Convergence of High Performance Computing and Big Data

Michael Kuhn
michael.kuhn@informatik.uni-hamburg.de

Scientific Computing
Department of Informatics
Universität Hamburg
https://wr.informatik.uni-hamburg.de

2018-05-23

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

About Us: Scientific Computing

- Analysis of parallel I/O
- I/O & energy tracing tools
- Middleware optimization
- Alternative I/O interfaces
- Data reduction techniques
- Cost & energy efficiency

We are an Intel Parallel Computing Center for Lustre
(“Enhanced Adaptive Compression in Lustre”)

Command Line Meets Big Data

- Big Data tools may not always be the best suited for a problem [1]
  - Use MapReduce to compute win/loss ratios of chess games
- Archive has a size of 1.75 GB and contains two million chess games

Command Line Meets Big Data

- Big Data tools may not always be the best suited for a problem [1]
  - Use MapReduce to compute win/loss ratios of chess games
- Archive has a size of 1.75 GB and contains two million chess games
  - MapReduce job took roughly 26 minutes (which equals a throughput of 1.14 MB/s)

Command Line Meets Big Data

- Big Data tools may not always be the best suited for a problem [1]
  - Use MapReduce to compute win/loss ratios of chess games
- Archive has a size of 1.75 GB and contains two million chess games
  - MapReduce job took roughly 26 minutes (which equals a throughput of 1.14 MB/s)
- Archive has a size of 3.46 GB
  - Command line tools can perform the same job in roughly 12 seconds (270 MB/s)

Command Line Meets Big Data

- Big Data tools may not always be the best suited for a problem [1]
  - Use MapReduce to compute win/loss ratios of chess games
- Archive has a size of 1.75 GB and contains two million chess games
  - MapReduce job took roughly 26 minutes (which equals a throughput of 1.14 MB/s)
- Archive has a size of 3.46 GB
  - Command line tools can perform the same job in roughly 12 seconds (270 MB/s)

```
find . -type f -name '*.pgn' -print0 | xargs -0 -n4 -P4 mawk '/Result/ { split($0, a,
    ↪ "-"); res = substr(a[1], length(a[1]), 1); if (res == 1) white++; if (res ==
    ↪ 0) black++; if (res == 2) draw++ } END { print white+black+draw, white, black,
    ↪ draw }' | mawk '{games += $1; white += $2; black += $3; draw += $4; } END {
    ↪ print games, white, black, draw }'
```

Software Ecosystems [4, 2]

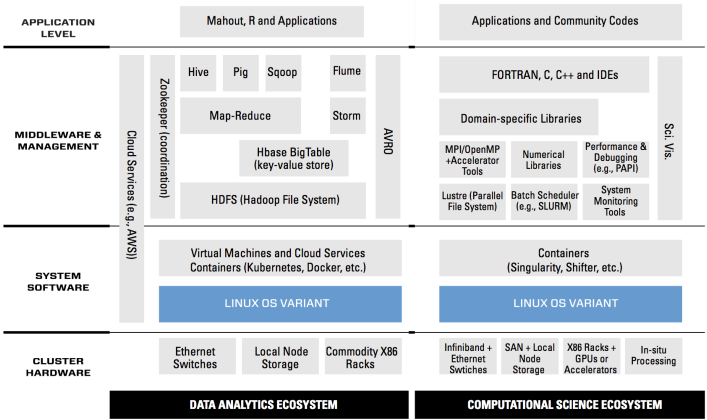


Figure 1: Different software ecosystems for high-end Data Analytics and for traditional Computational Science. [Credit: Reed and Dongarra [66]]

Architectures

- Until 2005: performance increase via clock rate
- Since 2005: performance increase via core count
  - Clock rate can not be increased further
  - Energy consumption/heat dissipation depends on clock rate
  - The largest supercomputers have more than 10,000,000 cores
- Categorization using memory connection
  - Shared and distributed memory
  - Hybrid systems are commonly found in practice

Threads vs. Processes

- Threads share one common address space
  - Communication using shared memory segments
  - If one thread crashes, all of them crash
- Processes have their own address spaces
  - Communication usually happens via messages
  - Overhead is typically higher than for shared memory
- Hybrid approaches in practice
  - A few processes per node (for example, one per socket)
  - Many threads per process (for example, one per core)

Parallelization

- Parallel applications run on multiple nodes
  - Independent processes, communication via messages
  - OpenMP threads within processes
- MPI provides communication operations
  - MPI is the de-facto standard
  - Process groups, synchronization, communication, reduction etc.
  - Point-to-point and collective communication
- MPI also supports input/output
  - Parallel I/O for shared files

Latencies

Level	Latency
L1 cache	≈ 1 ns
L2 cache	≈ 5 ns
L3 cache	≈ 10 ns
RAM	≈ 100 ns
InfiniBand	≈ 500 ns
Ethernet	≈ 100,000 ns
SSD	≈ 100,000 ns
HDD	≈ 10,000,000 ns

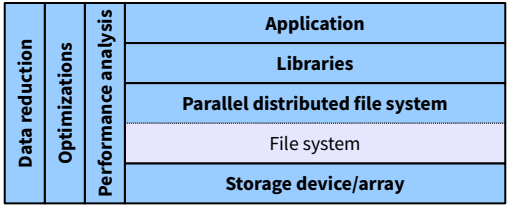
Table: Latencies [5, 3]

- Processors require data fast
  - Caches should be used optimally
- Additional latency due to I/O and network

Performance and Energy Efficiency

- Supercomputers are very complex
  - Performance yield is typically not optimal
  - Interaction of many different components
    - Processors, caches, main memory, network, storage system
  - Performance analysis is an important topic
- Supercomputers are very expensive
  - Should be used as efficiently as possible
  - Procurement costs: € 40,000,000–250,000,000
  - Operating costs (per year):
    - Sunway TaihuLight (rank 1): 15.4 MW ≈ € 15,400,000 (in Germany)
    - DKRZ (rank 42): 1.1 MW ≈ € 1,100,000

I/O Layers



- I/O is often responsible for performance problems
  - High latency causes idle processors
  - I/O is often still serial
- I/O stack is layered
  - Many different components are involved in storing data
  - An unoptimized layer can significantly decrease performance

Hard Disk Drives

- First HDD: 1956
  - IBM 350 RAMAC (3,75 MB, 8,8 KB/s, 1.200 RPM)
- HDD development
  - Capacity: factor of 100 every 10 years
  - Throughput: factor of 10 every 10 years

Parameter	Started with	Developed to	Improvement
Capacity (formatted)	3.75 megabytes[9]	eight terabytes	two-million-to-one
Physical volume	68 cubic feet (1.9 m^3)[c][3]	2.1 cubic inches (34 cc)[10]	57,000-to-one
Weight	2,000 pounds (910 kg)[3]	2.2 ounces (62 g)[10]	15,000-to-one
Average access time	about 600 milliseconds[3]	a few milliseconds	about 200-to-one
Price	US$9,200 per megabyte[11][*dubious – discuss*]	< $0.05 per gigabyte by 2013[12]	180-million-to-one
Areal density	2,000 bits per square inch[13]	826 gigabits per square inch in 2014[14]	> 400-million-to-one

Figure: HDD development [12]

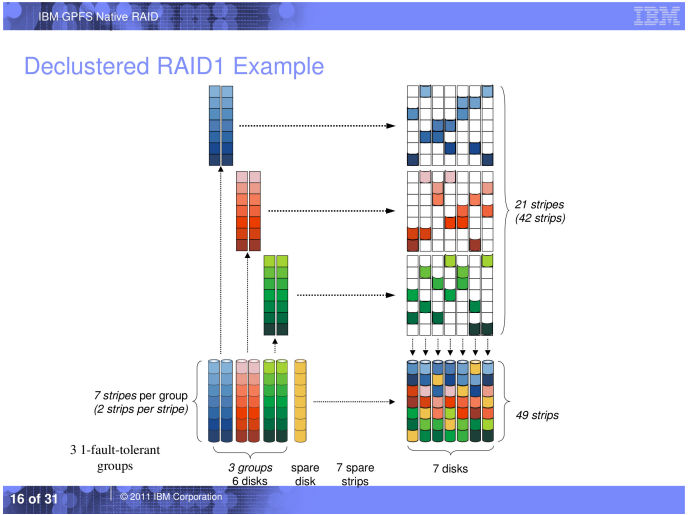
Solid-State Drives

- Benefits
  - Read/write throughput: factor of 10–20
  - Latency: factor of 100
  - Energy consumption: factor of 1–10
- Drawbacks
  - Price: factor of 7–10
  - Write cycles: 10.000–100.000
  - Complexity
    - Different optimal access sizes for reads and writes
    - Address translation, thermal issues etc.

RAID

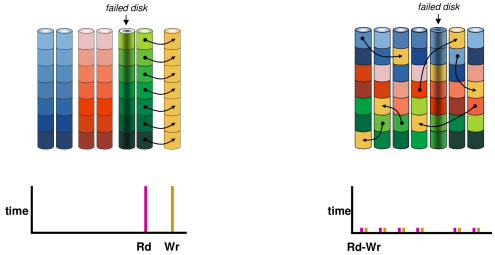
- RAID
  - RAID 0: striping
  - RAID 1: mirroring
  - RAID 2/3: bit/byte striping
  - RAID 4: block striping
  - RAID 5/6: block striping
- Failures
  - HDDs usually have roughly the same age
  - Fabrication defects within same batch
- Reconstruction
  - Read errors on other HDDs
  - Duration (30 min in 2004, 11 h in 2017)

RAID... [10]


IBM GPFS Native RAID
IBM
Declustered RAID1 Example
21 stripes
(42 strips)
7 stripes per group
(2 strips per stripe)
49 strips
3 1-fault-tolerant
groups
3 groups
6 disks
spare
disk
7 spare
strips
7 disks
16 of 31
© 2011 IBM Corporation

RAID... [10]

Declustered RAID Rebuild Example – Single Fault



Rebuild activity confined to just a few disks – slow rebuild, disrupts user programs

Rebuild activity spread across many disks, faster rebuild or less disruption to user programs

Tasks

- Structure
  - Typically files and directories
  - Hierarchical organization
  - Other approaches: tagging
- Management of data and metadata
  - Block allocation
  - Access permissions, timestamps etc.
- File systems use underlying storage devices or arrays
  - Logical Volume Manager (LVM) and/or mdadm

I/O Interfaces

- Requests are realized through I/O interfaces
  - Forwarded to the file system
  - Different abstraction levels
- Low-level functionality: POSIX, MPI-IO etc.
- High-level functionality: HDF, NetCDF etc.

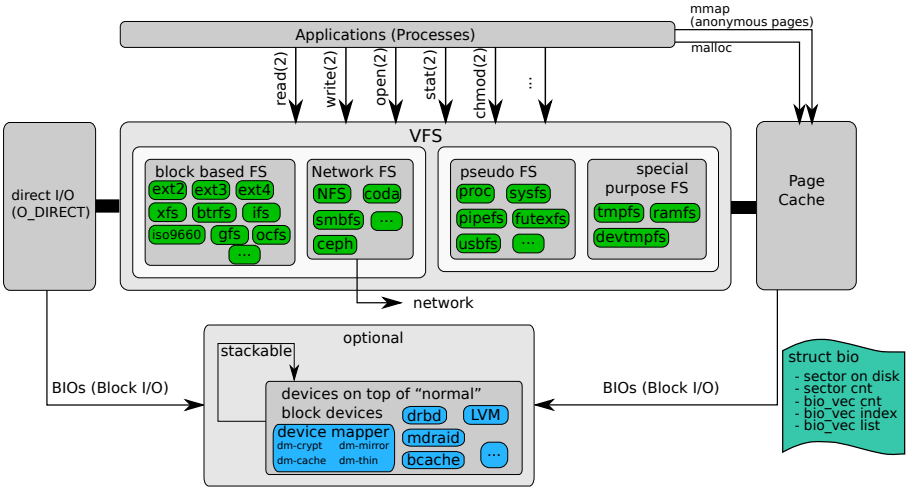
```
1 fd = open("/path/to/file", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
2 nb = write(fd, data, sizeof(data));
3 rv = close(fd);
4 rv = unlink("/path/to/file");
```

- Initial access via path
  - Afterwards access via file descriptor (few exceptions)
- Functions are located in `libc`
  - Library executes system calls

Virtual File System (Switch)

- Central file system component in the kernel
  - Sets file system structure and interface
- Forwards applications' requests based on mount point
- Enables supporting multiple different file systems
  - Applications are still portable due to POSIX
- POSIX: standardized interface for all file systems
  - Syntax
    - open, close, creat, read, write, lseek, chmod, chown, stat etc.
  - Semantics
    - `write`: *"POSIX requires that a read(2) which can be proved to occur after a write() has returned returns the new data. Note that not all filesystems are POSIX conforming."*

Virtual File System (Switch)… [11]



The Linux Storage Stack Diagram
http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Created by Werner Fischer and Georg Schönberger
License: CC-BY-SA 3.0, see http://creativecommons.org/licenses/by-sa/3.0/

File System Objects

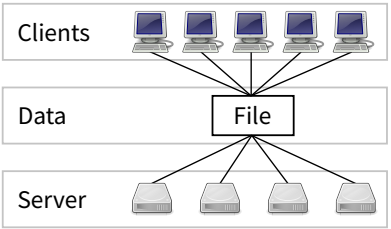
- User vs. system view
  - Users see files and directories
  - System manages inodes
    - Relevant for `stat` etc.
- Files
  - Contain data as byte arrays
  - Can be read and written (explicitly)
  - Can be mapped to memory (implicit)
- Directories
  - Contain files and directories
  - Structures the namespace

Modern File Systems

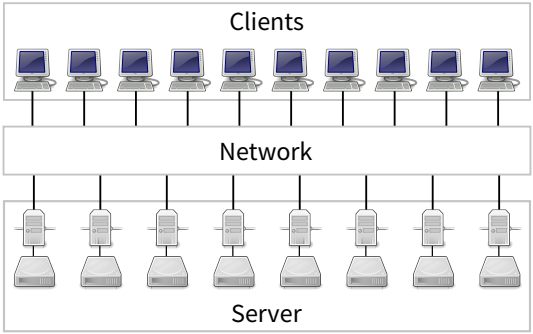
- File system demands are growing
  - Data integrity, storage management, convenience functionality
- Error rate for SATA HDDs: 1 in 10^{14} to 10^{15} bits [9]
  - That is, one bit error per 12,5–125 TB
  - Additional bit errors in RAM, controller, cable, driver etc.
- Error rate can be problematic
  - Amount can be reached in daily use
  - Bit errors can occur in the superblock
- File system does not have knowledge about storage array
  - Knowledge is important for performance
  - For example, special options for ext4

Definition

- Parallel file systems
  - Allow parallel access to shared resources
  - Access should be as efficient as possible
- Distributed file systems
  - Data and metadata is distributed across multiple servers
  - Single servers do not have a complete view
- Naming is inconsistent
  - Often just “parallel file system” or “cluster file system”



Architecture

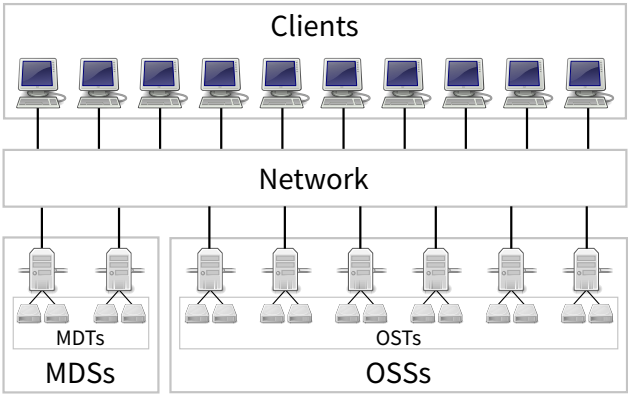


- Access to file system via I/O interface
  - Typically standardized, frequently POSIX
- Interface consists of syntax and semantics
  - Syntax defines operations, semantics defines behavior

Semantics

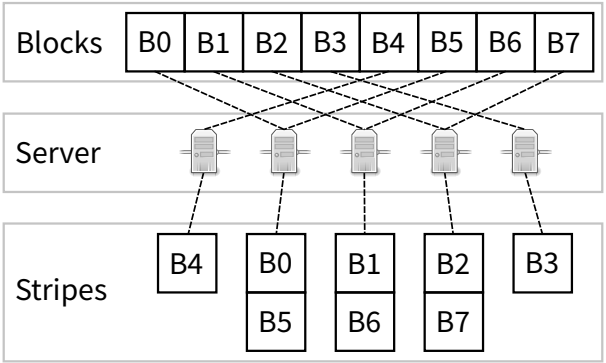
- POSIX has strong consistency/coherence requirements
  - Changes have to be visible globally after `write`
  - I/O should be atomic
- POSIX for local file systems
  - Requirements easy to support due to VFS
- Contrast: Network File System (NFS)
  - Same syntax, different semantics
- Session semantics
  - Changes only visible to other clients after session ends
  - `close` writes changes and returns potential errors
- MPI-IO has been designed for parallel I/O
  - Less strict for higher scalability

Architecture



- Separate servers for data and metadata
  - Produce different access patterns

Architecture…



- File is split into blocks, distributed across servers
  - In this case, with a round-robin distribution
- Distribution does not have to start at first server

Performance

- Parallel distributed file systems enable massive high performance storage systems
- Blizzard (DKRZ, GPFS)
  - Capacity: 7 PB
  - Throughput: 30 GB/s
- Mistral (DKRZ, Lustre)
  - Capacity: 54 PiB
  - Throughput: 450 GB/s (5.9 GB/s per node)
  - IOPS: 80,000 operations/s (phase 1 only)
- Titan (ORNL, Lustre)
  - Capacity: 40 PB
  - Throughput: 1.4 TB/s

Overview

- POSIX and MPI-IO can be used for parallel I/O
  - Both interfaces are not very convenient for developers
- Problems
  - Exchangeability of data, complex programming, performance
- Libraries offer additional functionality
  - Self-describing data, internal structuring, abstract I/O
- Alleviating existing problems
  - SIONlib (performance)
  - NetCDF, HDF (exchangeability)
  - ADIOS (abstract I/O)

SIONlib

- Mainly exists to circumvent deficiencies in existing file systems
  - On the one hand, problems with many files
    - Low metadata performance but high data performance
  - On the other hand, shared file access also problematic
    - POSIX requires locks, access pattern very important
- Offers efficient access to process-local files
  - Accesses are mapped to one or a few physical files
  - Aligned to file system blocks/stripes
- Backwards-compatible and convenient to use
  - Wrappers for `fread` and `fwrite`
  - Opening and closing via special functions

NetCDF

- Developed by Unidata Program Center
  - University Corporation for Atmospheric Research
- Mainly used for scientific applications
  - Especially in climate science, meteorology and oceanography
- Consists of libraries and data formats
  1. Classic format (CDF-1)
  2. Classic format with 64 bit offsets (CDF-2)
  3. NetCDF-4 format
- Data formats are open standards
  - Classic formats are international standards of the Open Geospatial Consortium

HDF

- Consists of libraries and data formats
  - Allows managing self-describing data
- Current version is HDF5
  - HDF4 is still actively supported
- Problems with previous versions
  - Complex API, limitations regarding 32 bit addressing
- Used as base for many higher-level libraries
  - NetCDF-4 (climate science)
  - NeXus (neutron, x-ray and muon science)

ADIOS

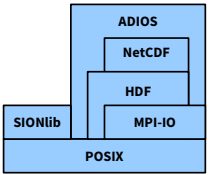
- ADIOS is heavily abstracted
  - No byte- or element-based access
  - Direct support for application data structures
- Designed for high performance
  - Mainly for scientific applications
  - Caching, aggregation, transformation etc.
- I/O configuration is specified via an XML file
  - Describes relevant data structures
  - Can be used to generate code automatically
- Developers specify I/O on a high abstraction level
  - No contact to middleware or file system

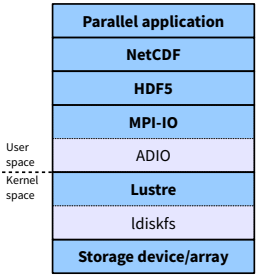
ADIOS...

```
1 <adios-config host-language="C">
2   <adios-group name="checkpoint">
3     <var name="rows" type="integer"/>
4     <var name="columns" type="integer"/>
5     <var name="matrix" type="double" dimensions="rows,columns"/>
6   </adios-group>
7   <method group="checkpoint" method="MPI"/>
8   <buffer size-MB="100" allocate-time="now"/>
9 </adios-config>
```

- Data is combined in groups
- I/O methods can be specified per group
- Buffer sizes etc. can be configured

Interaction…





- Data transformation
  - Transport through all layers
  - Loss of information
- Complex interaction
  - Optimizations and workarounds on all layers
  - Information about other layers
  - Analysis is complex
- Convenience vs. performance
  - Structured data in application
  - Byte stream in POSIX

Software Ecosystems [4, 2]

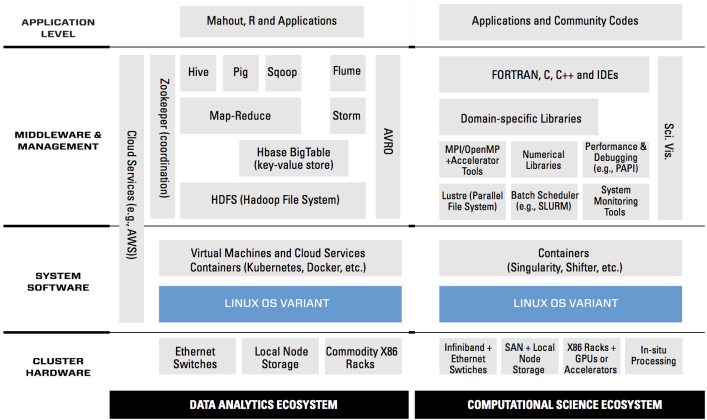
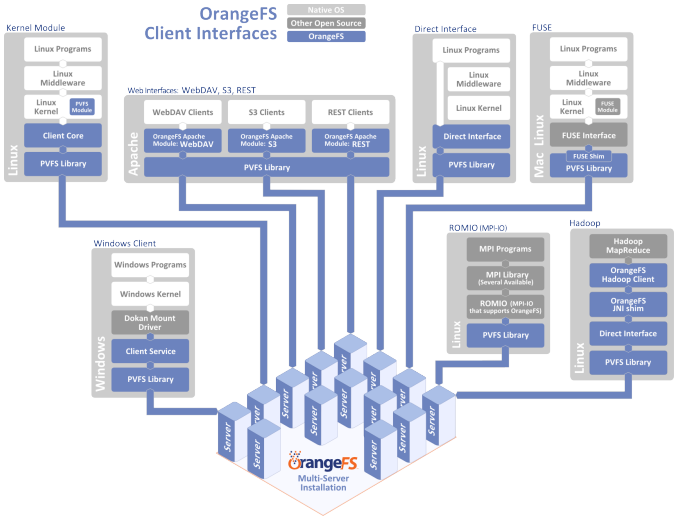


Figure 1: Different software ecosystems for high-end Data Analytics and for traditional Computational Science. [Credit: Reed and Dongarra [66]]

Software Ecosystems…

- The importance of Big Data is growing
  - Performance is often not optimal due to Ethernet, HTTP etc.
- Hadoop usually makes use of HDFS
  - Data is copied to local storage devices
  - Communication via HTTP
- HPC frameworks are starting to support Big Data applications
  - Lustre, OrangeFS etc.

Interfaces [8]



Big Data…

- Concepts from the Big Data and cloud environments can be applied in HPC
  - Elasticity to dynamically adapt the file system
  - For example, to add servers during high load situations
- Object stores provide efficient storage abstractions
  - Many applications do not need POSIX file systems
  - MPI-IO functionality can be mapped to object stores
- Investigated as part of the BigStorage project
  - Example: Týr is a blob storage system with support for transactions [7]
  - For more information, see `http://bigstorage-project.eu`

Big Data... [6]

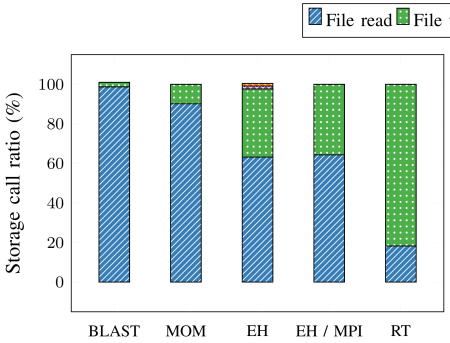


Fig. 1. Measured relative amount of different storage calls to the persistent file system for HPC applications

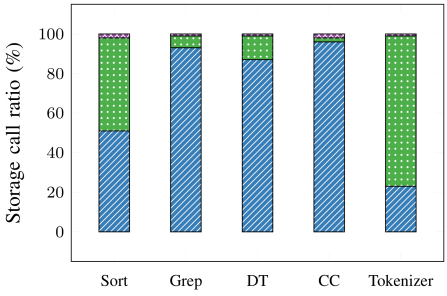


Fig. 2. Measured relative amount of different storage calls to the persistent file system for Big Data applications

Summary

- Achieving high performance I/O is a complex task
  - Many layers: storage devices, file systems, libraries etc.
- File systems organize data and metadata
  - Modern file systems provide additional functionality
- Parallel distributed file systems allow efficient access
  - Data is distributed across multiple servers
- I/O libraries facilitate ease of use
  - Exchangeability of data is an important factor
- HPC and Big Data technologies are converging
  - Enable more versatile and efficient systems

References I

[1] Adam Drake. Command-line Tools can be 235x Faster than your Hadoop Cluster. https://adamdrake.com/command-line-tools-can-be-235x-faster-than-your-hadoop-cluster.html.

[2] J.-C. Andre, G. Antoniu, M. Asch, R. Badia Sala, M. Beck, P. Beckman, T. Bidot, F. Bodin, F. Cappello, A. Choudhary, B. de Supinski, E. Deelman, J. Dongarra, A. Dubey, G. Fox, H. Fu, S. Girona, W. Gropp, M. Heroux, Y. Ishikawa, K. Keahey, D. Keyes, W. Kramer, J.-F. Lavignon, Y. Lu, S. Matsuoka, B. Mohr, T. Moore, D. Reed, S. Requena, J. Saltz, T. Schulthess, R. Stevens, M. Swany, A. Szalay, W. Tang, G. Varoquaux, J.-P. Vilotte, R. Wisniewski, Z. Xu, and I. Zacharov. Big Data and Extreme-Scale Computing: Pathways to Convergence. Technical report, EXDCI Project of EU-H2020 Program and University of Tennessee, 01 2018.

References II

http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/bdec2017pathways.pdf.

[3] Jian Huang, Karsten Schwan, and Moinuddin K. Qureshi. Nvram-aware logging in transaction systems. *Proc. VLDB Endow.*, 8(4):389–400, December 2014.

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

[5] Jonas Bonér. Latency Numbers Every Programmer Should Know. https://gist.github.com/jboner/2841832.

References III

[6] Pierre Matri, Yevhen Alforov, Alvaro Brandon, Michael Kuhn, Philip H. Carns, and Thomas Ludwig. Could blobs fuel storage-based convergence between HPC and big data? In *2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017*, pages 81–86, 2017.

[7] Pierre Matri, Alexandru Costan, Gabriel Antoniu, Jesús Montes, and María S. Pérez. Týr: blob storage meets built-in transactions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*, pages 573–584, 2016.

[8] OrangeFS Development Team. OrangeFS Documentation. http://docs.orangefs.com/.

[9] Seagate. Desktop HDD. http://www.seagate.com/www-content/datasheets/pdfs/desktop-hdd-8tbDS1770-9-1603DE-de_DE.pdf.

References IV

[10] Veera Deenadhayalan. General Parallel File System (GPFS) Native RAID. https://www.usenix.org/legacy/events/lisa11/tech/slides/deenadhayalan.pdf.

[11] Werner Fischer and Georg Schönberger. Linux Storage Stack Diagramm. https://www.thomas-krenn.com/de/wiki/Linux_Storage_Stack_Diagramm.

[12] Wikipedia. Hard disk drive. http://en.wikipedia.org/wiki/Hard_disk_drive.