Current and Future Developments

Parallel Storage Systems

2023-07-10

Prof. Dr. Michael Kuhn

`michael.kuhn@ovgu.de`

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
`https://parcio.ovgu.de`

Outline

- Which technology improves at the fastest rate?
    1. Storage capacity
    2. Storage throughput
    3. Network throughput
    4. Memory throughput
    5. Computation

- Which overhead does deduplication introduce?
    1. Processor utilization
    2. Main memory utilization
    3. Storage utilization
    4. All of the above

- Which overhead does compression introduce?
  1. Processor utilization
  2. Main memory utilization
  3. Storage utilization
  4. All of the above

- Which compression algorithm would you use for archival?
  1. lz4
  2. lz4hc
  3. xz
  4. zstd

Outline

Exascale

- Supercomputers are getting more powerful all the time
  - Storage systems get more capacity and throughput
  - Scalability requirements are increased by higher process counts
- Amount of data keeps growing
  - Supercomputers have more than 1 PB of main memory
  - Even at 1 TB/s, a checkpoint would take more than 15 minutes
- Storage hardware is becoming more efficient
  - Software has to keep up with efficiency increases
  - I/O stacks have traditionally been heavy-weight

Exascale...

- One file per process does not scale
  - TaihuLight has more than 40,000 compute nodes
  - Large systems can have more than 10,000,000 cores
- POSIX is still widely used
  - Strict coherence and consistency requirements
  - Changes have to be visible globally immediately
- Traditional file systems are often implemented in the kernel
  - Other components already bypass the kernel due to performance
  - For example, InfiniBand uses kernel bypass to reduce overhead

Outline

Memory and Storage Hierarchy

- Memory and storage hierarchy has several levels
  - L1, L2, L3 cache, RAM, SSD, HDD and tape
- Huge latency gap between RAM and SSD
  - Significant performance degradation if data is not in RAM
  - Network causes additional overhead in distributed contexts
- Gap is especially pronounced on supercomputers
  - Data is either locally in RAM or in the parallel distributed file system
- New technologies are supposed to close this gap
  - NVRAM, NVMe, 3D XPoint etc.
  - Introduces additional hierarchy levels

- Very low latency between CPU and caches
  - One cycle takes 0.5–0.33 ns (2–3 GHz)
  - RAM is already significantly slower

Level	Latency
L1 Cache	≈ 1 ns
L2 Cache	≈ 5 ns
L3 Cache	≈ 10 ns
RAM	≈ 100 ns

[Bonér, 2012] [Huang et al., 2014]

Memory and Storage Hierarchy...

- Very low latency between CPU and caches
  - One cycle takes 0.5–0.33 ns (2–3 GHz)
  - RAM is already significantly slower
- There is a factor of 1,000 between RAM and SSD
  - Network latency in distributed systems

Level	Latency
L1 Cache	≈ 1 ns
L2 Cache	≈ 5 ns
L3 Cache	≈ 10 ns
RAM	≈ 100 ns
SSD	≈ 100,000 ns
HDD	≈ 10,000,000 ns
Tape	≈ 50,000,000,000 ns

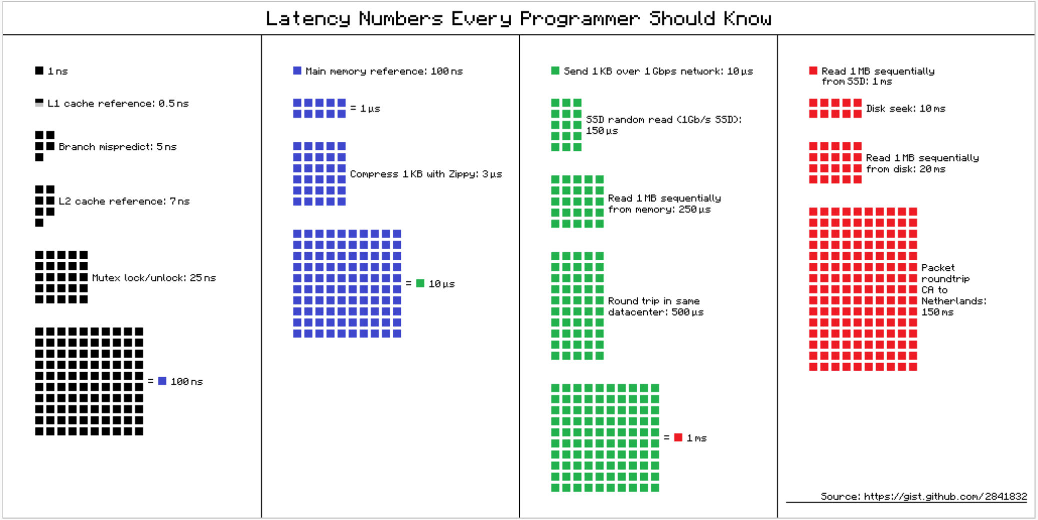
[Bonér, 2012] [Huang et al., 2014]

Memory and Storage Hierarchy...

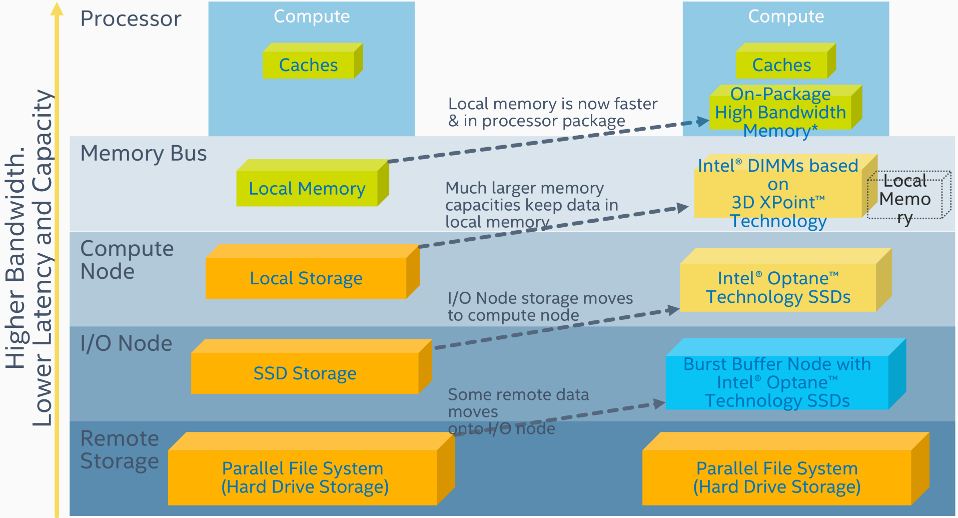
- Very low latency between CPU and caches
  - One cycle takes 0.5–0.33 ns (2–3 GHz)
  - RAM is already significantly slower
- There is a factor of 1,000 between RAM and SSD
  - Network latency in distributed systems
- Multiple intermediate levels in future systems
  - NVRAM will also offer novel approaches

Level	Latency
L1 Cache	$\approx$ 1 ns
L2 Cache	$\approx$ 5 ns
L3 Cache	$\approx$ 10 ns
RAM	$\approx$ 100 ns
NVRAM	$\approx$ 1,000 ns
NVMe	$\approx$ 10,000 ns
SSD	$\approx$ 100,000 ns
HDD	$\approx$ 10,000,000 ns
Tape	$\approx$ 50,000,000,000 ns

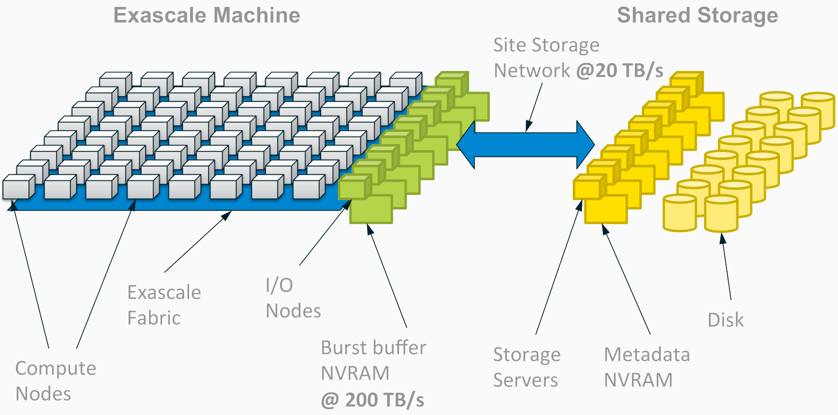
[Bonér, 2012] [Huang et al., 2014]


Latency Numbers Every Programmer Should Know
1 ns
L1 cache reference: 0.5 ns
Branch mispredict: 5 ns
L2 cache reference: 7 ns
Mutex lock/unlock: 25 ns
= 100 ns
Main memory reference: 100 ns
= 1 µs
Compress 1 KB with Zippy: 3 µs
= 10 µs
Send 1 KB over 1 Gbps network: 10 µs
SSD random read (1Gb/s SSD): 150 µs
Read 1 MB sequentially from memory: 250 µs
Round trip in same datacenter: 500 µs
= 1 ms
Read 1 MB sequentially from SSD: 1 ms
Disk seek: 10 ms
Read 1 MB sequentially from disk: 20 ms
Packet roundtrip CA to Netherlands: 150 ms
Source: https://gist.github.com/2841832

[Bonér, 2012]


Higher Bandwidth.
Lower Latency and Capacity
Processor
Compute
Caches
Compute
Caches
On-Package
High Bandwidth
Memory*
Local memory is now faster
& in processor package
Memory Bus
Local Memory
Much larger memory
capacities keep data in
local memory
Intel® DIMMs based on 3D XPoint™ Technology
Local Memo ry
Compute Node
Local Storage
Intel® Optane™ Technology SSDs
I/O Node storage moves
to compute node
I/O Node
SSD Storage
Burst Buffer Node with Intel® Optane™ Technology SSDs
Some remote data moves onto I/O node
Remote Storage
Parallel File System
(Hard Drive Storage)
Parallel File System
(Hard Drive Storage)

[Gorda, 2016]



[Gorda, 2013]

- I/O nodes are equipped with burst buffers and close to the compute nodes
- Network between I/O nodes and storage servers is slower

Exascale…

- I/O nodes can take over additional tasks
  - Certain computations and transformations
  - Scheduling and aggregating I/O operations
- Data can be reorganized for more efficient access
  - For instance, row- and column-wise storage
  - Requires knowledge about the underlying data format
- Computational power is higher closer to the compute nodes
  - Makes the most sense while data is being produced
  - Care has to be taken to not influence performance negatively

- I/O behavior is often not uniform
  - Applications compute and then write a shared checkpoint
  - High I/O load during checkpointing, no I/O activity afterwards
- I/O spikes can slow down applications
  - Multiple applications performing I/O in parallel
  - Storage systems are usually not designed to handle high spikes
    - Example Mistral: $\approx$ 20 TB/s (compute nodes) vs. $\approx$ 0.5 TB/s (file system)
- Guaranteeing high throughput can become expensive
  - HDDs for capacity, SSDs for throughput (and latency)
  - Network introduces additional performance constraints

Burst Buffers…

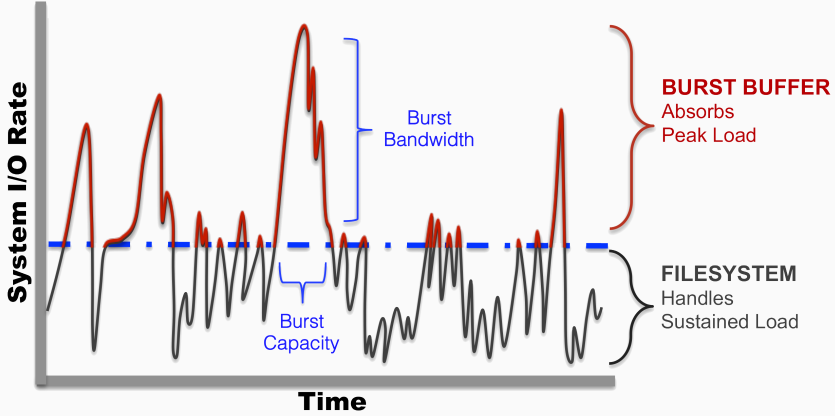
- Applications do not coordinate their I/O phases
    - Could be used to keep the I/O load balanced
    - Very complex to realize since it depends on timing etc.
        - Involves the application, the scheduler and the file system
- File systems are usually shared resources
    - There are quality of service approaches for I/O
    - Applications could communicate their requirements to the scheduler

Quiz

- How much storage bandwidth is used on average?
    1. 99 %
    2. 50 %
    3. 33 %
    4. 5 %

Analysis of a major HPC production storage system

- 99% of the time, storage BW utilization < 33% of max
- 70% of the time, storage BW utilization < 5% of max



[Vildibill, 2015]

- Compute nodes achieve higher throughput
  - Up to 100 GiB/s instead of 10–20 GiB/s
  - No change for slow applications
- Applications finish earlier
  - Can spend more time performing computation

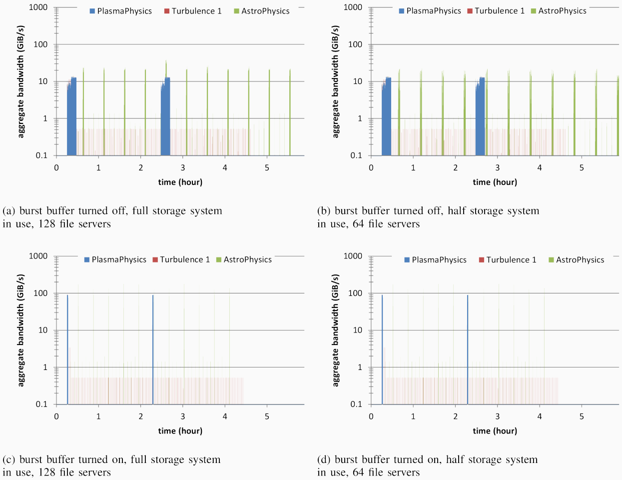
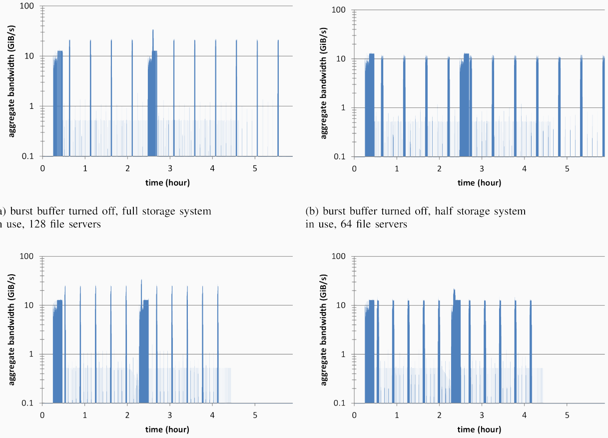


Fig. 5: Ten second average data transfer rate for the compute nodes observed during the multiapplication simulations.

[Liu et al., 2012]

- No throughput changes on storage servers
  - Depends on number of storage servers
- Utilization increased
  - Idle times are reduced



(a) burst buffer turned off, full storage system in use, 128 file servers

(b) burst buffer turned off, half storage system in use, 64 file servers

(c) burst buffer turned on, full storage system in use, 128 file servers

(d) burst buffer turned on, half storage system in use, 64 file servers

Fig. 6: Ten second average data transfer rate for the external storage system observed during the multiapplication simulations.

[Liu et al., 2012]

Burst Buffers…

- Burst buffers allow saving significant amounts of money
  - Storage systems do not have to be designed to handle high I/O spikes
  - Allows using a smaller storage system or one with less throughput
- Might also allow using slower storage technologies
  - Ethernet instead of InfiniBand (or cheaper InfiniBand)
  - HDDs with 5,400 RPM instead of 7,200 RPM
- Makes it possible to increase device utilization
  - Burst buffers can also absorb problematic I/O patterns such as random I/O
  - I/O operations are "pre-processed" and then forwarded to storage system

Accelerators

- Data reduction etc. can be compute-intensive
  - Deduplication, compression etc. require compute power
- GPUs are often not suitable since the PCIe bus limits I/O
  - At most $\approx$ 16 GB/s (PCIe 3.0) or $\approx$ 32 GB/s (PCIe 4.0)
- Several acceleration interfaces could be used in the future
  - Intel's processors support QuickAssist with DEFLATE and LZS (since Haswell)
  - Intel is working on socketed accelerators, which can access RAM directly
  - PCIe 5.0 with $\approx$ 64 GB/s

Outline

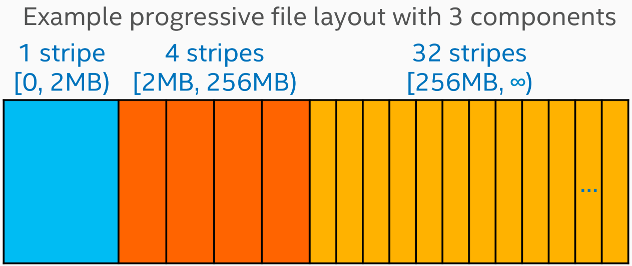
Lustre

- Classical parallel distributed file systems are not enough anymore
  - Should serve as the basis for future storage systems
  - They are the only production-ready solution at the moment
- Lustre is in active development [Gorda, 2016]
  - Encryption
  - Compression
  - Complex data layouts
  - Further I/O optimizations

Lustre...

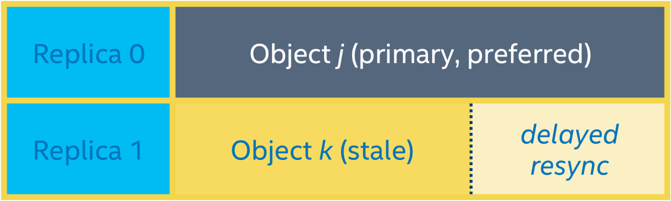
- Encryption is increasingly important
  - Governments, military, classified research etc.
- Support for multiple access levels is necessary
  - Unclassified, confidential, secret, top secret
  - Data is not allowed to be transferred across levels
- Authentication and authorization are important components
  - Can be implemented using Kerberos, which is widely used
- Data is encrypted in flight to prevent unauthorized access
  - Support for encryption at rest has been recently added

- Applications and data formats have different data distribution requirements
  - Small files should only be distributed across a few OSTs
  - Large files should be distributed across as many OSTs as possible
- Lustre can adapt striping parameters intelligently
  - Goal is to minimize overhead and maximize performance



[Gorda, 2016]

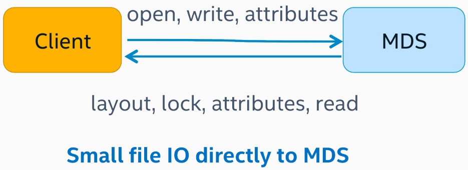
- Replicating files can be useful for popular data
  - Replication can be set on a per-file basis
  - Can be used to achieve high availability, robustness, higher read throughput, migration across storage classes etc.
- Synchronization is done in the background



Overlapping (mirror) layout

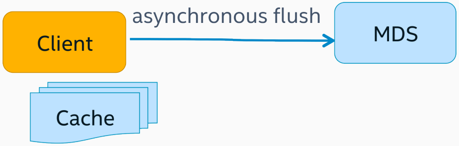
[Gorda, 2016]

- Small files should be stored directly on the MDT
  - Similar to optimizations in ext4, ZFS etc.
  - Can reduce communication overhead by not talking to OSTs
  - Potential for further optimizations such as readahead
- MDTs are typically optimized for small accesses
  - Large accesses still handled by the OSTs
- Data might have to be migrated when files grow



[Gorda, 2016]

- Metadata operations are usually small
  - Corresponds to high network overhead
- Caching allows aggregating multiple operations
  - Requires locks to avoid conflicts from concurrent operations
  - Example: Lock a directory and create many files within it



Client Metadata Operations Cache

[Gorda, 2016]

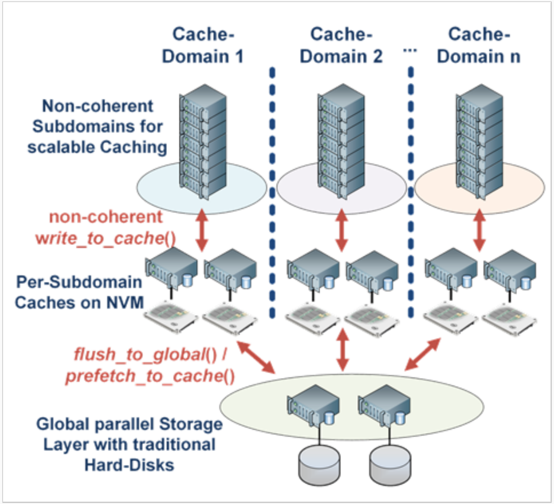
POSIX

- POSIX is still a major limitation for I/O performance and scalability
    - POSIX is very portable but poses a performance bottleneck
- There are a few existing alternatives
    - MPI-IO still uses the POSIX interface anyway
    - POSIX file systems are used locally in many cases
- New interfaces and semantics are being investigated
    - Object stores often provide enough features for HPC I/O
    - When using POSIX, its semantics is usually relaxed

POSIX…

- One idea is to not provide global coherence anymore
  - File system is instead partitioned into non-coherent zones
  - For instance, using burst buffers and forwarders
- Example: Cache domains as used in BeeGFS
  - Applications are running in different domains
  - Data is first written to the non-coherent cache
  - Caches can be located in a local NVRAM
  - Data is then migrated from the cache into the file system

- Non-coherent domains can scale
  - Data does not have to be synchronized across all applications
  - Applications typically do not access same output data
- Flushed to file system afterwards
  - Similar to burst buffer concept



[ThinkParQ, 2017]

DAOS

- DAOS is a holistic approach for a new storage stack
  - Distributed Application Object Storage (DAOS)
- DAOS supports multiple storage models
  - Arrays and records as base objects
  - Objects consist of arrays and records (key-array)
  - Containers contain objects
  - Storage pools consist of containers
- DAOS supports versioning data
  - Operations are performed as transactions
  - Transactions are merged and persisted as epochs
- Makes extensive use of modern storage technologies

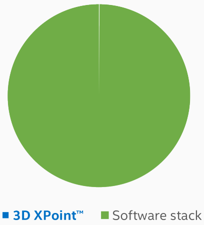
Quiz

- How much overhead does the I/O software stack introduce?
  1. 99 %
  2. 50 %
  3. 33 %
  4. < 1 %

- I/O latencies are becoming problematic
  - Additional software layers introduce overhead
- I/O granularity needs to be adapted
  - Often still at 1 MiB, soon 16 MiB, which might cause additional conflicts
    - Network and storage devices require larger accesses

msec vs. µsec

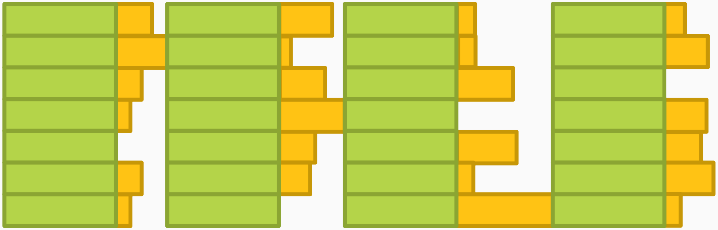
µsec vs. µsec



nsec vs. µsec

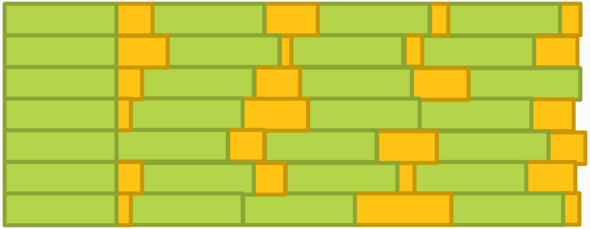
[Dilger, 2017]

- I/O is typically performed synchronously
  - Applications have to wait for the slowest process/thread
  - Results in waiting times in which processors are idle
- I/O variability is the norma
  - Storage systems are shared resources, others can influence performance
  - Quality of service or other performance guarantees are rare



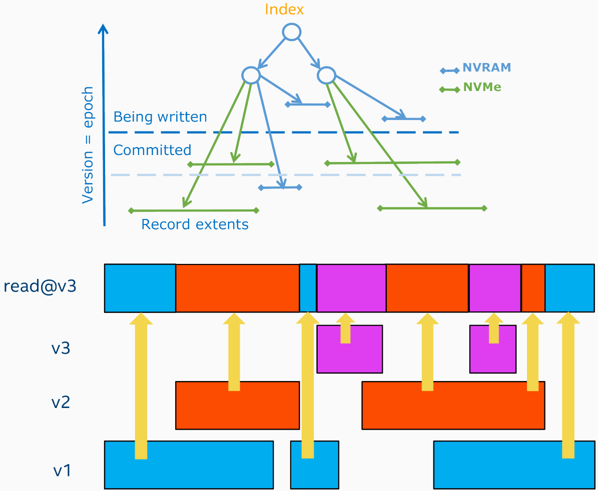
[Gorda, 2013]

- I/O should happen asynchronously to reduce idle times
  - Processes/threads do not have to wait for others anymore
- Raises the problem of file consistency
  - Currently, a file is consistent when all processes have finished I/O
  - File is only consistent as long as it is not modified again



[Gorda, 2013]

- Solution: Transactions and epochs
  - Operations are performed in transactions
  - Multiple transactions are merged into an epoch
- Epochs are globally consistent
  - Epochs are on a per-object basis
  - Eliminates coherence problems when reading data
  - Epochs can use copy on write for efficiently storing versions

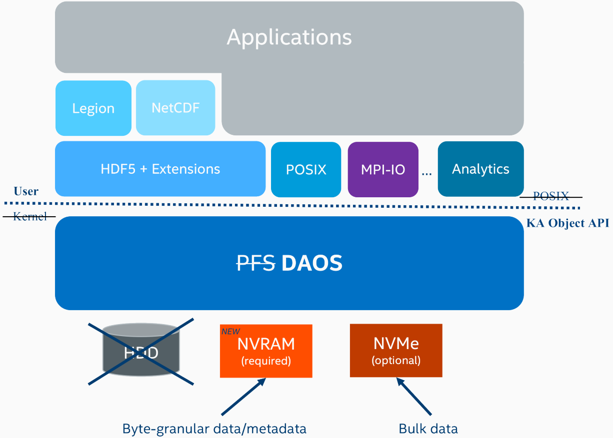


[Dilger, 2017]

DAOS…

- DAOS supports several I/O interfaces natively
  - Makes supporting legacy applications much easier
- HDF5 is mapped onto DAOS's objects
  - An HDF5 file corresponds to a DAOS container
  - Mapping can be used to reorganize data for efficient access etc.
- Other I/O interfaces can be added on top
  - POSIX and MPI-IO for legacy applications
  - Big data interfaces for MapReduce etc.
  - S3, NFS, block devices etc.

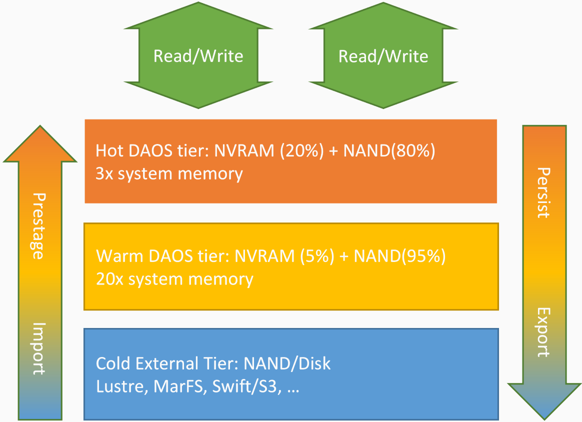
- Functionality moved to user space
  - Kernel bypass to eliminate costly context switches
- Software stack for NVRAM/NVMe
  - HDDs are controlled by existing file systems such as Lustre



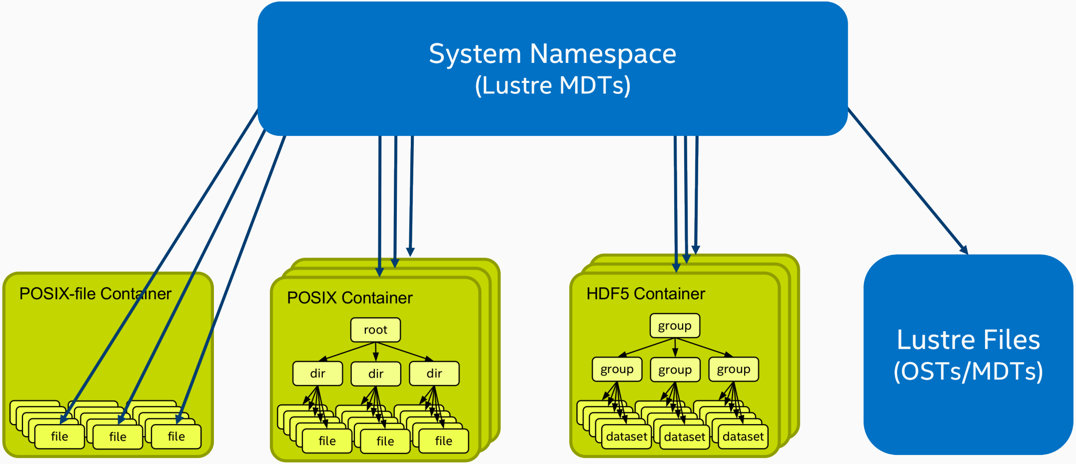
[Dilger, 2017]

DAOS…

- Hot and warm data managed by DAOS
  - Cold data managed by file system
- Prestage migrates data into cache
  - Persist stores data
- Import/export to/from cold storage
  - File systems like Lustre
  - Object stores like S3



[Dilger, 2017]



Legacy Apps
Near-POSIX compliant

Well-behaved POSIX apps
Highly scalable

Data-model/domain–
specific library

Libraries, binaries, user files,
Migration path

[Dilger, 2017]

Big Data

- Big data technologies are widely used
  - Big data software is often not as performant as HPC software
- Hadoop is an important big data component and uses HDFS
  - Data is copied to local storage devices
  - Communication happens via HTTP
- HPC software increasingly supports big data use cases
  - Lustre, OrangeFS etc.
- Problem: Two completely separate software stacks

Big Data...

- Big data often uses commodity hardware
  - Ethernet, local storage, HTTP etc.
- HPC tuned for high performance
  - InfiniBand, dedicated storage nodes, accelerators etc.

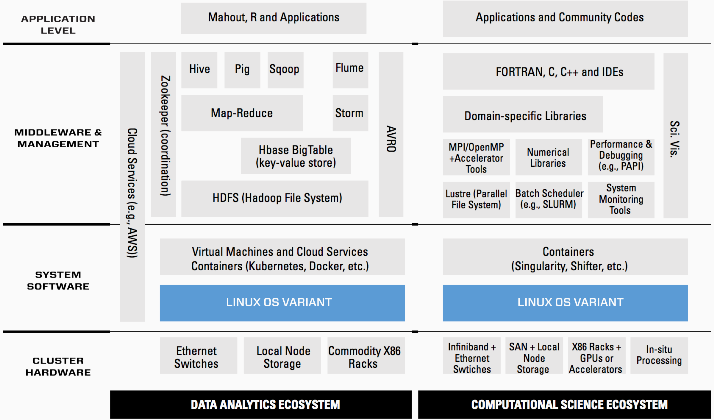
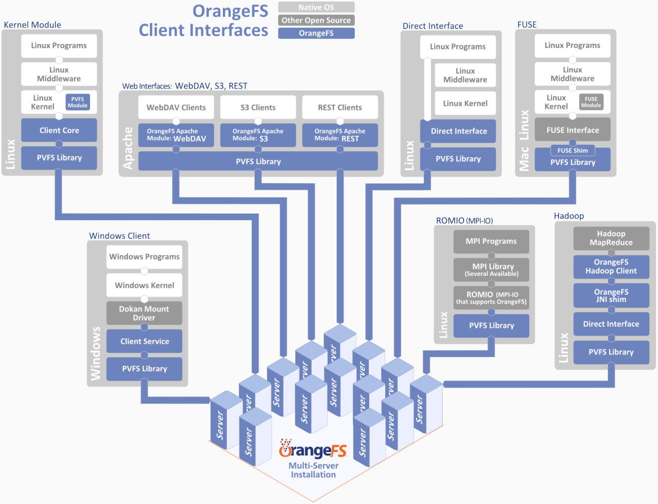


Figure 1: Different software ecosystems for high-end Data Analytics and for traditional Computational Science. [Credit: Reed and Dongarra [66]]

[Russell, 2018] [Andre et al., 2018]

- OrangeFS offers several I/O interfaces
  - POSIX compatibility via kernel module, FUSE or direct interface
  - HPC support via MPI-IO
  - Web interfaces for cloud
  - Big data support using Hadoop interface



[OrangeFS Development Team, 2021]

- HDFS uses node-local storage
  - Best case: Data can be accessed locally
- OrangeFS uses storage servers
  - Data is always accessed remotely

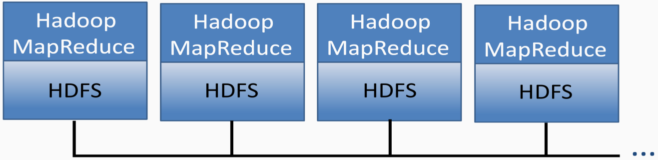


Fig. 2: Typical Hadoop with HDFS local storage (HDFS in short).

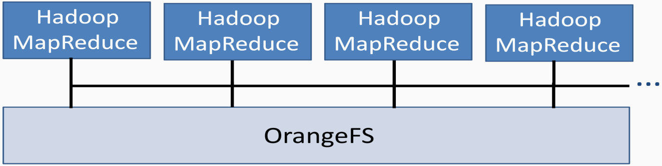


Fig. 3: Hadoop with the OrangeFS dedicated storage (OFS in short).

[Li et al., 2016]

Big Data…

- Approaches from the big data and cloud fields are also interesting for HPC
  - Elasticity could allow adapting file systems dynamically
  - Adding and removing file system servers on demand
- Object stores are being used for storing data
  - POSIX file system functionalities are often not required
  - MPI-IO's features can be mapped to object stores

Outline

Summary

- New hardware technologies will change the storage stack
  - New hierarchy levels provided by NVRAM and NVMe
  - Systems will become complexer but also more performant
  - Burst buffers offer possibilities for reducing costs
  - Data transformation can be deployed across the stack
- Current I/O software is being redesigned from the ground up
  - Applications can continue using existing high-level interfaces
  - POSIX limits performance and is often not necessary
  - New approaches from the big data and cloud fields will be integrated

References

[Andre et al., 2018] Andre, J.-C., Antoniu, G., Asch, M., Sala, R. B., Beck, M., Beckman, P., Bidot, T., Bodin, F., Cappello, F., Choudhary, A., de Supinski, B., Deelman, E., Dongarra, J., Dubey, A., Fox, G., Fu, H., Girona, S., Gropp, W., Heroux, M., Ishikawa, Y., Keahey, K., Keyes, D., Kramer, W., Lavignon, J.-F., Lu, Y., Matsuoka, S., Mohr, B., Moore, T., Reed, D., Requena, S., Saltz, J., Schulthess, T., Stevens, R., Swany, M., Szalay, A., Tang, W., Varoquaux, G., Vilotte, J.-P., Wisniewski, R., Xu, Z., and Zacharov, I. (2018). **Big Data and Extreme-Scale Computing: Pathways to Convergence.** Technical report, EXDCI Project of EU-H2020 Program and University of Tennessee. `http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/bdec2017pathways.pdf`.

[Bonér, 2012] Bonér, J. (2012). **Latency Numbers Every Programmer Should Know.** `https://gist.github.com/jboner/2841832`.

[Dilger, 2017] Dilger, A. (2017). **DAOS: Scale-out Object Storage for NVRAM.** `https://www.dagstuhl.de/17202`.

References ...

[Gorda, 2013] Gorda, B. (2013). **HPC Technologies for Big Data.** http://www.hpcadvisorycouncil.com/events/2013/Switzerland-Workshop/Presentations/Day_2/3_Intel.pdf.

[Gorda, 2016] Gorda, B. (2016). **HPC Storage Futures – A 5-Year Outlook.** http://lustre.ornl.gov/ecosystem-2016/documents/keynotes/Gorda-Intel-keynote.pdf.

[Huang et al., 2014] Huang, J., Schwan, K., and Qureshi, M. K. (2014). **NVRAM-aware Logging in Transaction Systems.** *Proc. VLDB Endow.*, 8(4):389–400.

[Li et al., 2016] Li, Z., Shen, H., Denton, J., and Ligon, W. (2016). **Comparing application performance on hpc-based hadoop platforms with local storage and dedicated storage.** In *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pages 233–242.

References ...

[Liu et al., 2012] Liu, N., Cope, J., Carns, P. H., Carothers, C. D., Ross, R. B., Grider, G., Crume, A., and Maltzahn, C. (2012). **On the role of burst buffers in leadership-class storage systems.** In *IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, April 16-20, 2012, Asilomar Conference Grounds, Pacific Grove, CA, USA*, pages 1–11. IEEE Computer Society.

[OrangeFS Development Team, 2021] OrangeFS Development Team (2021). **OrangeFS Documentation.** https://docs.orangefs.com/.

[Russell, 2018] Russell, J. (2018). **New Blueprint for Converging HPC, Big Data.** https://www.hpcwire.com/2018/01/18/new-blueprint-converging-hpc-big-data/.

[ThinkParQ, 2017] ThinkParQ (2017). **BeeGFS Cache API.** https://www.beegfs.io/wiki/CacheAPI.

[Vildibill, 2015] Vildibill, M. (2015). **Advanced IO Architectures.** http://storageconference.us/2015/Presentations/Vildibill.pdf.