ExCaliwork: Optimizing Data Movement with Active Storage

Konstantinos Chasapis, kchasapis@ddncom
Jean-Thomas Acquaviva, jtacquaviva@ddn.Com

ddn

Excaliwork - Excalibur subproject: Data Reduction

Two techniques to reduce data movement: (1) Active (Computational) Storage and (2)Ensemble Data Analysis

Bryan Lawrence[1,2], Grenville Lister[1], Jeff Cole[1], David Hassell[1], Valeriu Predoi[1], Stig Telfer[3], Matt Pryor[3], Konstantinos Chasapis[4], Jean-Thomas Acquaviva[4] , Scott Davidson[3], Mark Goddard[3]

[1]National Centre for Atmospheric Science, [2]Department of Meteorology and Department of Computer Science, [1]University of Reading, [3]StackHPC, [4]DDN

Big Picture

AIM: Deliver a completely new paradigm for where (some) computations are performed **by reducing the amount of data movement** needed – and in the context of simulations, particularly for **ensemble outputs**.

CONTEXT: Excalibur Cross-Cutting Project, **ExcaliWork,** aiming to address a range of use-cases.

COMPONENTS: Active-Storage (Python/S3/POSIX), In-Flight Ensemble Analysis

PI: Pr. Bryan Lawrence from University of Reading / NCAS

Data Movement and Performance Optimization

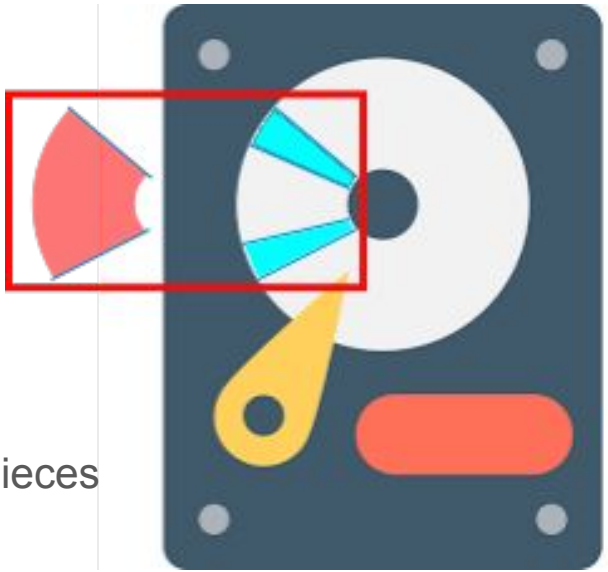
IO community has a long tradition of doing ‘tricks’ on the behalf of end-users for performance purpose

e.g. Caching, Look-ahead

Data sieving:

Hard Drive IOPS are expensive

- Limit the number of head movements on HDD
- Bring a large chunk of data across the fabrics
- Throw away most of the chunk and keep only interesting pieces



Trading data movement for IOPS

… less IOPS better performance

… more data movement more **power**

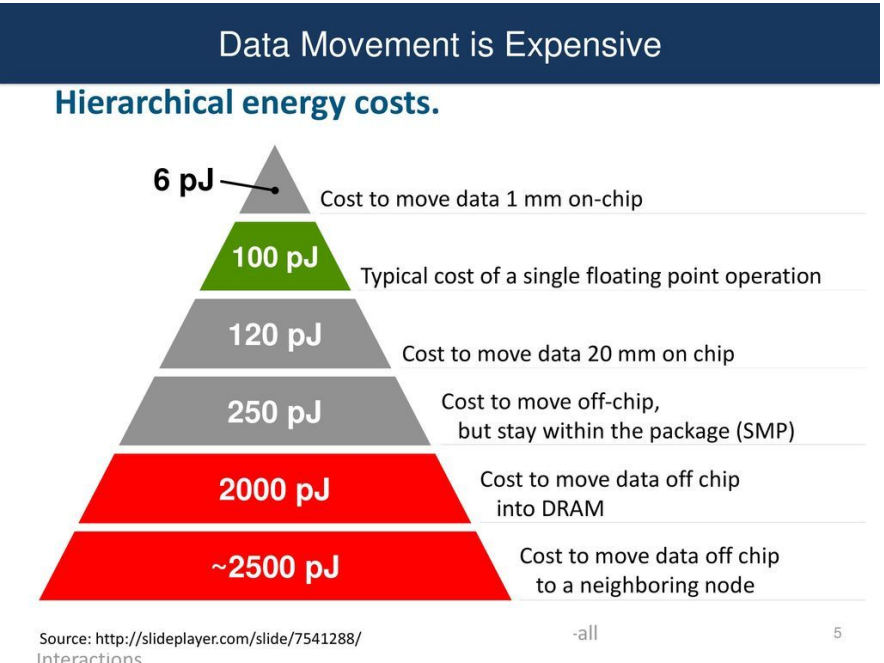
Flash Technology Challenges Old Optimizations

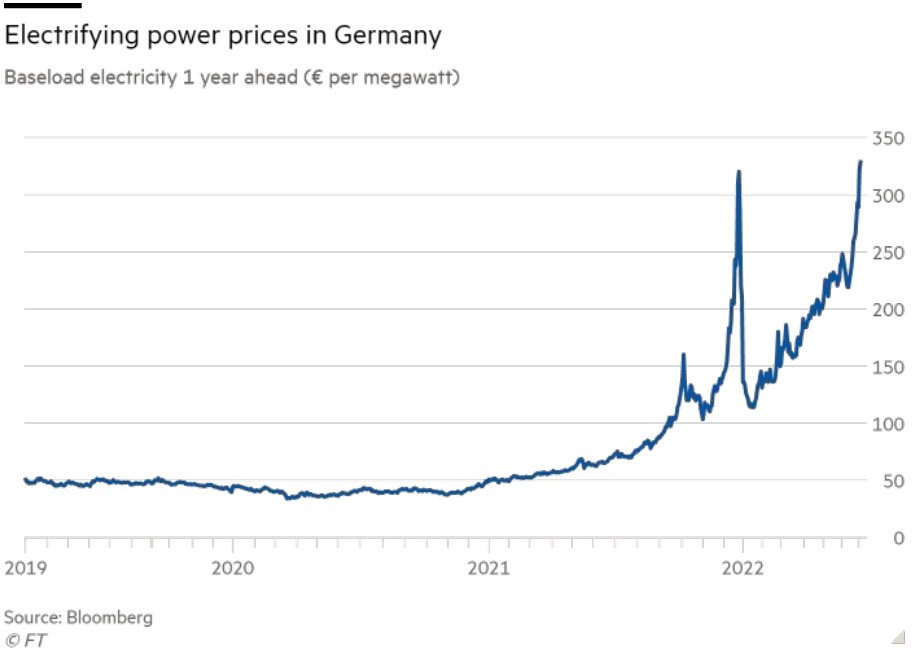
- Flash IOPS budget >> HDD by 3 orders of magnitude
  - To 10 µsec from 10 msec
- IOPS no longer such an important topic
- Flash Bandwidth >> HDD by 2 orders of magnitudes
  - To 10 GB/s from 0.2 GB/s
- Bandwidth no longer such an important topic

Where should we put the optimization effort?

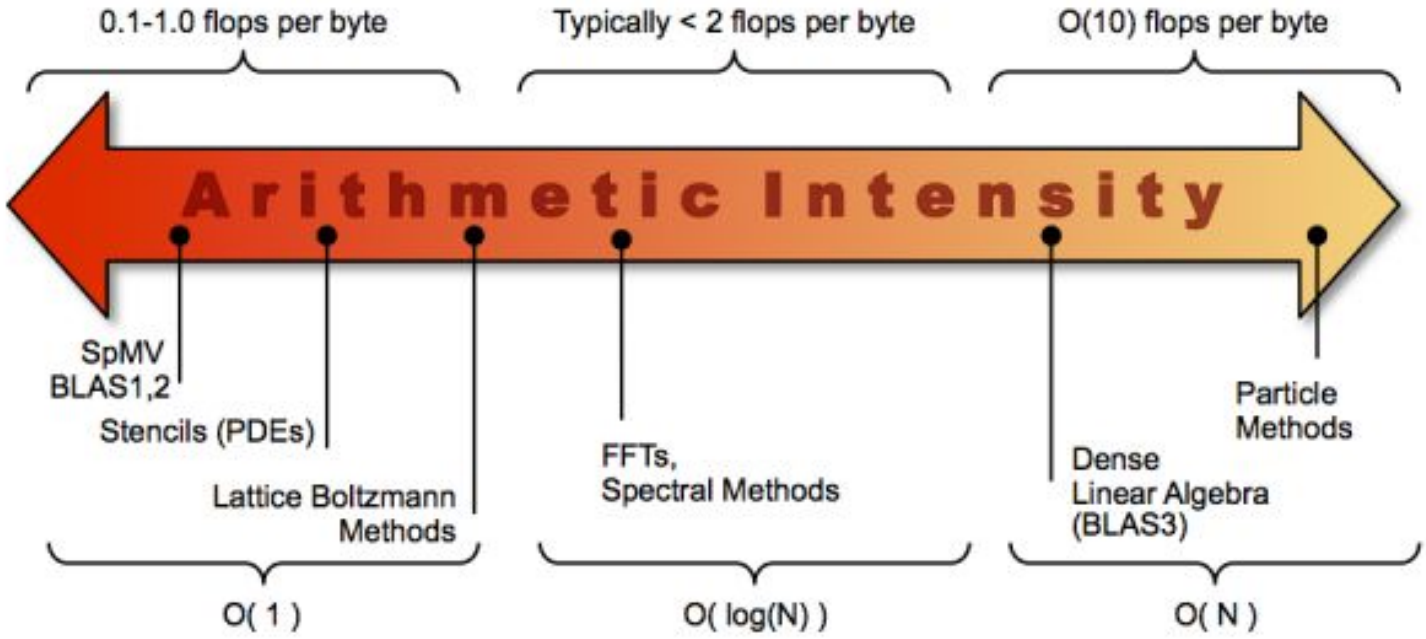
New Technology Challenges Old Optimizations

- Infrastructures size is caped by Power
  - Power Requirement and Power Cost
- **Power** very much an important topic





Arithmetic Intensity: Data Requirement and FLOPS



What can we do here?

Offload to Accelerator

Offloading Kernels to Storage Servers

Comprehensive Offloading Options

- High Arithmetic intensity: Good Candidate for Accelerator offloading
- Low Arithmetic intensity: Good Candidate to Storage offloading
  - Minimize the volume of data transiting through the network
  - Increase performance, decrease power
- Credential promotion from end user to Root
- No loop -> guarantee of Termination
- Limited set of operations: MPI Reduction, Min, Max, Add,

Active Storage: Application Support API is Required

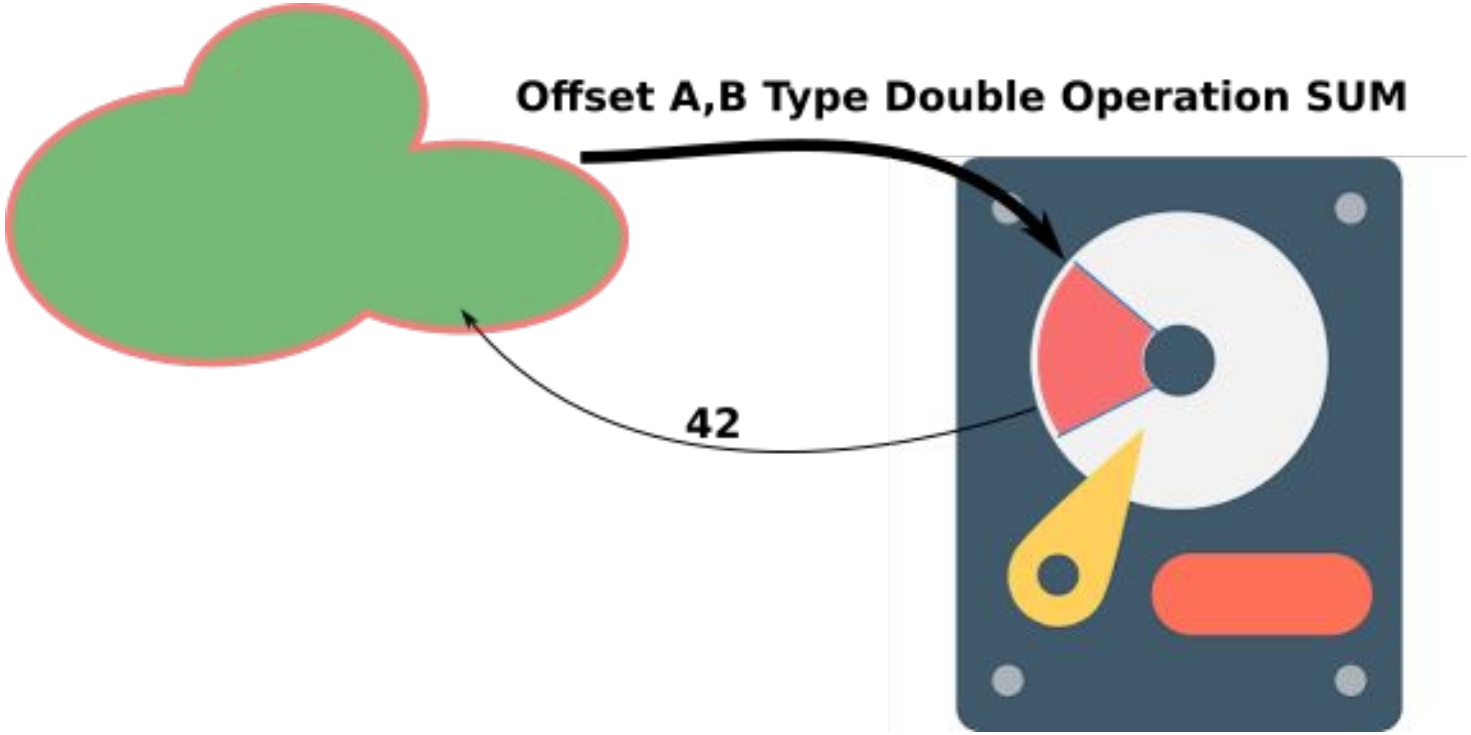
Support provided by University of Reading

- Instantiated as a Python library
- Bridging semantic gap:
  - Storage knows 0 and 1
  - Application knows data type
- Check if Active Storage is available
  - if Not: Read a data range, apply operation and compute result
  - if Yes: make a specific call the Storage Client
    - Data Range, Operands and Result Type, Operation to apply

Active Storage: Experimental Set-up

- Source tree of one of DDN's Parallel File System software
- Implement API support within the file system storage client
  - Running in User Space on the Compute Node
  - Intercept Application Calls
- Implement Storage Client to Storage Server RPC protocol
  - Leverage Existing RPC Calls
- Implement Active Storage function within Storage Server Code
- Demonstrate with U. Reading benchmarks

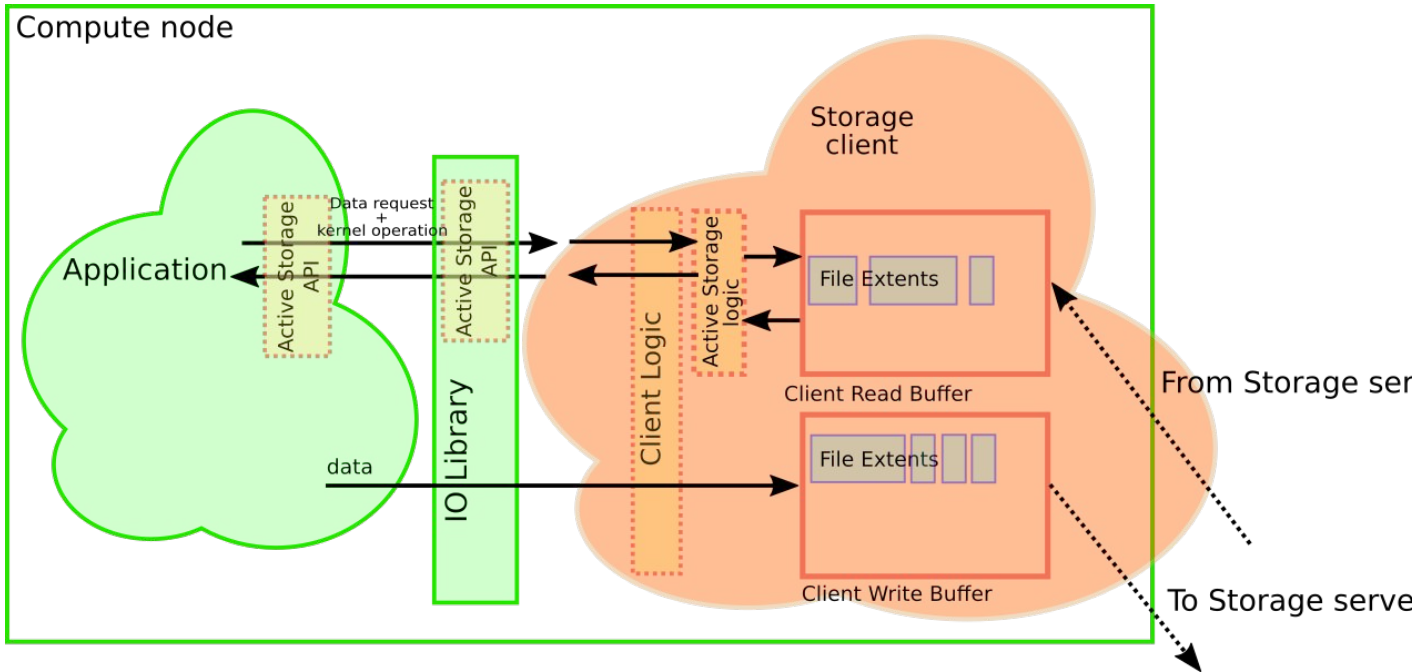
Active Storage: Concept



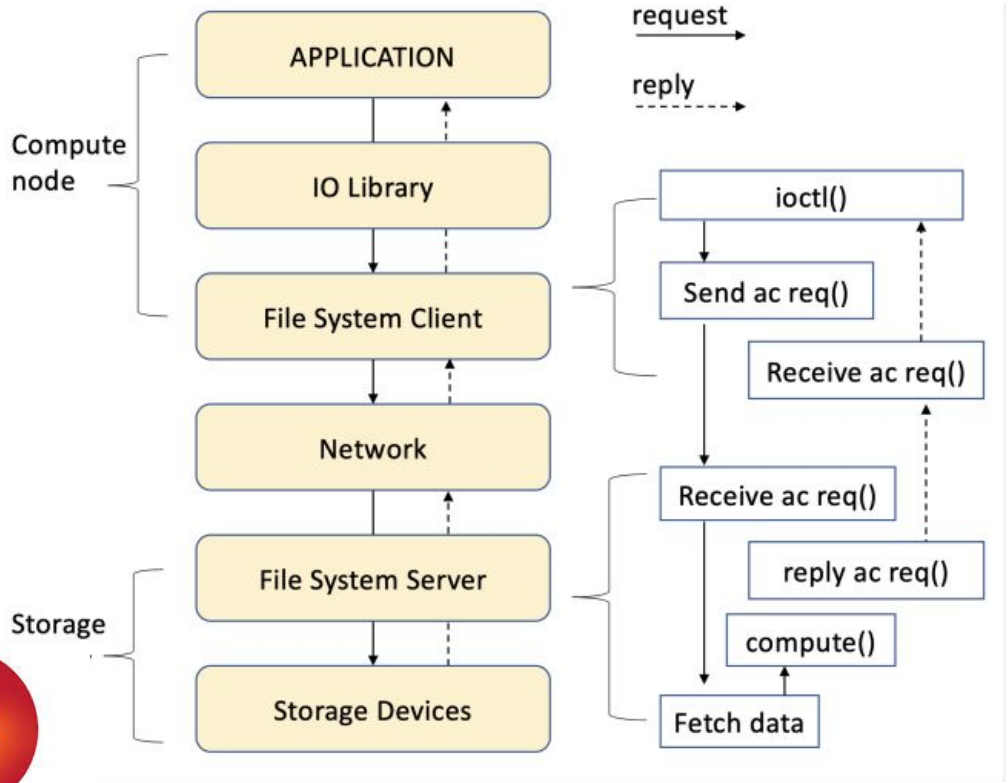
Active Storage: 2-Phases Implementation

Initial implementation through an Active Client

- FUSE prototype of the Client File System
- API with Application



Active Storage: Full-fledged Software Stack

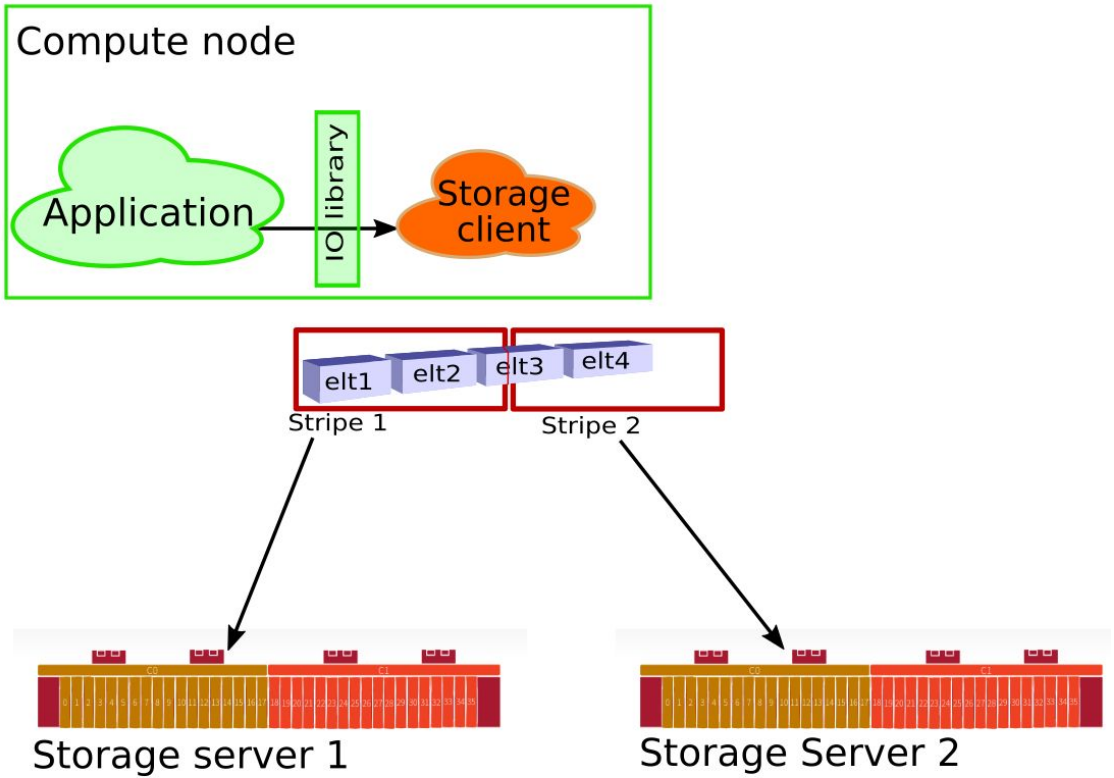


- Application issues an ioctl to the file system client
- File system client sends an active storage request to the primary storage server
- The storage server receives the active storage request
- Issues a fetch data call and loads data in memory
- Storage server applies the computation indicated by the active storage call
- Storage server replies to the file system client with the result of the compute function
- File system client receives the reply from the storage server and exposes the result of the compute function to the application

Functional Validation on Virtual Clusters

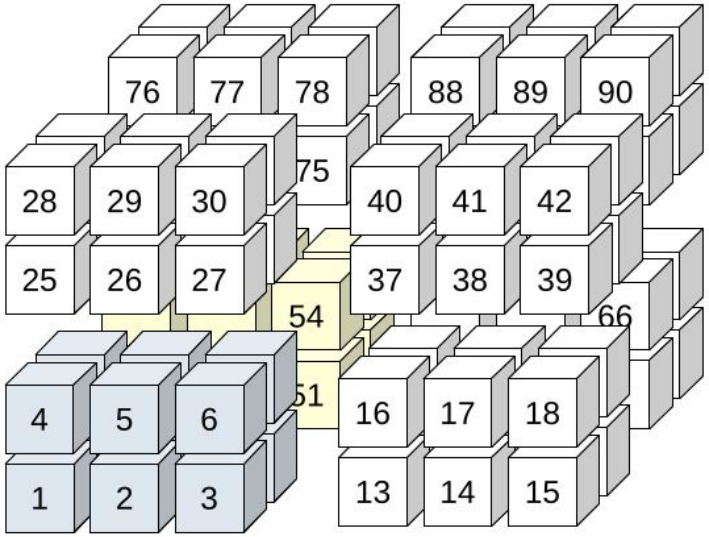
- Conducted Tests demonstrate feasibility
  - Fully implemented in a production grade parallel file system
- Fully operation on small benchmarks
- Demonstrate on Virtual Cluster
  - No meaningful Performance Measurement
  - Lack of HW testbed

Striping: the Art of Data Layout in Parallel File Systems



- Files can be distributed over multiple Storage Servers

Active Storage: Data Layout HDF5 & ZARR



HDF5

The HDF5 library allows the application to request alignment of all objects in a file over a particular size threshold, with the **`H5Pset_alignment`** API call. This allows aligning the chunks for chunked datasets to a favored block boundary for the file system.

ZARR

`class zarr.storage.DirectoryStore(path, normalize_keys=False, dimension_separator=None)[source]`

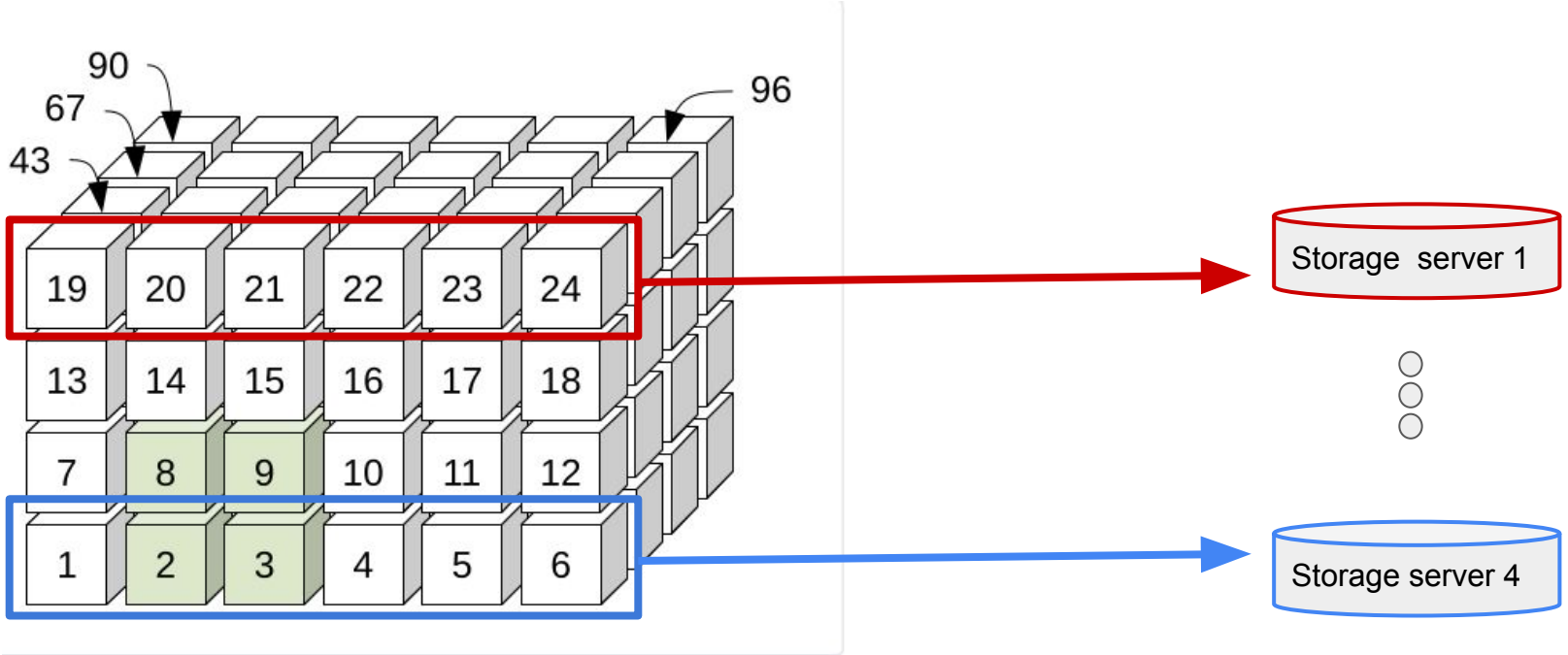
Storage class using directories and files on a standard file system.

- import zarr
- store = zarr.DirectoryStore('data/array.zarr')
- z = zarr.zeros((10, 10), chunks=(5, 5), store=store, overwrite=True)
- z[...] = 42

Each chunk of the array is stored as a separate file on the file system

https://zarr.readthedocs.io/en/stable/api/storage.html

Active Storage: data layout on distributed servers



Having each chunk handled as an individual object / files allows each server to process at the right granularity

Take-Away

Data Reduction is a major opportunity for performance/power optimization

We have demonstrated the feasibility of Active Storage in a Production Grade parallel File System.

Some future works can be foreseen

- Evaluation with larger application
- Testbed: Quantified the Power / network traffic relation
- Increase the scope of supported operations
- Lateral cooperation between storage servers (split chunks)


EXCALIBUR
10
Thanks!
Questions?
ddn

Active Storage: data layout HDF5 & ZARR

- File layout is manageable in Lustre (single stripe)
  - Check the status for RED
- Pursue RED client implementation
- Investigate ZARR file per chunk compression in Fuse

- Kang, D., Rübel, O., Byna, S., & Blanas, S. (2020, May). Predicting and comparing the performance of array management libraries. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 906-915). IEEE.
- Howison, M. (2010). Tuning hdf5 for lustre file systems.

Offload numerical kernels to limit data movement

Scientific workloads contain code segments with limited arithmetic intensity:

- Parse a large amount of data to apply a simple kernel
- Client will fetch data from servers and apply the computational kernel on the read data

Toward Server implementation

- Read pattern in RED is N:M
- no layout guarantee