPC msg
- Data is copied to/from pre-registered RDMA buffers
- RDMA “in place” by registering memory on demand

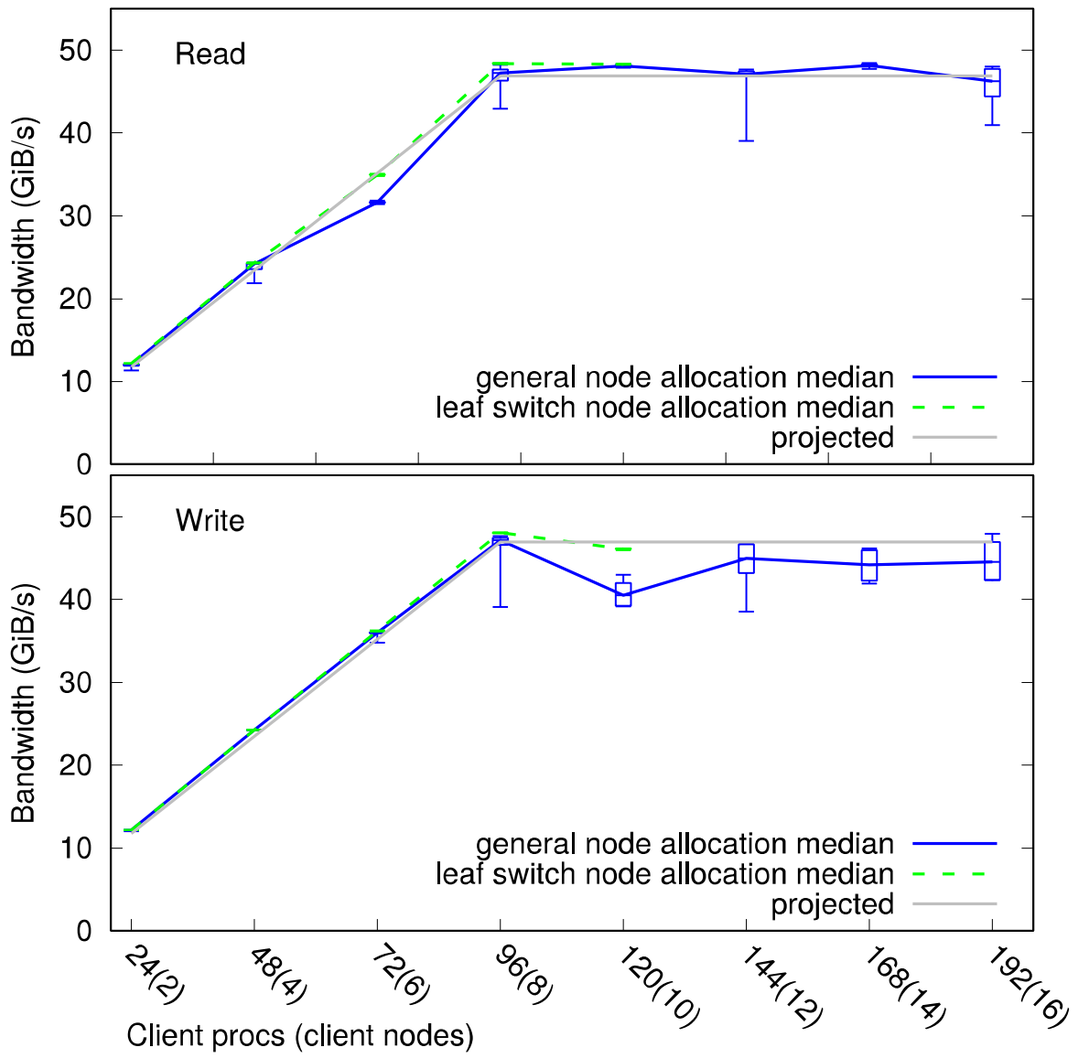
Crossover points would be different depending on transport

ACCESS LATENCY

Observations and questions

- Single digit microsecond access latencies, but could it be tuned further?
  - Consider adaptive polling
  - Optimize memory allocation
- What about the long tail?
  - Previous slide shows confidence interval for 10,000 samples at each point, and the intervals are quite narrow
  - But outliers are present: one of the noop samples was > 70 microseconds
- The cost of memory copy vs. registration is a key factor in choosing protocol crossover points

AGGREGATE BANDWIDTH



- 8 server nodes, with one service daemon per node
- 2 to 16 client nodes, with 12 application processes per node (one per core)
- Grey line is projected maximum
- Blue line is normal, random allocation
  - Whiskers (min and max) have significant variance
- Green line is special allocation with all nodes on one leaf switch
  - Whiskers (min and max) have very little variance

AGGREGATE BANDWIDTH

Observations and questions

- InfiniBand switches are not true crossbars, no matter how much us computer scientists would like them to be
  - Consider switch routing and congestion-avoidance algorithms?
  - Would better internal service instrumentation help?
  - Are we going to observe similar phenomena on other networks?
- The service can saturate bandwidth aggregate bandwidth relatively easily
- No drop-off up to 192 application processes accessing 8 daemons
  - What if we scale higher?
- What about a real scientific application?

CONCLUSIONS AND FUTURE WORK

WHAT'S NEXT?

- Initial results are encouraging
- Continue tuning and developing best practices
  - Investigate performance outliers
  - Isolate latency costs
- What can be done to further evaluate concept?
  - Real applications
  - Larger scale
  - More architectures
  - Real NVM hardware
  - Stage in and out to long term storage
- Update microservice model to allow remote or local access under consistent API
- Continue to leverage new techniques from the community

AVAILABLE CODE

Key components are already open-sourced under permissive licenses

- CCI: https://github.com/CCI/cci
  - Network abstraction
- Mercury: https://mercury-hpc.github.io
  - RPC framework
- Argobots: https://github.com/pmodels/argobots
  - User-level threading
- Margo: https://xgitlab.cels.anl.gov/sds/margo
  - Mercury/Argobots bindings
- Abt-snoozer: https://xgitlab.cels.anl.gov/sds/abt-snoozer/
  - Argobots I/O-aware scheduler

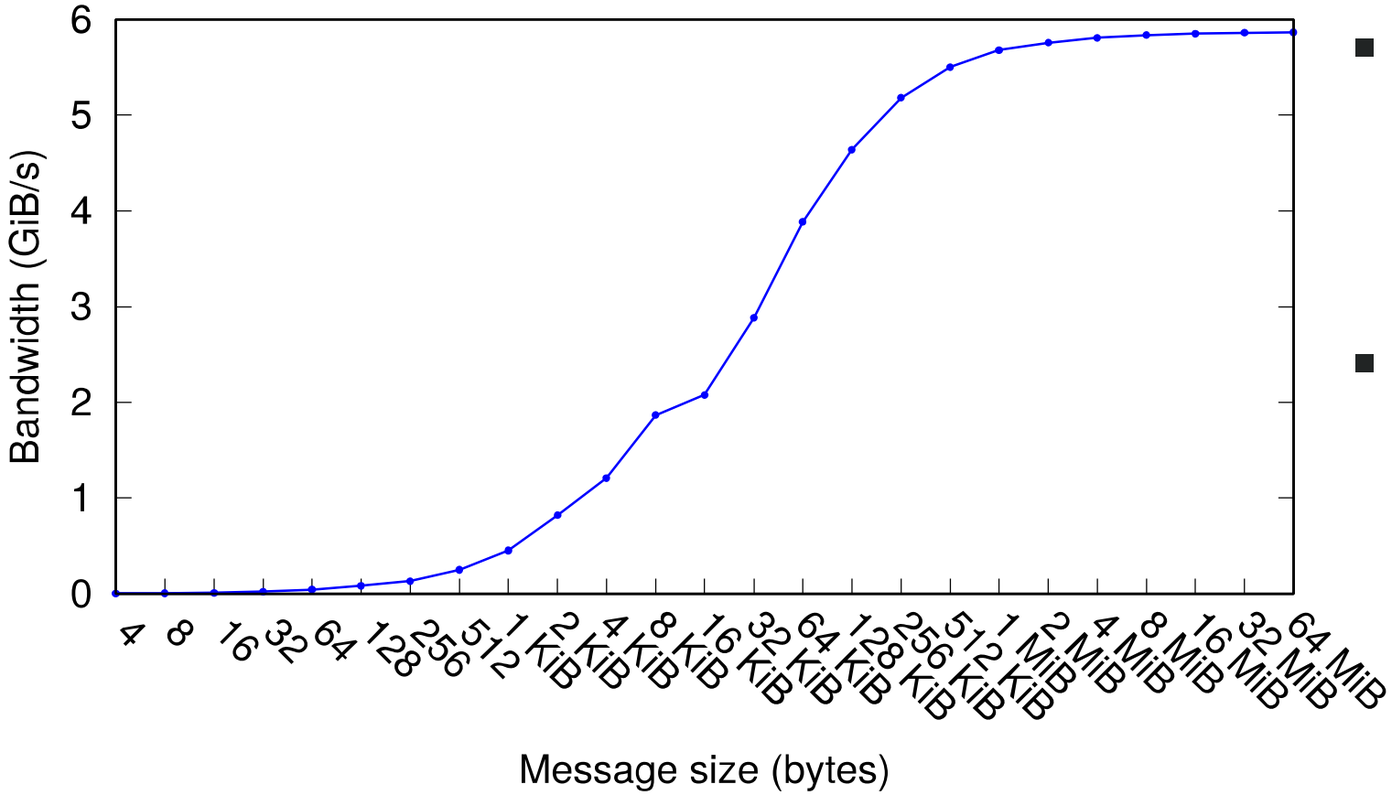
THANK YOU!

THIS WORK WAS SUPPORTED BY THE U.S. DEPARTMENT OF ENERGY, OFFICE OF SCIENCE, ADVANCED SCIENTIFIC COMPUTING RESEARCH, UNDER CONTRACT DE-AC02-06CH11357.

THIS RESEARCH USED RESOURCES OF THE ARGONNE LEADERSHIP COMPUTING FACILITY, WHICH IS A DOE OFFICE OF SCIENCE USER FACILITY SUPPORTED UNDER CONTRACT DE-AC02-06CH11357.

BASELINE PERFORMANCE

How did we come up with projected bandwidth and access latency values?



- We used *mpptest* benchmark atop the *MVAPICH* MPI implementation to gather baseline numbers
- 5.9 GiB/s max bandwidth and 2.6 microsecond min round-trip latency with two-sided asynchronous messaging