# Storage and I/O

## Hardware Architecture of HPC Systems

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

2020-06-10

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

# About Us: Scientific Computing (Wissenschaftliches Rechnen)

- High Performance Computing
- Storage and Parallel I/O
- Data Reduction Techniques

- Middleware Optimization
- Alternative I/O Interfaces
- Cost and Energy Efficiency

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

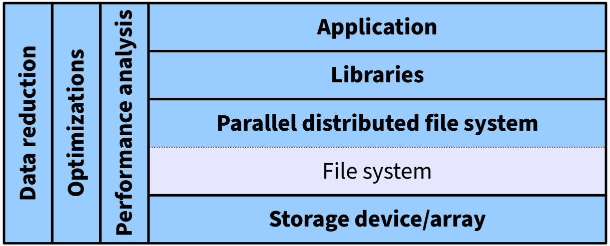
# Computation and I/O

- Parallel applications run on multiple nodes
  - Communication via MPI
- Computation is only one part of applications
  - Input data has to be read
  - Output data has to be written
  - Example: checkpoints
- Processors require data fast
  - Caches should be used optimally
  - Additional latency due to I/O and network

| Level | Latency |
|---:|---:|
| L1 cache | $\approx$ 1 ns |
| L2 cache | $\approx$ 5 ns |
| L3 cache | $\approx$ 10 ns |
| RAM | $\approx$ 100 ns |
| InfiniBand | $\approx$ 500 ns |
| Ethernet | $\approx$ 100,000 ns |
| SSD | $\approx$ 100,000 ns |
| HDD | $\approx$ 10,000,000 ns |

**Table 1:** Latencies [4, 3]

# Computation and I/O…



Data reduction

Optimizations

Performance analysis

Application

Libraries

Parallel distributed file system

File system

Storage device/array

- I/O is often responsible for performance problems
    - High latency causes idle processors
    - I/O is often still serial, limiting throughput
- I/O stack is layered
    - Many different components are involved in accessing data
    - One 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: 100× every 10 years
  - Throughput: 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 1:** HDD development [9]

# Solid-State Drives

- Benefits
  - Read throughput: factor of 15
  - Write throughput: factor of 10
  - Latency: factor of 100
  - Energy consumption: factor of 1–10
- Drawbacks
  - Price: factor of 10
  - Write cycles: 10,000–100,000
  - Complexity
    - Different optimal access sizes for reads and writes
    - Address translation, thermal issues etc.

# RAID

- Storage arrays for higher capacity, throughput and reliability
  - Proposed in 1988 at the University of California, Berkeley
    - Originally: Redundant Array of Inexpensive Disks
    - Today: Redundant Array of Independent Disks

- Capacity
  - Storage array can be addressed like a single, large device
- Throughput
  - All storage devices can contribute to the overall throughput
- Reliability
  - Data can be stored redundantly to survive hardware failures
  - Devices usually have same age, fabrication defects within same batch

# RAID…

- Five different variants initially
  - RAID 1: mirroring
  - RAID 2/3: bit/byte striping
  - RAID 4: block striping
  - RAID 5: block striping with distributed parity
- New variants have been added
  - RAID 0: striping
  - RAID 6: block striping with double parity

# RAID 1

- Improved reliability via mirroring
- Advantages
  - One device can fail without losing data
  - Read performance can be improved
- Disadvantages
  - Capacity requirements and costs are doubled
  - Write performance equals that of a single device

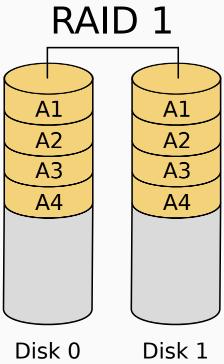


**Figure 2:** RAID 1: Mirroring [10]

# RAID 5

- Improved reliability via parity
    - Typically simple XOR
- Advantages
    - Performance can be improved
    - Requests can be processed in parallel
    - Load is distributed across all devices

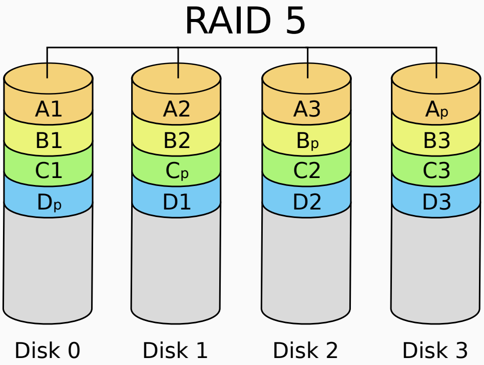


**Figure 3:** RAID 5: Block striping with distributed parity [10]

# RAID 5…

- Data can be reconstructed easily due to XOR
    - $?_A = A1 \oplus A2 \oplus A_p$, $?_B = B1 \oplus B2 \oplus B3, \dots$
- Problems
    - Read errors on other devices
    - Duration (30 min in 2004, 17–18 h in 2020 for HDDs)
    - New approaches like declustered RAID

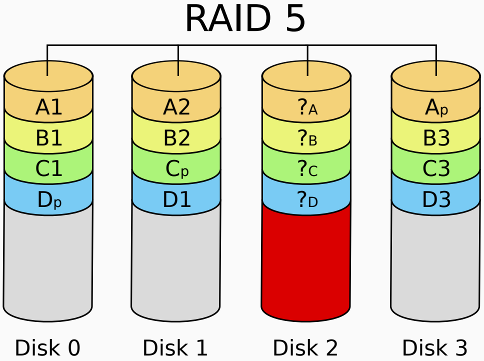


**Figure 4:** RAID 5: Reconstruction [10]

# Performance Assessment

- Different performance criteria
  - Data throughput (photo/video editing, numerical applications)
  - Request throughput (databases, metadata management)
- Appropriate hardware
  - Data throughput
    - HDDs: 150–250 MB/s, SSDs: 0.5–3.5 GB/s
  - Request throughput
    - HDDs: 75–100 IOPS (7,200 RPM), SSDs: 90,000–600,000 IOPS
- Appropriate configuration
  - Small blocks for data, large blocks for requests
  - Partial block/page accesses can reduce performance

Introduction and Motivation

Storage Devices and Arrays

File Systems

Parallel Distributed File Systems

Libraries

Future Developments

Summary

# Overview

- File systems provide structure
  - Files and directories are the most common file system objects
  - Nesting directories results in hierarchical organization
    - Other approaches: tagging
- Management of data and metadata
  - Block allocation is important for performance
  - Access permissions, timestamps etc.
- File systems use underlying storage devices or arrays

# 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
  - Structure the namespace

# I/O Interfaces

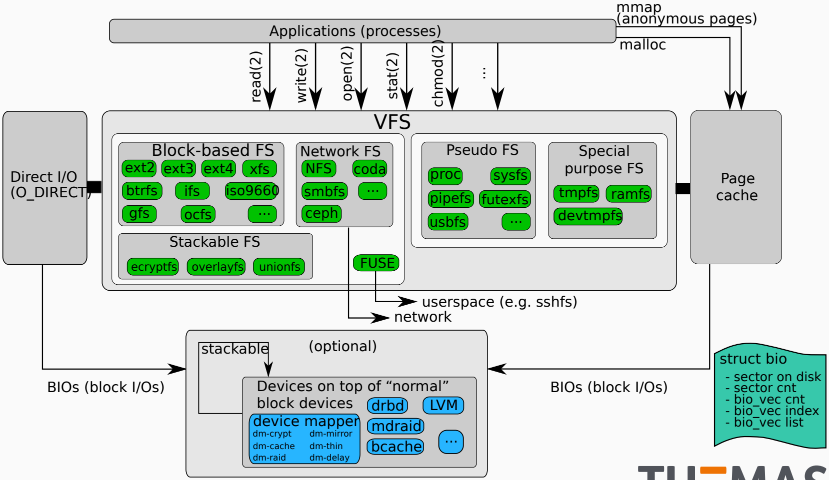
- Requests are realized through I/O interfaces
  - Forwarded to the file system
- Different abstraction levels
  - Low-level functionality: POSIX etc.
  - High-level functionality: NetCDF etc.
- Initial access via path
  - Afterwards access via file descriptor (few exceptions)
- Functions are located in `libc`
  - Library executes system calls

```
fd = open("/path/to/file", ...);
nb = write(fd, data,
           sizeof(data));
rv = close(fd);
rv = unlink("/path/to/file");
```

# Virtual File System (Switch)

- Central file system component in the kernel
  - Sets file system structure and interface
- Forwards applications' requests based on path
- Enables supporting multiple different file systems
  - Applications are still portable due to POSIX
- POSIX: standardized interface for all file systems
  - Syntax defines available operations and their parameters
    - open, close, creat, read, write, lseek, chmod, chown, stat etc.
  - Semantics defines operations' behavior
    - `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)… [8]



The Linux Storage Stack Diagram
http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Created by Werner Fischer and Georg Schönberger

# 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 [6]
  - 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

Introduction and Motivation

Storage Devices and Arrays

File Systems

## Parallel Distributed File Systems

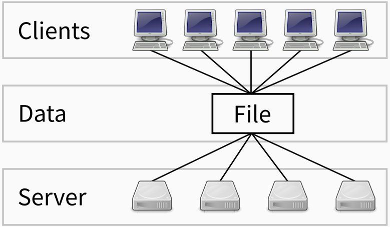
Libraries

Future Developments

Summary

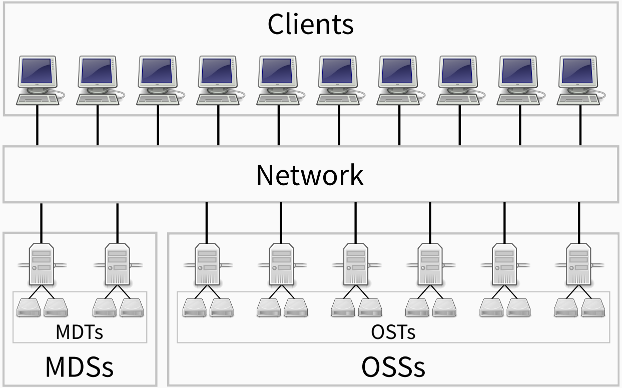
# Overview

- 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

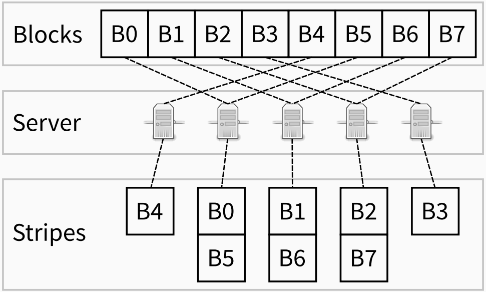
- Access via I/O interface
  - Typically standardized, frequently POSIX
- Interface consists of syntax and semantics
  - Syntax defines operations, semantics defines behavior
- Data and metadata servers
  - Different access patterns



# Semantics

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

# Data Distribution



- 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
    - Allows data and load to be distributed evenly

# Performance

- 2009: Blizzard (DKRZ, GPFS)
  - Computation: 158 TFLOPS
  - Capacity: 7 PB
  - Throughput: 30 GB/s
- 2012: Titan (ORNL, Lustre)
  - Computation: 17.6 PFLOPS
  - Capacity: 40 PB
  - Throughput: 1.4 TB/s
- 2015: Mistral (DKRZ, Lustre)
  - Computation: 3.6 PFLOPS
  - Capacity: 60 PB
  - Throughput: 450 GB/s (5.9 GB/s per node)
  - IOPS: 400,000 operations/s
- 2019: Summit (ORNL, Spectrum Scale)
  - Computation: 148.6 PFLOPS
  - Capacity: 250 PB
  - Throughput: 2.5 TB/s

Introduction and Motivation

Storage Devices and Arrays

File Systems

Parallel Distributed File Systems

Libraries

Future Developments

Summary

# Overview

- Low-level interfaces can be used for parallel I/O
    - They are typically not very convenient for developers
- Additional 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)

# 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. Classic format with full 64 bit support (CDF-5)
  4. NetCDF-4 format
- Data formats are open standards
  - CDF-1 and CDF-2 are international standards of the Open Geospatial Consortium

# NetCDF…

- NetCDF supports groups and variables
  - Groups contain variables, variables contain data
  - Attributes can be attached to variables
- Supports multi-dimensional arrays
  - `char`, `byte`, `short`, `int`, `float` and `double`
  - NetCDF-4: `ubyte`, `ushort`, `uint`, `int64`, `uint64` and `string`
- Dimensions can be sized arbitrarily
  - Only one unlimited dimension with CDF-1, CDF-2 and CDF-5
  - Multiple unlimited dimensions with NetCDF-4

# NetCDF...

1. Create file with `nc_create("file.nc", ..., &ncid)`
    - Parallel access with `nc_create_par`

# NetCDF…

1. Create file with `nc_create("file.nc", ..., &ncid)`
    - Parallel access with `nc_create_par`
2. Define dimension with `nc_def_dim(ncid, "dim", ..., &dimid)`

# NetCDF…

1. Create file with `nc_create("file.nc", ..., &ncid)`
    - Parallel access with `nc_create_par`
2. Define dimension with `nc_def_dim(ncid, "dim", ..., &dimid)`
3. Define group with `nc_def_grp(ncid, "group", &grpid)`

# NetCDF…

1. Create file with `nc_create("file.nc", ..., &ncid)`
    - Parallel access with `nc_create_par`
2. Define dimension with `nc_def_dim(ncid, "dim", ..., &dimid)`
3. Define group with `nc_def_grp(ncid, "group", &grpid)`
4. Define variable with `nc_def_var(grpid, "data", ..., &varid)`

# NetCDF. . .

1. Create file with `nc_create("file.nc", ..., &ncid)`
    - Parallel access with `nc_create_par`
2. Define dimension with `nc_def_dim(ncid, "dim", ..., &dimid)`
3. Define group with `nc_def_grp(ncid, "group", &grpid)`
4. Define variable with `nc_def_var(grpid, "data", ..., &varid)`
5. Write attribute with `nc_put_att_*(grpid, varid, "attr", ...)`

# NetCDF…

1. Create file with `nc_create("file.nc", ..., &ncid)`
    - Parallel access with `nc_create_par`
2. Define dimension with `nc_def_dim(ncid, "dim", ..., &dimid)`
3. Define group with `nc_def_grp(ncid, "group", &grpid)`
4. Define variable with `nc_def_var(grpid, "data", ..., &varid)`
5. Write attribute with `nc_put_att_*(grpid, varid, "attr", ...)`
6. Leave define mode with `nc_enddef(ncid)`
    - Performed implicitly with NetCDF-4 format
    - Compression, endianness, fill values etc. can only be set on first definition

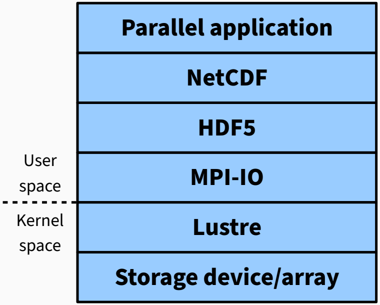
# NetCDF...

1. Create file with `nc_create("file.nc", ..., &ncid)`
   - Parallel access with `nc_create_par`
2. Define dimension with `nc_def_dim(ncid, "dim", ..., &dimid)`
3. Define group with `nc_def_grp(ncid, "group", &grpid)`
4. Define variable with `nc_def_var(grpid, "data", ..., &varid)`
5. Write attribute with `nc_put_att_*(grpid, varid, "attr", ...)`
6. Leave define mode with `nc_enddef(ncid)`
   - Performed implicitly with NetCDF-4 format
   - Compression, endianness, fill values etc. can only be set on first definition
7. Write variable with `nc_put_var_*(grpid, varid, ...)`

## NetCDF…

1. Create file with `nc_create("file.nc", ..., &ncid)`
    - Parallel access with `nc_create_par`
2. Define dimension with `nc_def_dim(ncid, "dim", ..., &dimid)`
3. Define group with `nc_def_grp(ncid, "group", &grpid)`
4. Define variable with `nc_def_var(grpid, "data", ..., &varid)`
5. Write attribute with `nc_put_att_*(grpid, varid, "attr", ...)`
6. Leave define mode with `nc_enddef(ncid)`
    - Performed implicitly with NetCDF-4 format
    - Compression, endianness, fill values etc. can only be set on first definition
7. Write variable with `nc_put_var_*(grpid, varid, ...)`
8. Close file with `nc_close(ncid)`

# 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

# Storage Hierarchy

- Current state
  - L1, L2, L3 cache, RAM, SSD, HDD, tape
- Latency gap from RAM to SSD
  - Huge performance loss if data is not in RAM
- Performance gap is worse on supercomputers
  - RAM is node-local, data is in parallel distributed file system

| 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 |

**Table 2:** Latencies [4, 3]

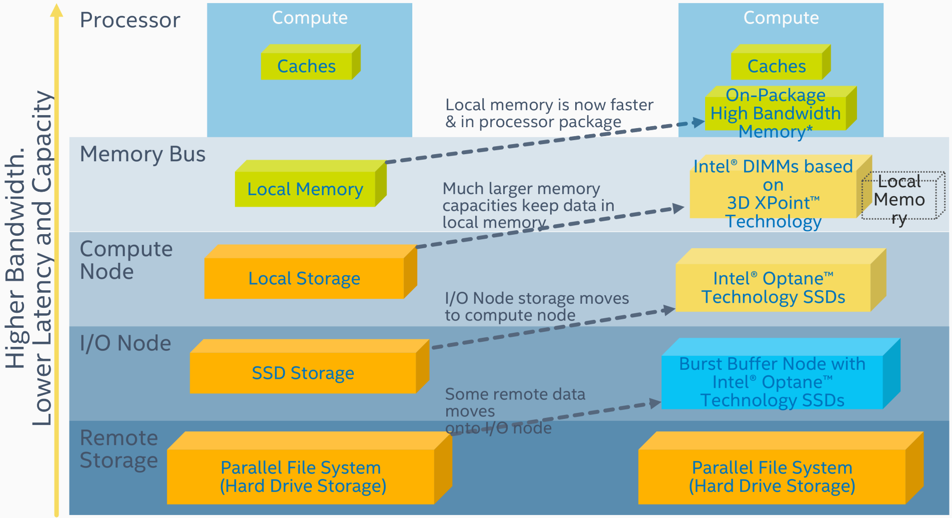
# Storage Hierarchy

- Current state
  - L1, L2, L3 cache, RAM, SSD, HDD, tape
- Latency gap from RAM to SSD
  - Huge performance loss if data is not in RAM
- Performance gap is worse on supercomputers
  - RAM is node-local, data is in parallel distributed file system
- New technologies to close gap
  - Non-volatile RAM (NVRAM), NVM Express (NVMe) etc.

| **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 |

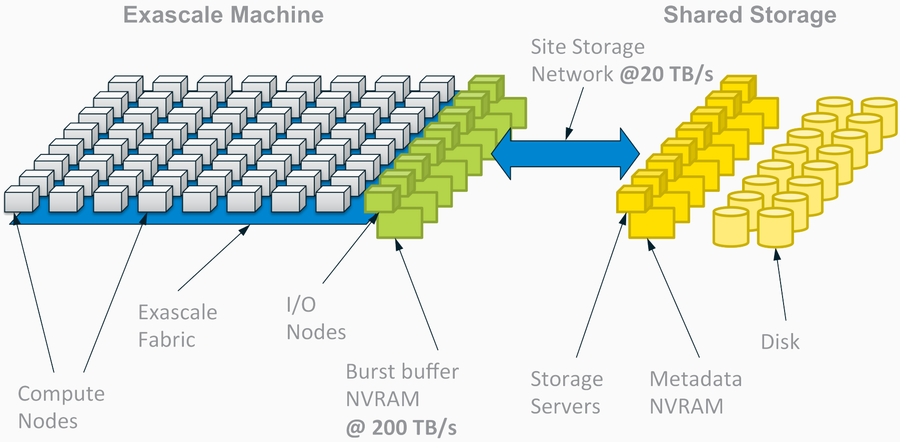
**Table 2:** Latencies [4, 3]

# Storage Hierarchy… [1]



# Storage Hierarchy. . . [2]

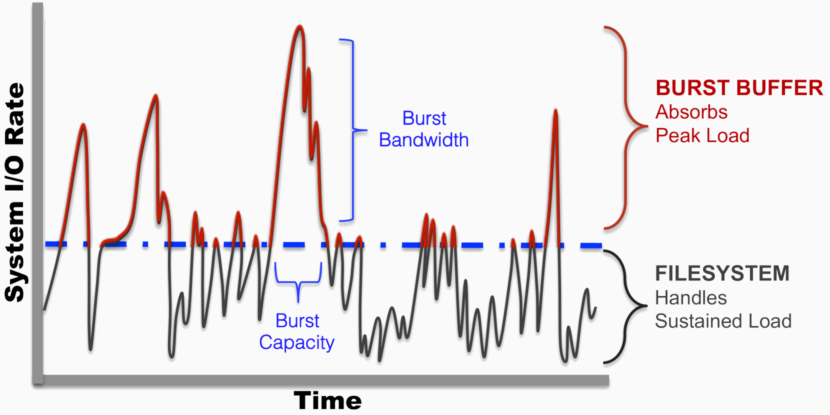
- I/O nodes with burst buffers close to compute nodes
- Slower storage network to file system servers



# Burst Buffers [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

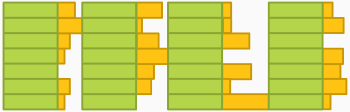


# DAOS

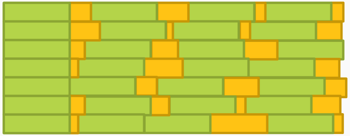
- New holistic approach for I/O
  - Distributed Application Object Storage (DAOS)
- Supports multiple storage models
  - Arrays and records are base objects
  - Objects contain arrays and records (key-array)
  - Containers consist of objects, storage pools consist of containers
- Support for versioning
  - Operations are executed in transactions
  - Transactions are persisted as epochs
- Make use of modern storage technologies

# DAOS… [2]

- I/O is typically performed synchronously
  - Applications have to wait for slowest process, variations are normal
  - File is only consistent after all processes have finished writing



- I/O should be completely asynchronous
  - Eliminates waiting times, makes better use of resources
  - Difficult to define consistency, transactions and snapshots can be used



Introduction and Motivation

Storage Devices and Arrays

File Systems

Parallel Distributed File Systems

Libraries

Future Developments

Summary

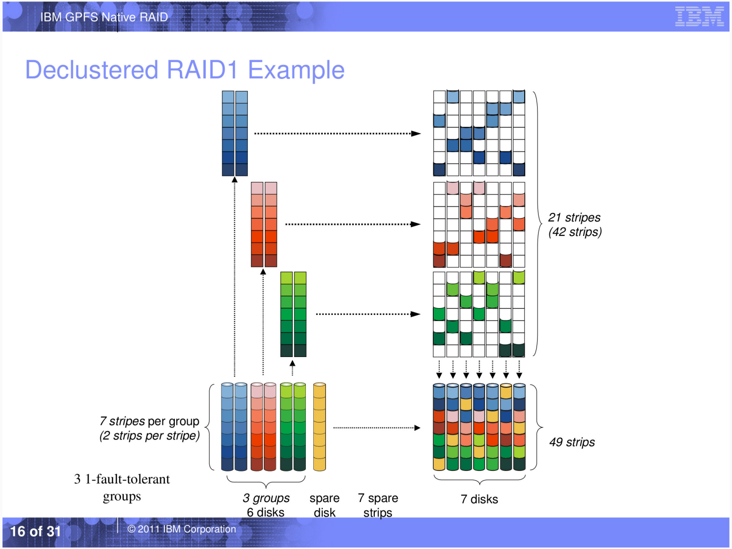
# Summary

- 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
- New technologies will make the I/O stack more complex
    - Future systems will offer novel I/O approaches

# Backup

# References

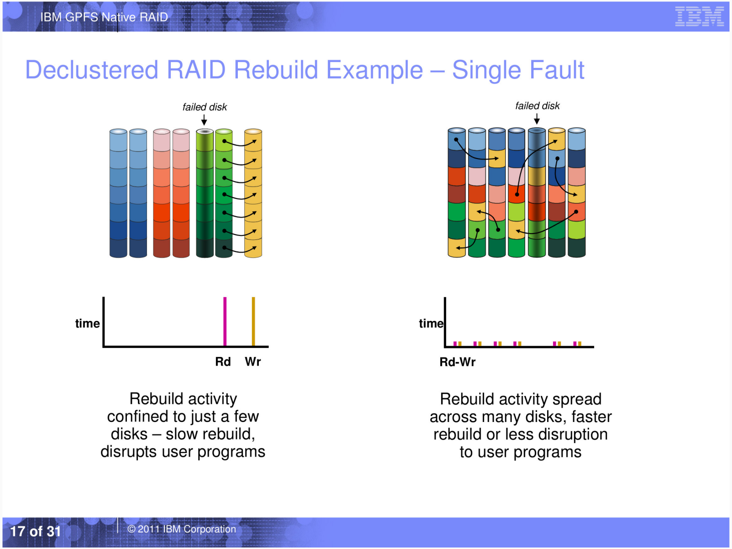
# Declustered RAID [7]



IBM GPFS Native RAID

## Declustered RAID1 Example

# Declustered RAID… [7]



## 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

# 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

# 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...

```xml
<adios-config host-language="C">
  <adios-group name="checkpoint">
    <var name="rows" type="integer"/>
    <var name="columns" type="integer"/>
    <var name="matrix" type="double" dimensions="rows,columns"/>
  </adios-group>
  <method group="checkpoint" method="MPI"/>
  <buffer size-MB="100" allocate-time="now"/>
</adios-config>
```

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

Backup

References

# References

[1] Brent Gorda. **HPC Storage Futures – A 5-Year Outlook.** `http://lustre.ornl.gov/ecosystem-2016/documents/keynotes/Gorda-Intel-keynote.pdf`.

[2] Brent Gorda. **HPC Technologies for Big Data.** `http://www.hpcadvisorycouncil.com/events/2013/Switzerland-Workshop/Presentations/Day_2/3_Intel.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] Jonas Bonér. **Latency Numbers Every Programmer Should Know.** `https://gist.github.com/jboner/2841832`.

[5] Mike Vildibill. **Advanced IO Architectures.** `http://storageconference.us/2015/Presentations/Vildibill.pdf`.

# References ...

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

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

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

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

[10] Wikipedia. **Standard RAID levels.** http://en.wikipedia.org/wiki/Standard_RAID_levels.