The HDF Group

ILLINOIS INSTITUTE OF TECHNOLOGY

March 26th 2021
Webinar

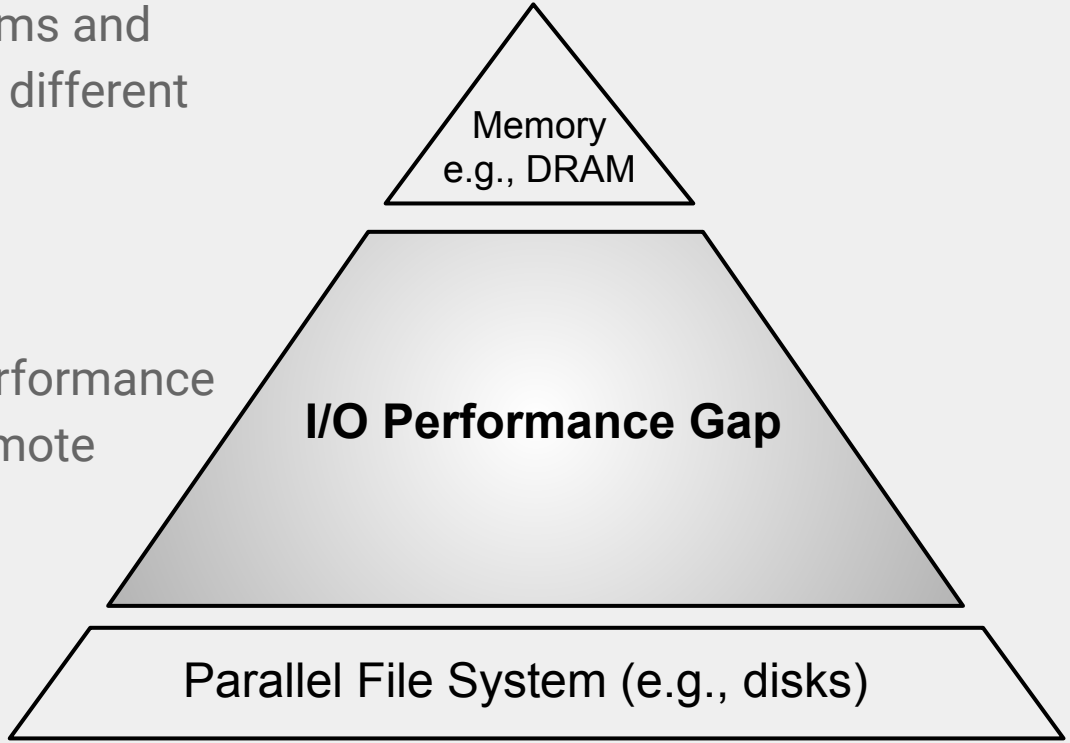
Agenda

- Hermes Project Overview
- Hermes in Detail
- Hermes Related Work
- Data Placement in Distributed Hierarchical Environments
- Hermes Container Library
- How can I use Hermes?
- Demo
- How can I get involved?

Project Overview

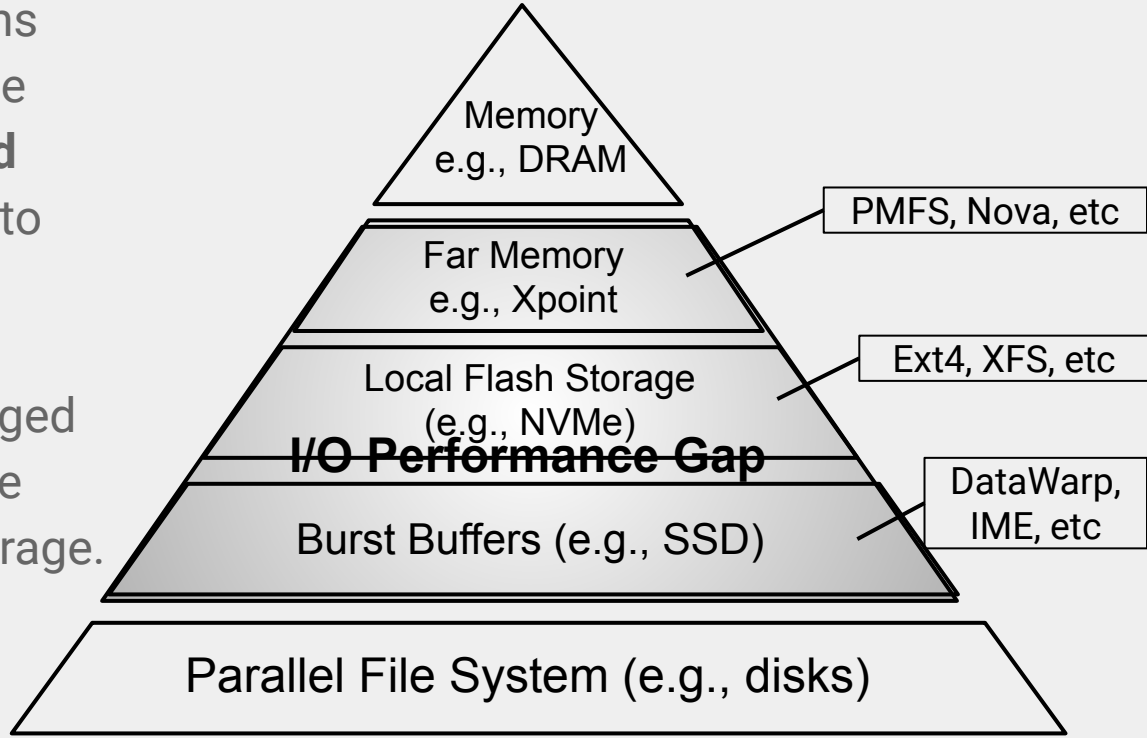
I/O Performance Gap

- Traditionally memory systems and storage demonstrate wildly different performance.
  - Access latency
  - Bandwidth
  - Data representation
- Applications experience performance degradation due to slow remote access to storage.



Hardware Answer

- Modern storage system designs include multiple tiers of storage organized in a **deep distributed storage hierarchy**. The goal is to mask the I/O gap.
- Each system is independently designed, deployed, and managed making very difficult to reap the benefits of the hierarchical storage.



Ideally, the presence of multiple tiers of storage should be **transparent** to applications without having to sacrifice **I/O performance**.

Hermes Project

The team

Collaborative project

funded by NSF

ILLINOIS INSTITUTE OF TECHNOLOGY

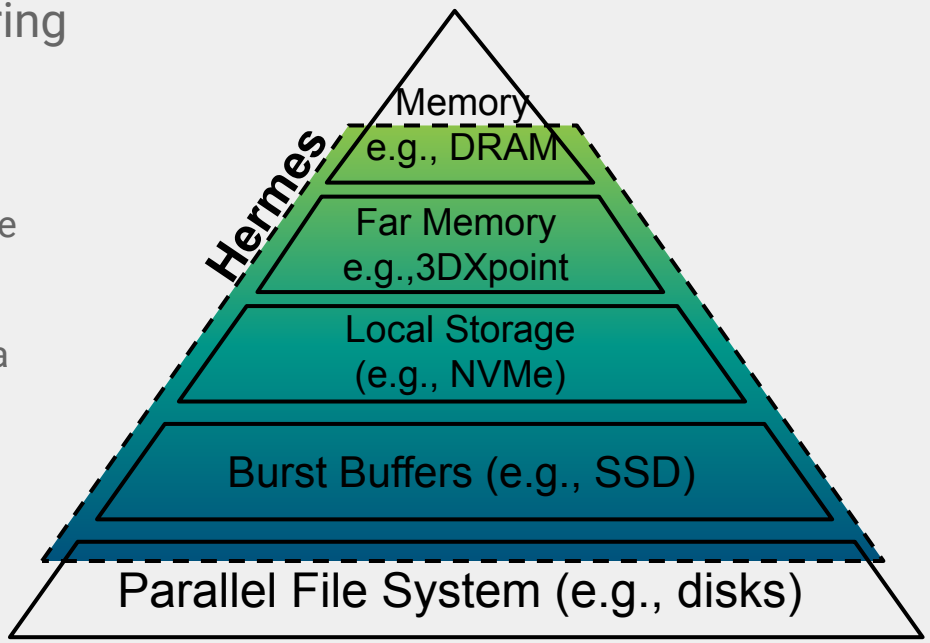
The HDF Group

ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

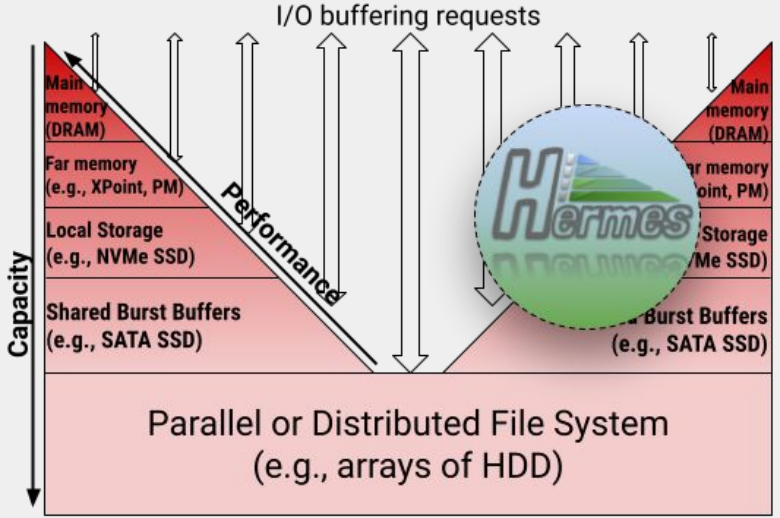
Hermes Overview

- A new, multi-tiered, distributed buffering system that:
  - Enables, manages, and supervises I/O operations in the Deep Distributed Storage Hierarchy (DDSH).
  - Offers selective and dynamic layered data placement.
  - Is modular, extensible, and performance-oriented.
  - Supports a wide variety of applications (scientific, BigData, etc.,).



Objectives

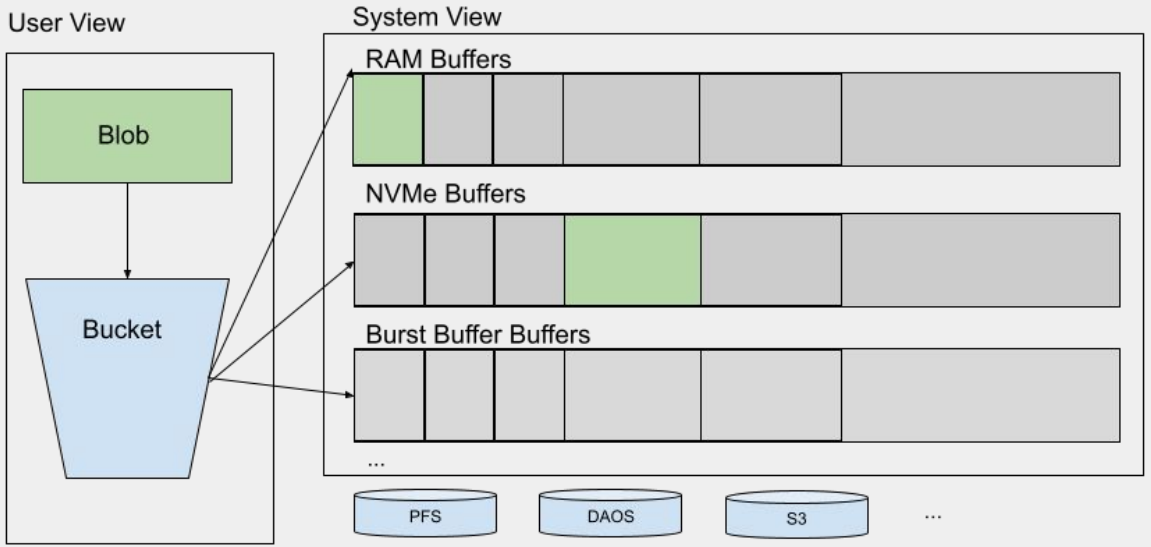
- DDSH requires:
  - a scalable, reliable, and high-performance software to efficiently and transparently manage data movement.
  - new data placement algorithms, memory and metadata management, and an efficient communication fabric.
- Hermes strives for:
  - being application- and system-aware,
  - maximizing productivity and path-to-science,
  - increasing resource utilization,
  - abstracting data movement,
  - maximizing performance, and
  - supporting a wide range of applications



Hermes in Detail

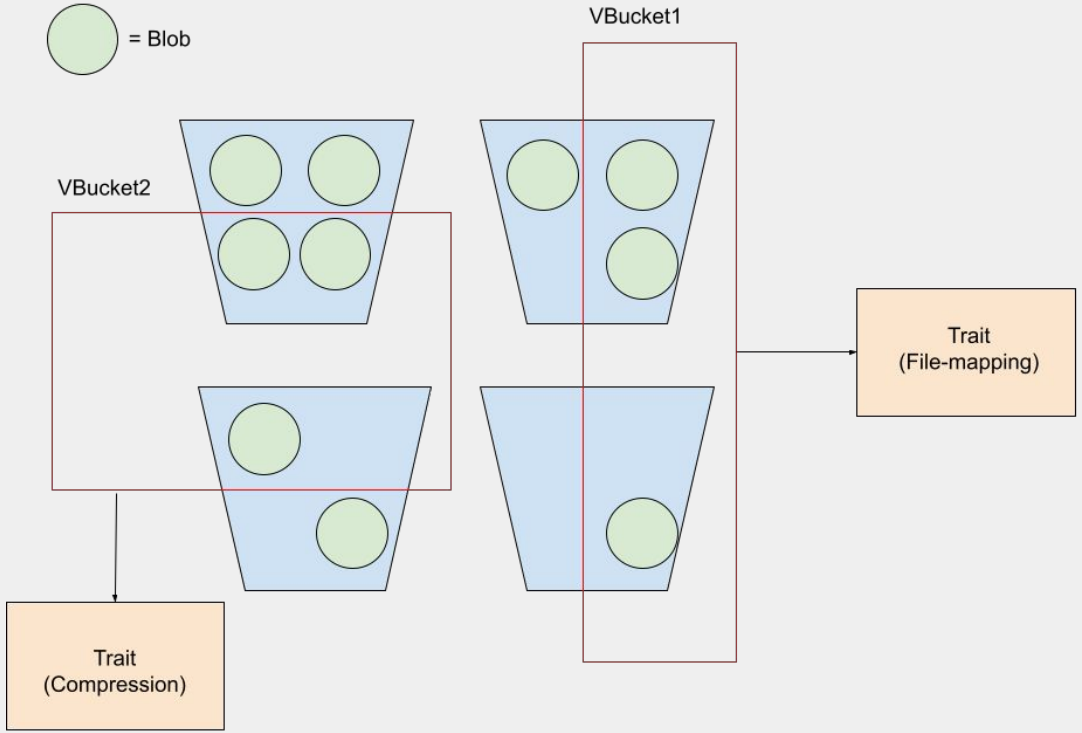
Abstractions

- **Buffer** - Hermes-aware, fixed-size storage, backed by preallocated blocks.
- **Blob** - A finite sequence of uninterpreted bytes. User data. Gets stored in Buffers.
- **Bucket** - A collection of named Blobs.



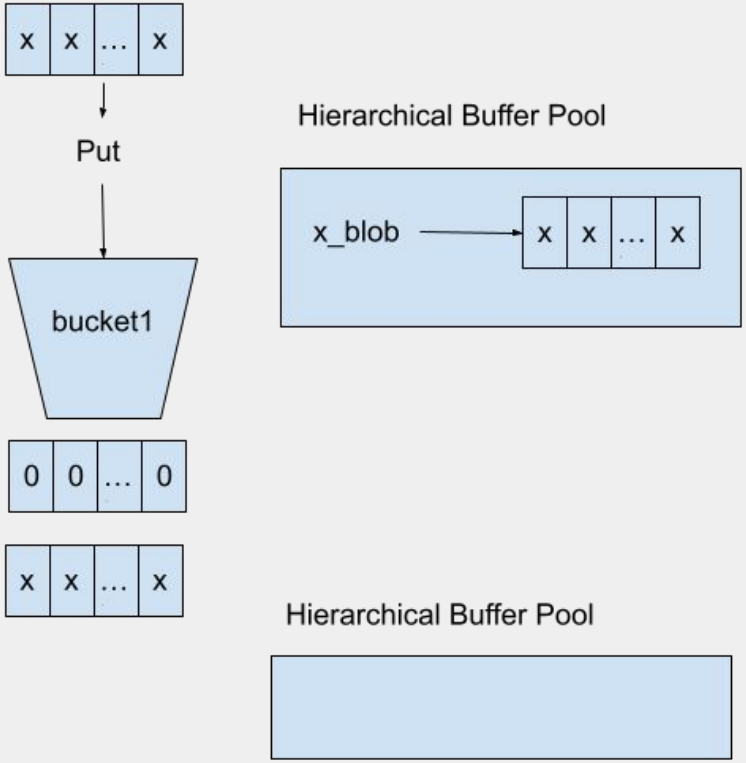
Abstractions

- **VBucket** - A collection of links to Blobs. Can be decorated with Traits.
- **Trait** - Event handlers, called when Blobs are linked (or unlinked) to VBuckets.
  - Hierarchy placement
  - File-mapping
  - Filtering
  - Compression
  - Encryption



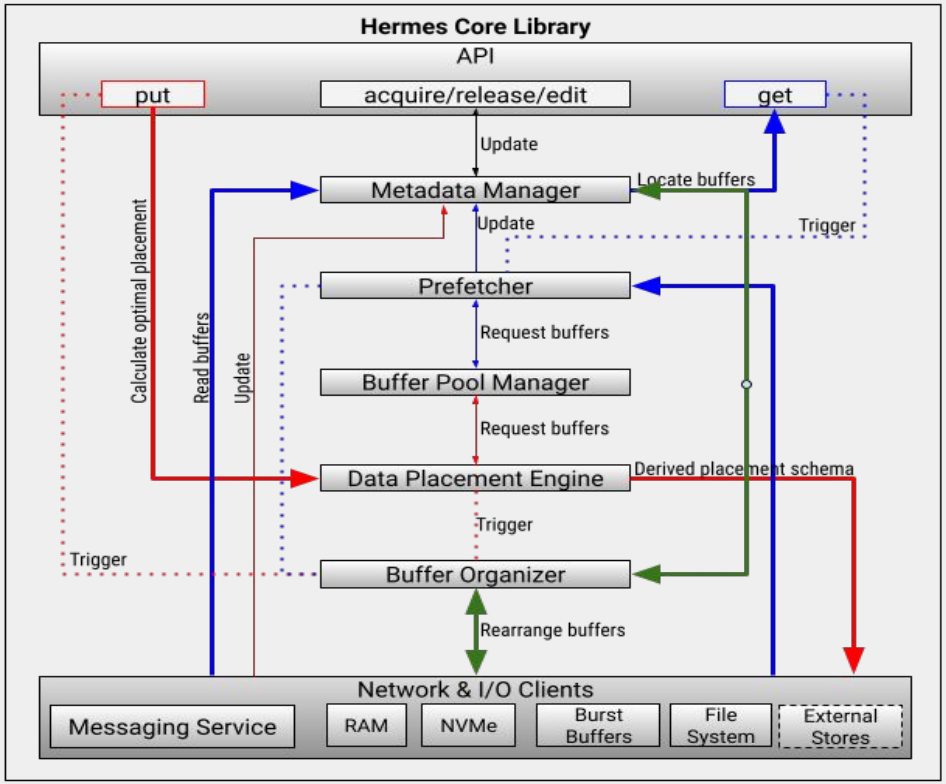
API Example


```
blob(4096, 'x')

blob_name = "x_blob"

bucket("bucket1")

bucket.Put(blob_name, blob)

…

retrieved_blob(4096)

bucket.Get(retrieved_blob)

bucket.Destroy()
```



High-Level Architecture

- API
- Metadata Manager
- Prefetcher
- Buffer Pool Manager
- Data Placement Engine
- Buffer Organizer
- I/O Clients



Related Work

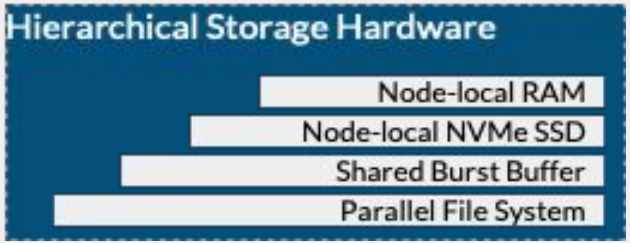
Related work

- LBNL
  - Data Elevator 2016 (https://github.com/hpc-io/vol-dataelevator)
    - Data elevator: Low-contention data movement in hierarchical storage system, 2016 IEEE 23rd HiPC
  - Unified View of Storage (UniviStor) 2018
    - UniviStor: Integrated hierarchical and distributed storage for HPC, 2018 CLUSTER
  - Proactive Data Containers (PDC) 2018 (https://github.com/hpc-io/pdc)
    - Proactive Data Containers (PDC): An Object-centric Data Store for Large-scale Computing Systems. In AGU Fall Meeting Abstracts, vol. 2018, pp. IN34B-09. 2018.
  - HDF5 Cache VOL 2020 (https://github.com/hpc-io/vol-cache)
    - HDF5 Cache VOL: Efficient Parallel I/O through Caching Data on Node-local Storage, 2020 IPDPS
- LLNL
  - UnifyFS (https://github.com/LLNL/UnifyFS)
    - Documentation https://unifyfs.readthedocs.io/en/latest/

Data Placement in Distributed Hierarchical Environments

Memory Hierarchy Targets

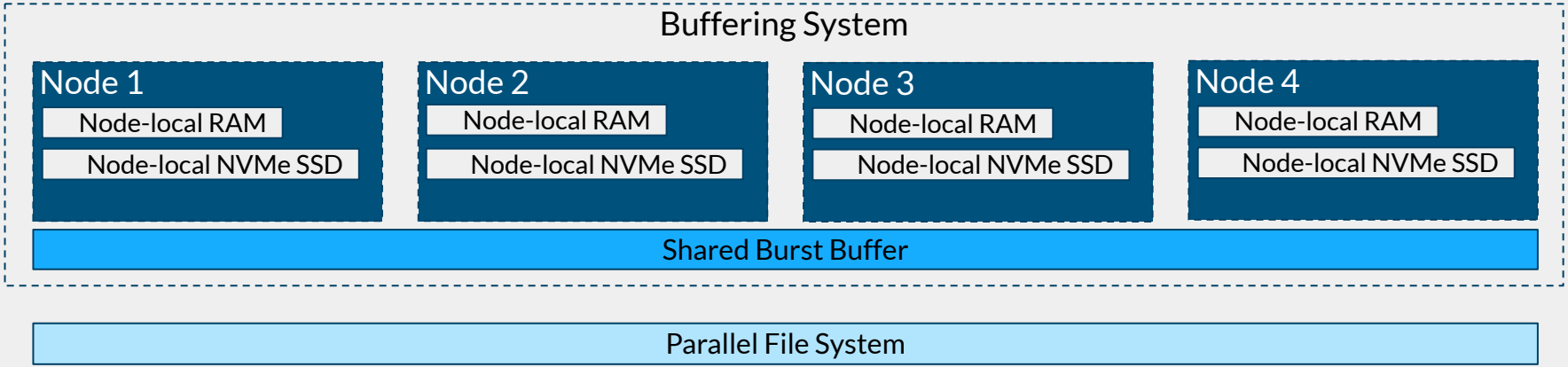
- Every Hermes system instance includes one or more Hermes *nodes*.
- A target is a buffering resource that is identified by a pair of node + target “coordinates”.
- Each target t_k has characteristics such as the following:
  - A capacity Cap[t_k]
  - A remaining capacity Rem[t_k]
  - A speed (or throughput) Speed[t_k]



The Data Placement Problem

- Given *N* targets, a data placement policy *P*, an objective function *F*, placement constraints *C*, and a set of Blobs.
- Problem: Place Blobs such that the *P* is followed, *F* is minimized/maximized, and *C* is enforced.
- Tradeoff
  - Time to solution
  - Cost to move data
  - Load balance

How to Place Blob(s) on Targets

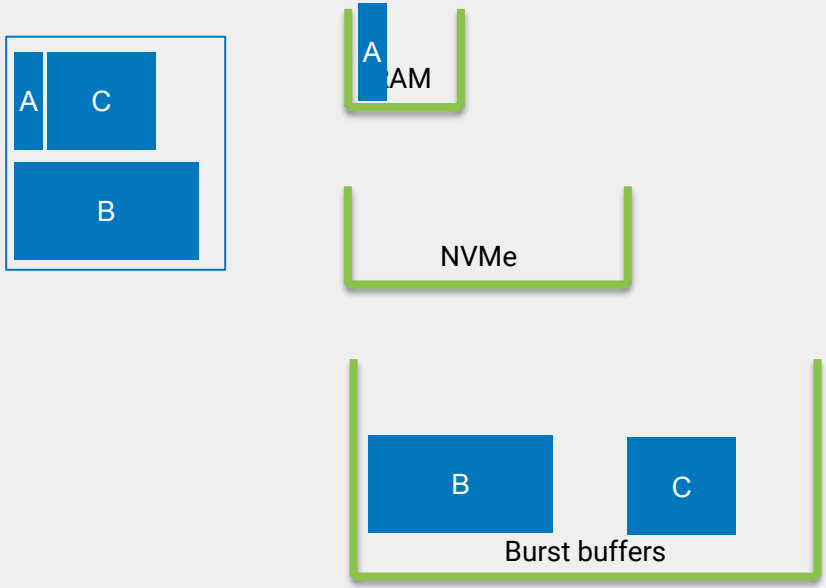


- It is not practical to consider all available *N* targets in each data placement.
- By default placement starts with node-local targets, then with neighborhood targets, then with global targets (three topologies of targets).

Data Placement Solvers

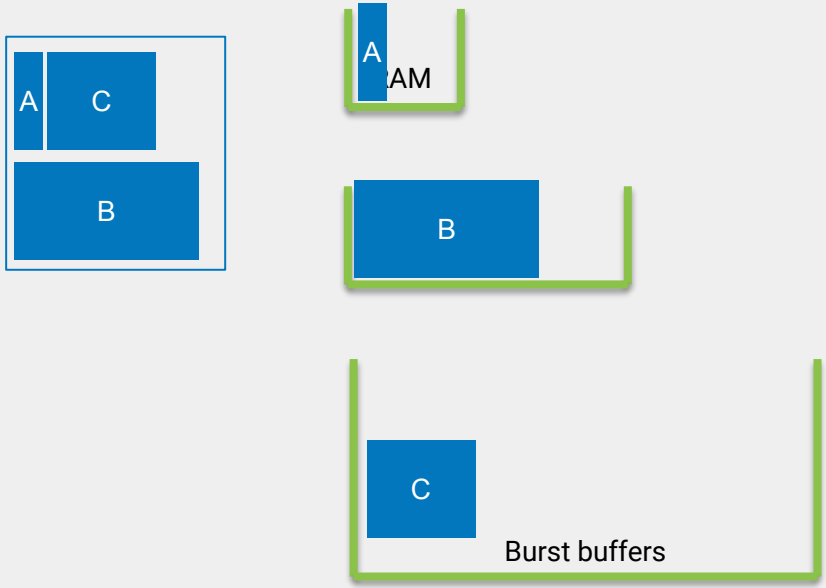
- Random
- Round-Robin
- Linear Programming

The Random Solver



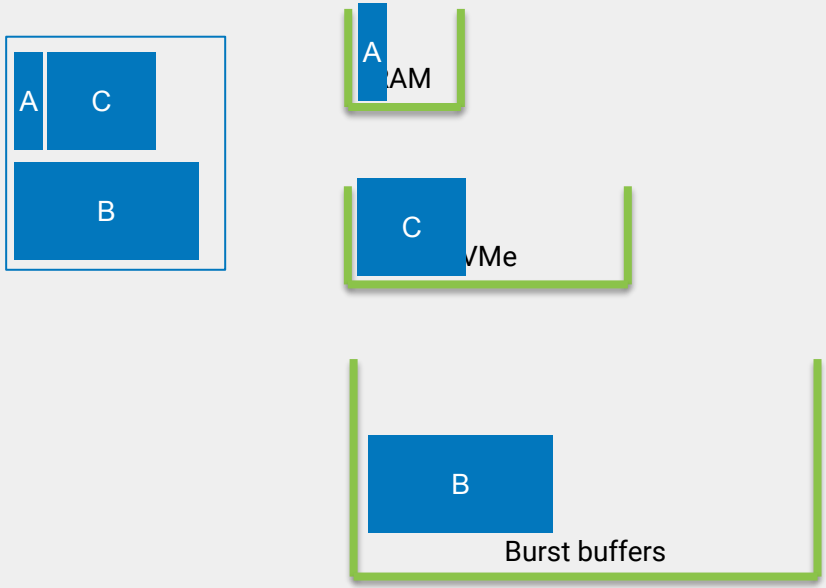
- Randomly pick a target from all targets, which has the capacity greater or equal to the BLOB size.

The Round-Robin Solver



- Pick the next target if the remaining capacity is greater or equal to the BLOB size, otherwise check the one after the next target until a target with enough capacity is found.

The Linear Programming Solver



- Minimize client I/O time.
- We use Google OR-Tools for linear optimization.

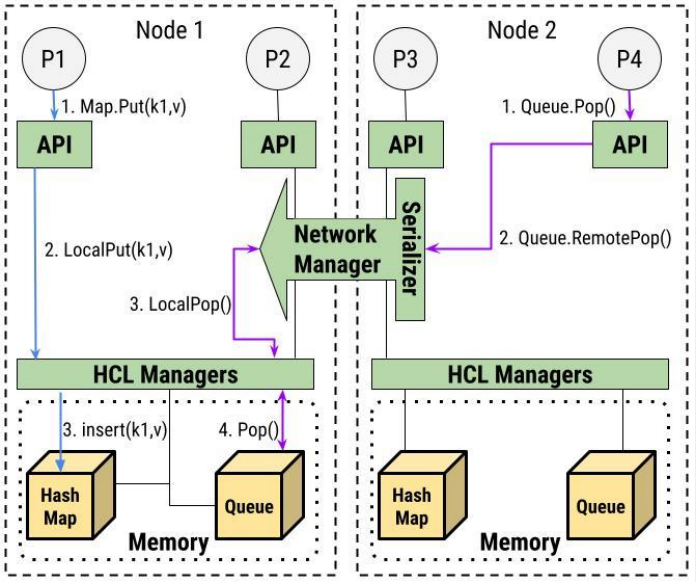
https://github.com/HDFGroup/hcl

Hermes Container Library

H. Devarajan, A. Kougkas, K. Bateman, and X. Sun. "***HCL: Distributing Parallel Data Structures in Extreme Scales***."
In 2020 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2020.

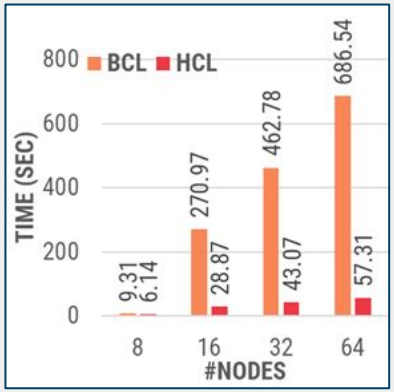
HCL Architecture

- Uses a hybrid data model data structure access
  - Shared memory for intra-node
  - RPC for inter-node
- Uses STL data structures within shared memory
- Uses three types of transports
  - RPC Lib: a C++ wrapper over rpc protocol
  - Thallium: a C++ wrapper over mercury-RPC protocol
    - TCP
    - ROCE

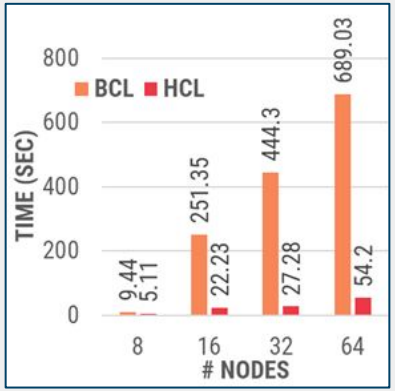


Evaluation Results

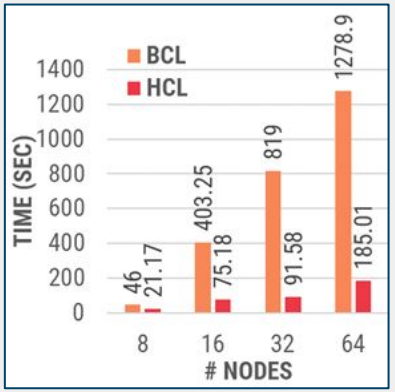
ISx Benchmark



Genome Contig Gen



Genome k-mer counting



Key Observations

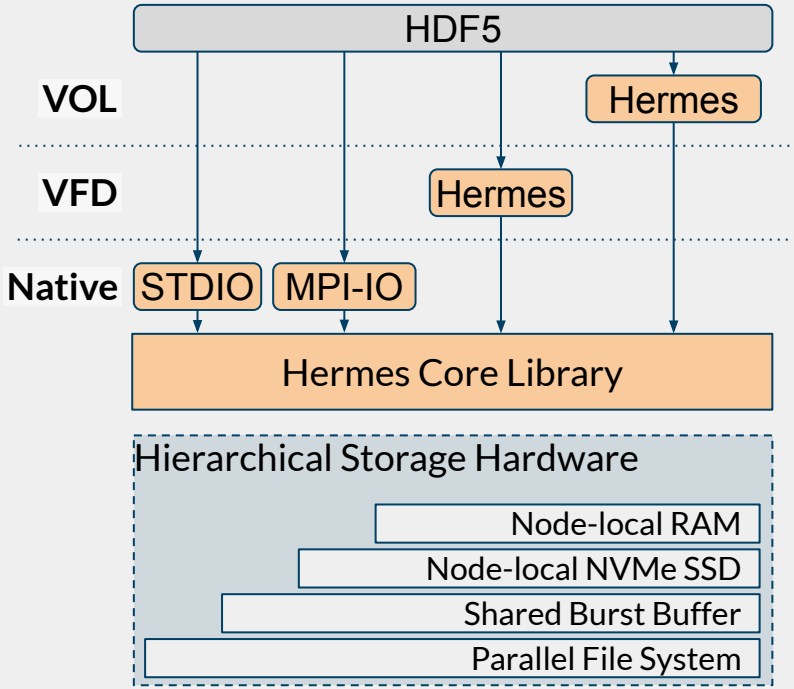
- Enables higher performance through hybrid data access model.
- Over 10x performance improvement.

How can I use Hermes?

Hermes Adapter Layer

Applications can natively interact with Hermes using existing I/O Interfaces

- Standard Interceptors
  - STDIO
  - POSIX
  - MPI-IO
- HDF5 Level
  - Hermes VFD
  - Hermes VOL



Deployment Models

Collocated

- Hermes Core is part of the application.
- Synchronization is managed internally by hermes lib.
- Isolates buffering data across applications.

```
mpirun -n 1280 -f app_hf ./application
```

Decoupled

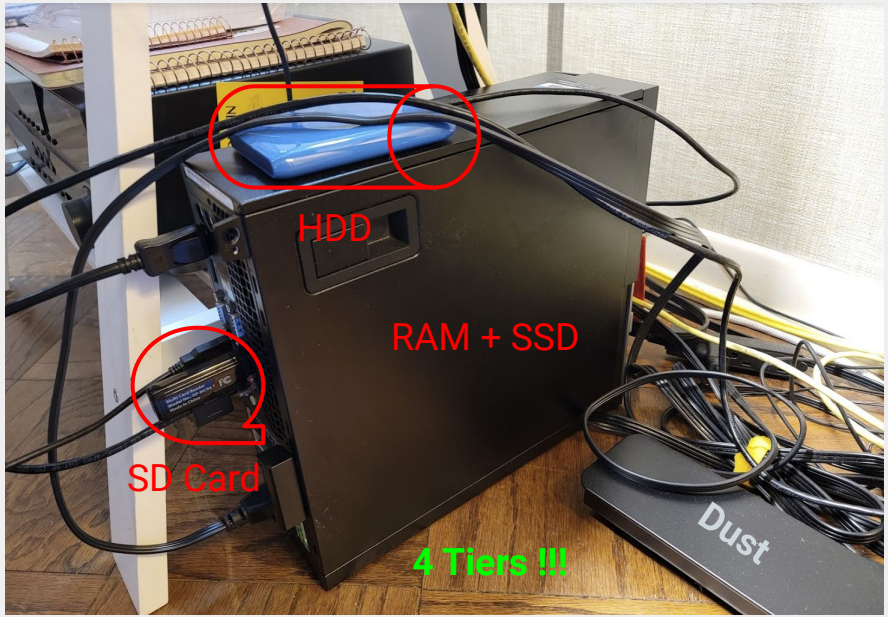
- Hermes Core is separate from the application.
- The Hermes core needs to be running before the application.
  - Manually, or
  - as a service
- Can share buffering data across applications.

```
mpirun -n 32 -f hermes_core_hf ./hermes_core
mpirun -n 1248 -f app_hf ./application
```

Demo

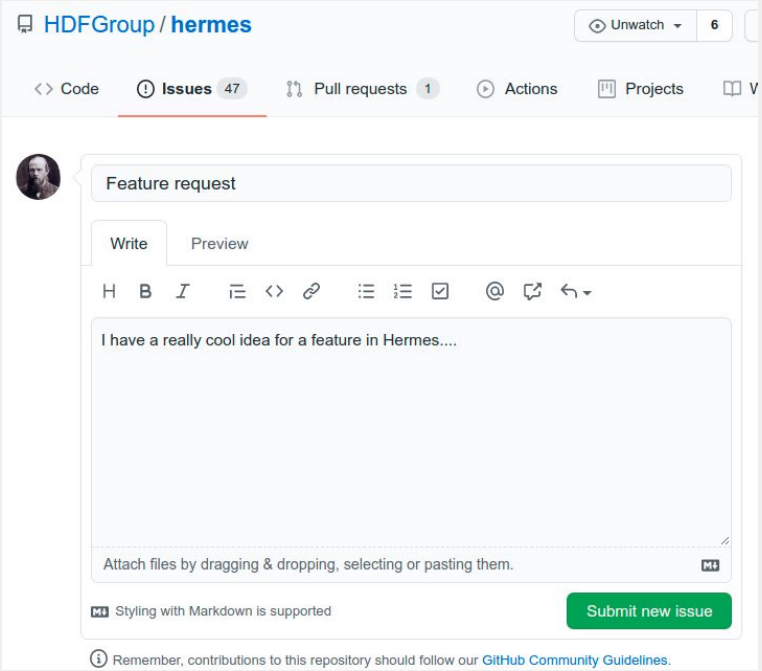
Hermes @Home Setup

- My Dell OptiPlex 7020 w/ [enhanced storage hierarchy] ™
- Running GNU Guix
- You can build Hermes from source
  - Spack
  - CMake
- I prefer the Docker image
- Two demos:
  - Hermes basics
  - UNIX STDIO adapter
- Demo source and details



How Can I Get Involved?

- Github repo: https://github.com/HDFGroup/hermes
- Create an issue to submit feedback, use cases, or feature requests.
- Note: Hermes is still under active development.



Multi-Tiered
Distributed I/O
Buffering System

Thank you.

Learn more

http://www.cs.iit.edu/~scs/assets/projects/Hermes/Hermes.html

https://github.com/HDFGroup/hermes

https://github.com/HDFGroup/hcl

Contact us

akougkas@iit.edu

gheber@hdfgroup.org

The HDF Group

ILLINOIS INSTITUTE OF TECHNOLOGY