

A Survey of Distributed File System Technology

Jakob Blomer
CERN PH/SFT and Stanford University

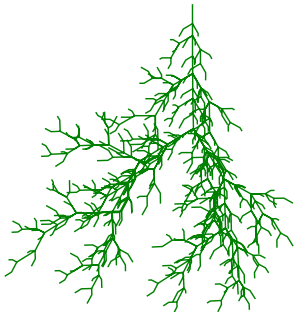
ACAT 2014
Prague

Motivation

- Physics experiments store their data in distributed file systems
- In High Energy Physics
 - Global federation of file systems
 - Hundreds of peta-bytes of data
 - Hundreds of millions of objects

Outline

- ① Usage of distributed file systems
- ② Survey and taxonomy
- ③ Critical areas in distributed file systems for physics applications
- ④ Developments and future challenges



A distributed file system (DFS) provides

- ① persistent storage
- ② of opaque data (files)
- ③ in a hierarchical namespace that is shared among networked nodes

- Files survive the lifetime of processes and nodes
- POSIX-like interface: `open()`, `close()`, `read()`, `write()`, ...
 - Typically transparent to applications
- Data model and interface distinguish a DFS from a distributed (No-)SQL database or a distributed key-value store



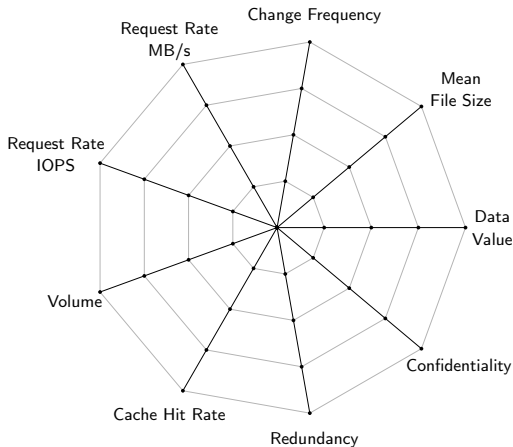
Popular DFSs:

AFS, Ceph, CernVM-FS,
dCache, EOS, FhGFS,
GlusterFS, GPFS, HDFS,
Lustre, MooseFS, NFS,
PanFS, XrootD

A distributed file system (DFS) provides

- ① persistent storage
- ② of opaque data (files)
- ③ in a hierarchical namespace that is shared among networked nodes

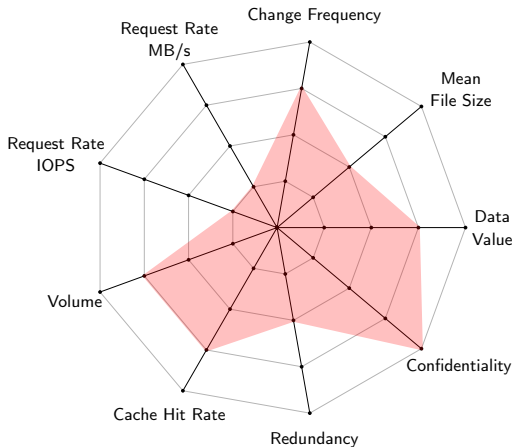
- Files survive the lifetime of processes and nodes
- POSIX-like interface: `open()`, `close()`, `read()`, `write()`, ...
 - Typically transparent to applications
- Data model and interface distinguish a DFS from a distributed (No-)SQL database or a distributed key-value store



[Data are illustrative]

Data Classes

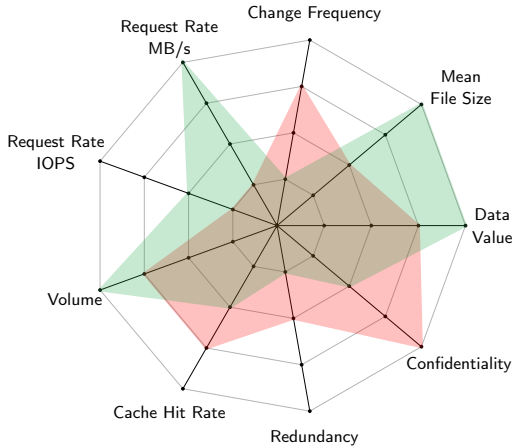
- Home folders
- Physics Data
 - Recorded
 - Simulated
 - Analysis results
- Software binaries
- Scratch area



[Data are illustrative]

Data Classes

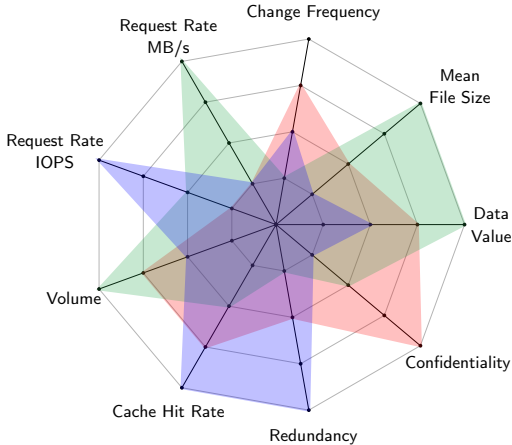
- Home folders —
- Physics Data
 - Recorded
 - Simulated
 - Analysis results
- Software binaries
- Scratch area



[Data are illustrative]

Data Classes

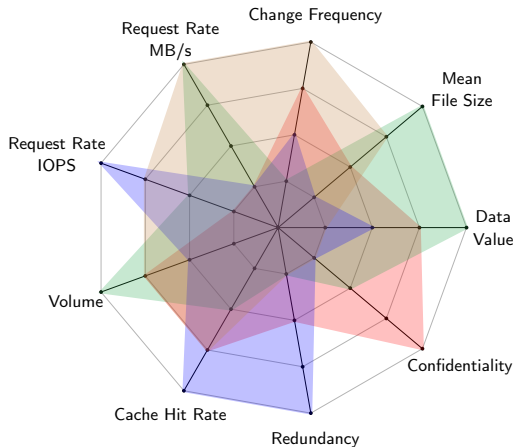
- Home folders —
- Physics Data —
 - Recorded
 - Simulated
 - Analysis results
- Software binaries
- Scratch area



[Data are illustrative]

Data Classes

- Home folders —
- Physics Data —
 - Recorded
 - Simulated
 - Analysis results
- Software binaries —
- Scratch area



[Data are illustrative]

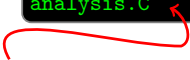
Data Classes

- Home folders —
- Physics Data —
 - Recorded
 - Simulated
 - Analysis results
- Software binaries —
- Scratch area —

Depending on the use case, the dimensions span orders of magnitude
(logarithmic axes)

Distributed File Systems and Use Cases

```
> ls .  
event_sample.root  
analysis.C
```



File system:

“please take special care of this file!”

- It is difficult to perform well under usage characteristics that differ by 4 orders of magnitude
- File system performance is highly susceptible to characteristics of individual applications
- There is no interface to specify quality of service (QoS) for a particular file

Distributed File Systems and Use Cases

```
> ls .  
event_sample.root  
analysis.C
```

File system:
"please take special care of this file!"



```
> ls /  
/home  
/data  
/software  
/scratch
```

Implicit QoS

- It is difficult to perform well under usage characteristics that differ by 4 orders of magnitude
 - File system performance is highly susceptible to characteristics of individual applications
 - There is no interface to specify quality of service (QoS) for a particular file
- We will deal with a number of DFSs for the foreseeable future

- No DFS is fully POSIX compliant
- It must provide **just enough to not break applications**
- Field test necessary

File system operations

essential

`create()`, `unlink()`, `stat()`
`open()`, `close()`,
`read()`, `write()`, `seek()`

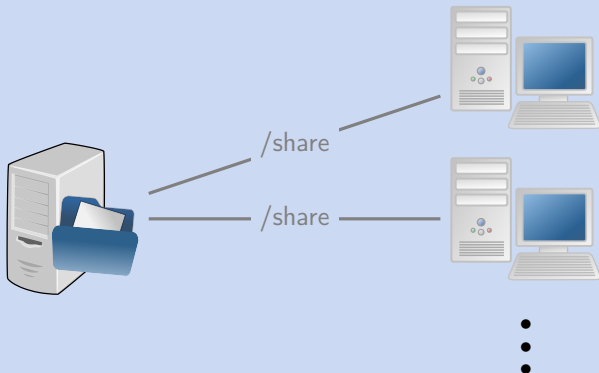
difficult for DFSs

File locks
Atomic `rename()`
Open unlinked files
Hard links

impossible for DFSs

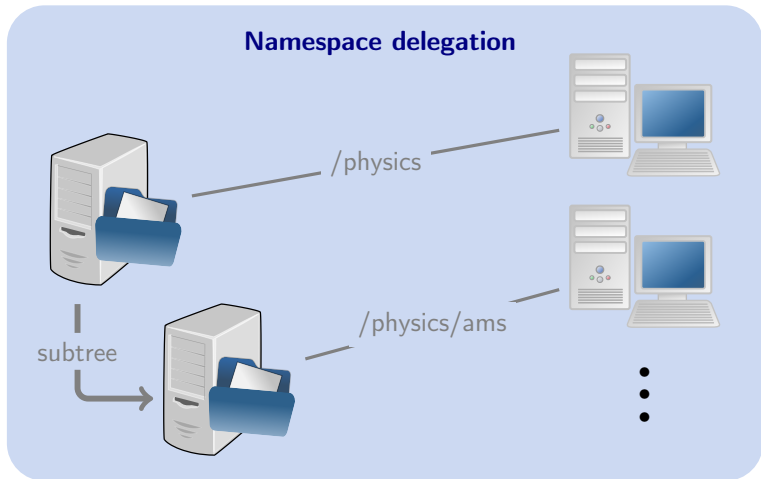
Device files, IPC files

Network shares, client-server



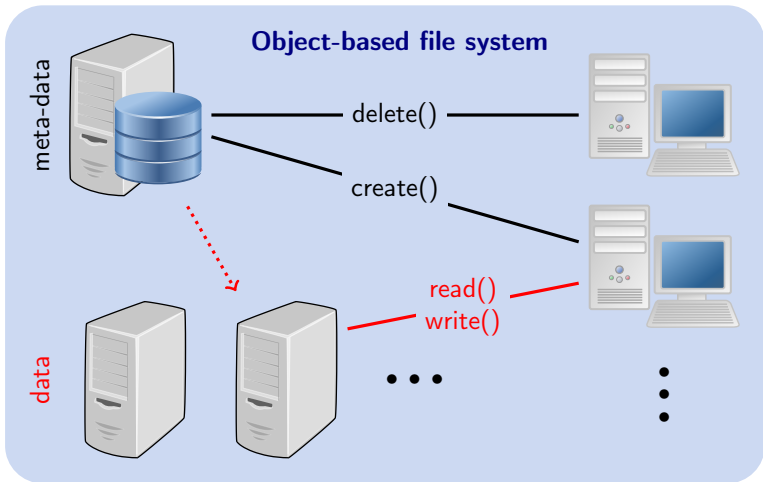
Goals: Simplicity, separate storage from application

Example: NFS 3



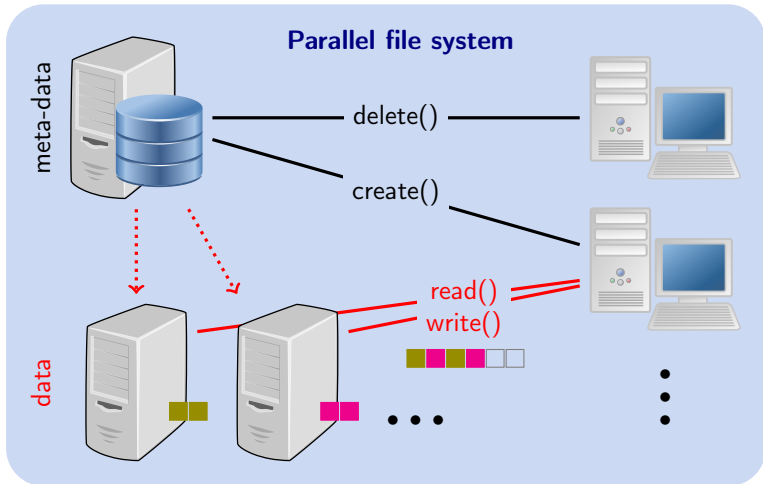
Goals: Scaling network shares, decentral administration

Example: AFS



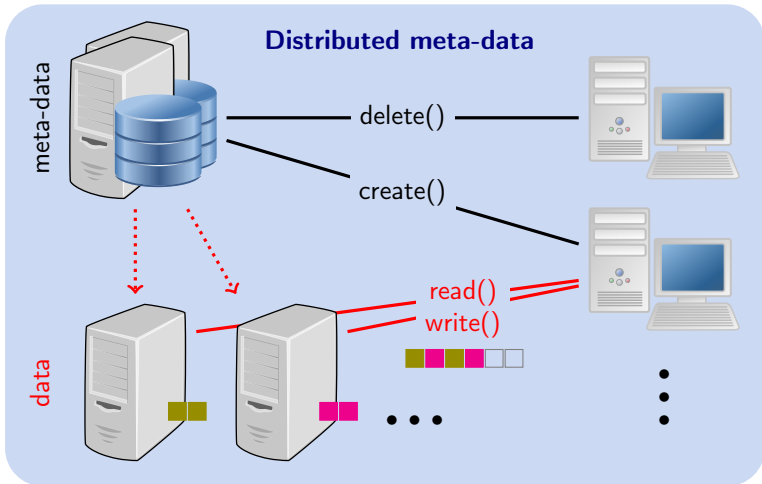
Goals: Separate meta-data from data, incremental scaling

Example: Google File System

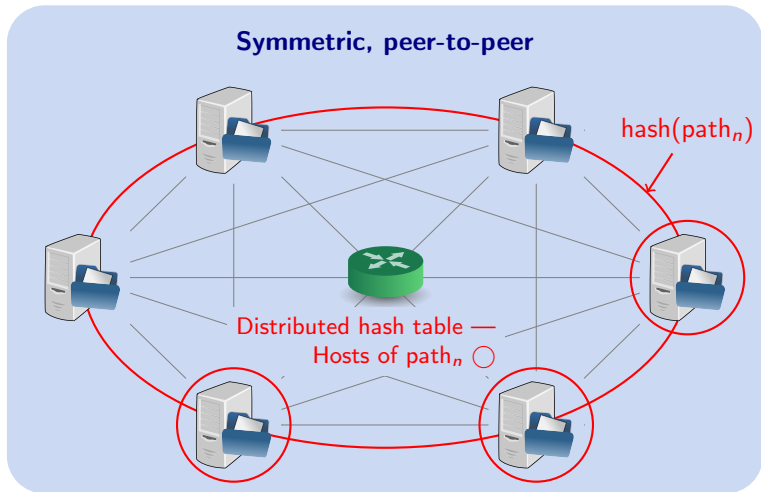


Goals: Maximum throughput, optimized for large files

Example: Lustre



Goals: Avoid single point of failure and meta-data bottleneck
Example: Ceph

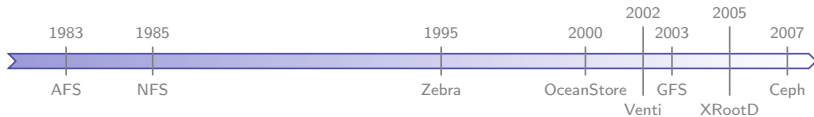


Goals: Conceptual simplicity, inherently scalable
Difficult to deal with node churn, long lookup beyond LAN

Example: GlusterFS

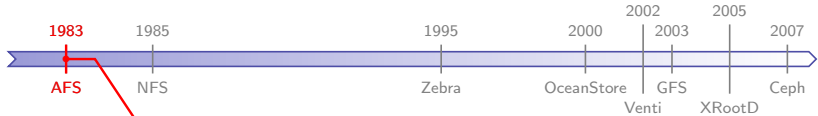
Milestones in Distributed File Systems

Biased towards open-source, production file systems



Milestones in Distributed File Systems

Biased towards open-source, production file systems



Andrew File System Client-server

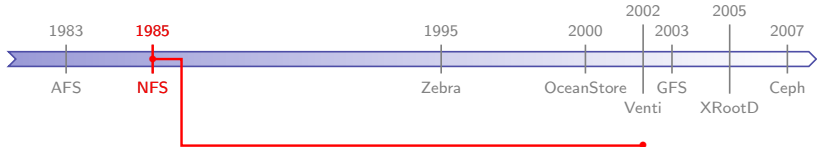
*"AFS was the first safe and efficient distributed computing system, available [...] on campus. It was a clear precursor to the **Dropbox**-like software packages today. [...] [It] allowed students (like Drew Houston and Arash Ferdowsi) access to all their stuff from any connected computer."*

<http://www.wired.com/2011/12/backdrop-dropbox>

- Roaming home folders
- Identity tokens and access control lists (ACLs)
- Decentralized operation ("Cells")

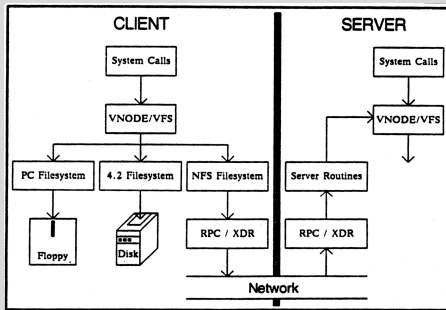
Milestones in Distributed File Systems

Biased towards open-source, production file systems



Network File System Client-server

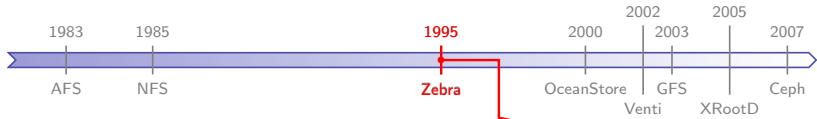
- Focus on portability
- Separation of protocol and implementation
- Stateless servers
 - Fast crash recovery



Sandberg, Goldberg, Kleiman, Walsh, Lyon (1985)

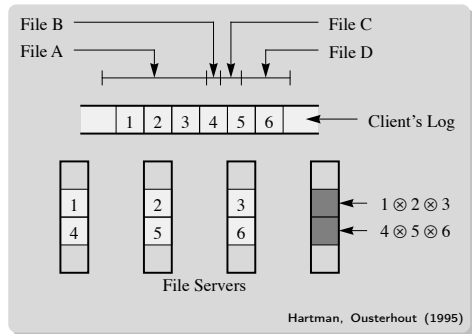
Milestones in Distributed File Systems

Biased towards open-source, production file systems



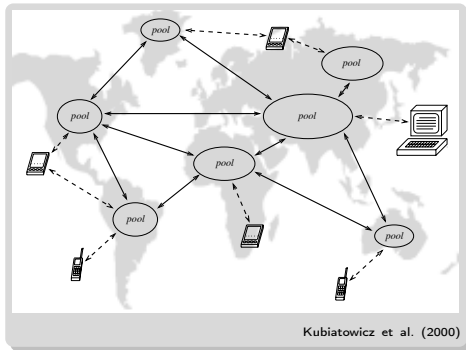
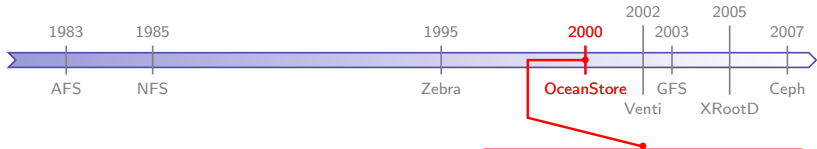
Zebra File System Parallel

- Striping and parity
- Redundant array of inexpensive nodes (RAIN)
- Log-structured data



Milestones in Distributed File Systems

Biased towards open-source, production file systems

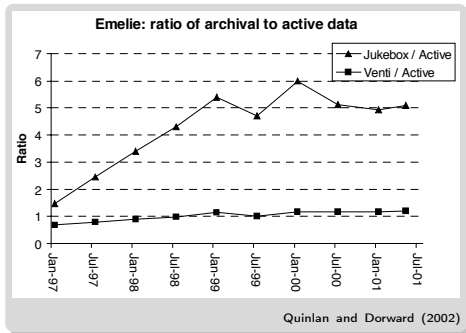
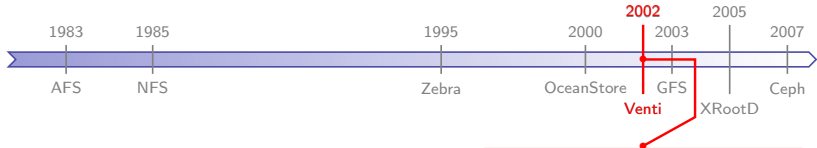


OceanStore Peer-to-peer

- “Global Scale”:
 10^{10} users, 10^{14} files
- Untrusted infrastructure
- Based on peer-to-peer overlay network
- Nomadic data through aggressive caching
- Foundation for today's decentral dropbox replacements

Milestones in Distributed File Systems

Biased towards open-source, production file systems

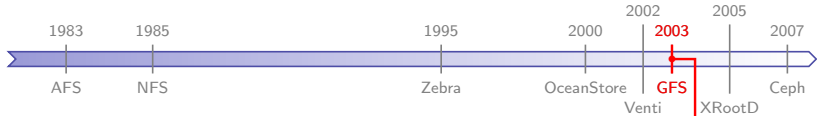


Venti
Archival storage

- De-duplication through content-addressable storage
- Content hashes provide intrinsic file integrity
- Merkle trees verify the file system hierarchy

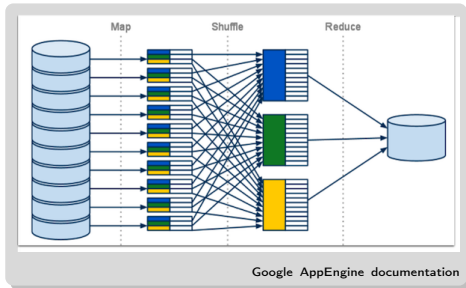
Milestones in Distributed File Systems

Biased towards open-source, production file systems



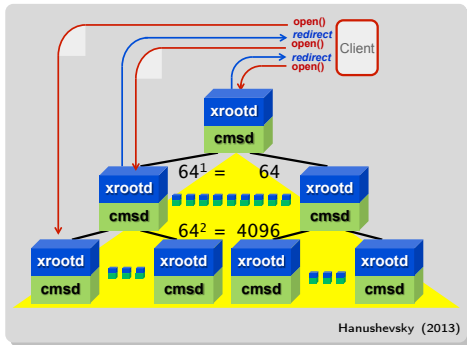
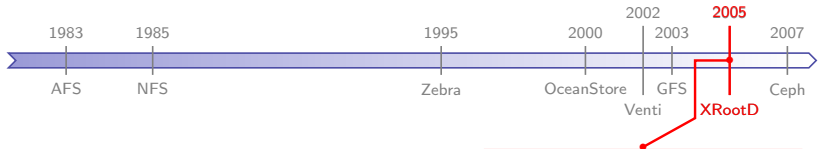
Google File System
Object-based

- Co-designed for map-reduce
- Coalesce storage and compute nodes
- Serialize data access



Milestones in Distributed File Systems

Biased towards open-source, production file systems

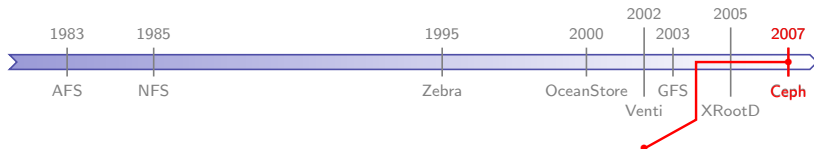


XRootD Namespace delegation

- Global tree of redirectors
- Flexibility through decomposition into pluggable components
- Namespace independent from data access

Milestones in Distributed File Systems

Biased towards open-source, production file systems



Action	Resulting \vec{i}
take(root)	root
select(1,row)	row2
select(3,cabinet)	cab21 cab23 cab24
select(1,disk)	disk2107 disk2313 disk2437
emit	

Weil (2007)

Ceph File System and RADOS
Parallel, distributed meta-data

- Peer-to-peer file system at the cluster scale
- Data placement across failure domains
- Adaptive workload distribution



Critical Areas in DFSs for Physics Applications

Fault-Tolerance

Fundamental problems as the number of components grows

- ① Faults are the norm
- ② Faults are often correlated
- ③ No safe way to distinguish temporary unavailability from faults

Bandwidth Utilization

- Data structures that work throughout the memory hierarchy
- Efficient writing of small files: analysis result merging, meta-data

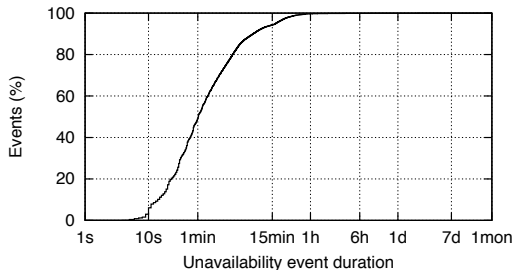
Data Reliability Techniques

- **Replication**: simple and fast but **large storage overhead**
 - Trend from random placement to “**de-correlation**”
- **Erasure codes**: any $n + \varepsilon$ out of $n + k$ blocks reconstruct data
 - Different codes offer different trade-offs between **computational complexity** and storage overhead
- **Checksums**: detect silent data corruption

Engineering Challenges

- Fault detection
- Automatic and fast recovery
- Failure prediction
e. g. based on MTTF and Markov models

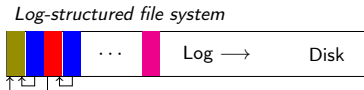
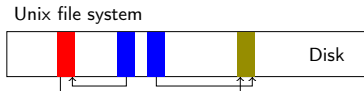
Example from Google data centers



Statistical separation
of temporary and
permanent faults

Ford et al. (2010) [▶ Link](#)

Log-Structured Data



Idea: Store all modifications in a change log

Use full hardware bandwidth for small objects

Used by

- Zebra experimental DFS
- Commercial filers (e. g. NetApp)
- Key-value stores
- File systems for flash memory

Advantages

- Minimal seek, in-place updates
- Fast and robust crash recovery
- Efficient allocation in DRAM, flash, and disks
- **Applicable for merging, meta-data**

Do we get what we need?



Typical physics experiment cluster

- Up to 1 000 standard nodes
- 1 GbE or 10 GbE network, high bisection bandwidth

Goals for a DFS for analysis applications

- At least 90 % available disk **capacity**
- At least 50 % of maximum **aggregated throughput**
- **Fault-tolerant** to a small number of disk/node failures
- **Symmetric** architecture, fully decentralized

Complexity and Decomposition in Distributed File Systems

For a DFS: At least 5 years from inception to widespread adoption

Complexity

- Open source DFSs comprise some 100 kLOC to 750 kLOC
- It takes a community effort to stabilize them
- Once established, it can become prohibitively expensive to move to a different DFS (data lock-in)

Complexity and Decomposition in Distributed File Systems

For a DFS: At least 5 years from inception to widespread adoption

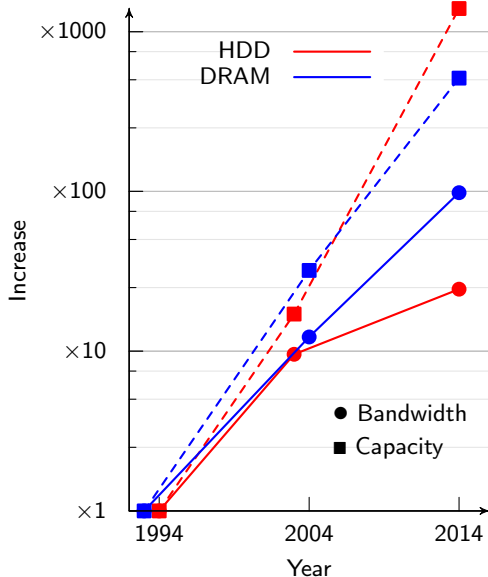
Complexity

- Open source DFSs comprise some 100 kLOC to 750 kLOC
- It takes a community effort to stabilize them
- Once established, it can become prohibitively expensive to move to a different DFS (data lock-in)

Decomposition

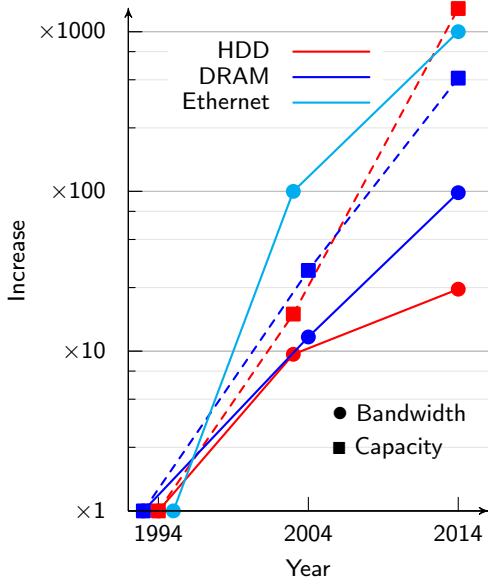
- Good track record of “outsourcing” tasks
e. g. authentication (Kerberos), distributed coordination (ZooKeeper)
- Ongoing: separation of namespace and data access
- Increases the number of standards and interfaces and temporarily increases the effort on the development side
- ⊕ Faster adaption to a changing computing landscape

Distributed File Systems in the Exascale



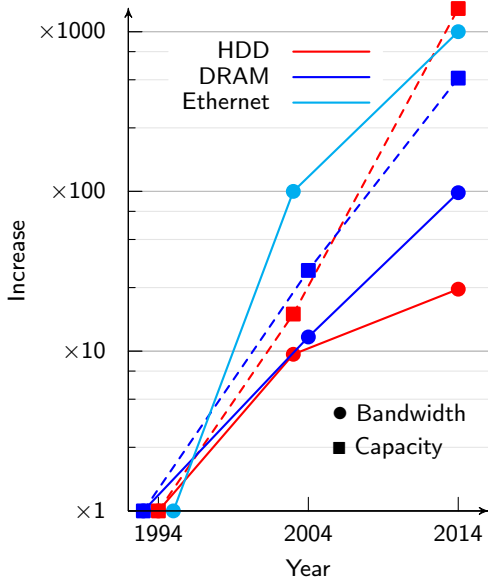
- “Exascale” computing (EB of data, 10^{18} ops/s) envisaged by 2020
- Storage capacity and bandwidth scale at different pace
- It will be more difficult or impossible to constantly “move data in and out”

Distributed File Systems in the Exascale



- “Exascale” computing (EB of data, 10^{18} ops/s) envisaged by 2020
- Storage capacity and bandwidth scale at different pace
- It will be more difficult or impossible to constantly “move data in and out”
- Ethernet bandwidth scaled similarly to capacity

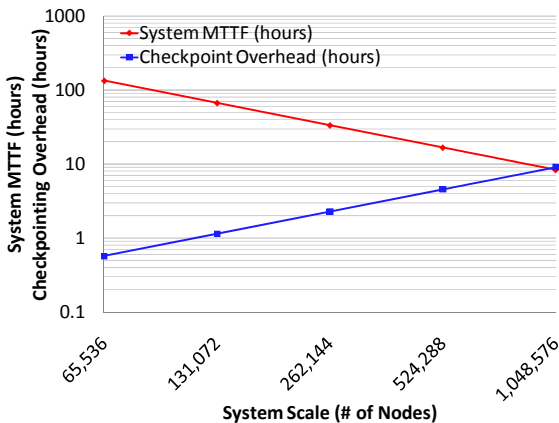
Distributed File Systems in the Exascale



- “Exascale” computing (EB of data, 10^{18} ops/s) envisaged by 2020
- Storage capacity and bandwidth scale at different pace
- It will be more difficult or impossible to constantly “move data in and out”
- Ethernet bandwidth scaled similarly to capacity
- Yielding segregation of storage and computing to a **symmetric DFS** can be part of a solution

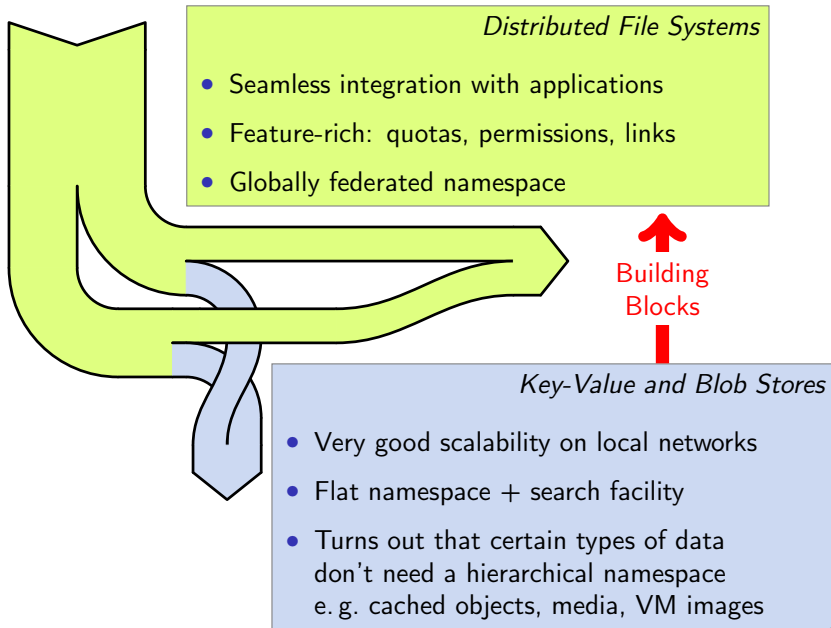
Distributed File Systems in the Exascale

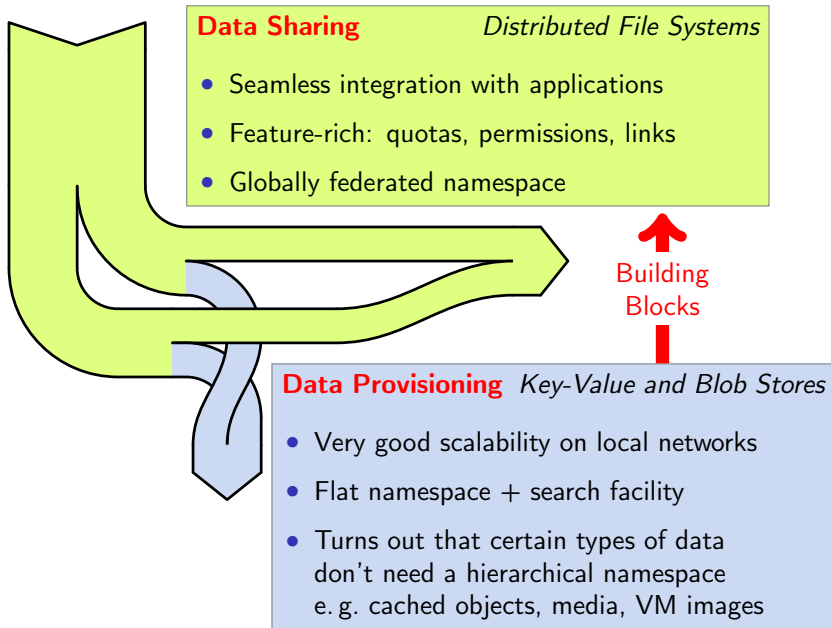
Example from HPC



Raicu, Foster, Beckman (2011) [▶ Link](#)

- “Exascale” computing (EB of data, 10^{18} ops/s) envisaged by 2020
- Storage capacity and bandwidth scale at different pace
- It will be more difficult or impossible to constantly “move data in and out”
- Ethernet bandwidth scaled similarly to capacity
- Yielding segregation of storage and computing to a **symmetric DFS** can be part of a solution





① Distributed file systems stay

- physics data processing applications use file systems
- the hierarchical namespace is a natural way to organize data

① Distributed file systems stay

- physics data processing applications use file systems
- the hierarchical namespace is a natural way to organize data

② Hard disks become data silos

- We need to focus on optimal bandwidth utilization
- Once written, we have to leave data where they are
→ storage and compute nodes coalesce

① Distributed file systems stay

- physics data processing applications use file systems
- the hierarchical namespace is a natural way to organize data

② Hard disks become data silos

- We need to focus on optimal bandwidth utilization
- Once written, we have to leave data where they are
→ storage and compute nodes coalesce

③ Every now and then, a new file system comes along — they all are assembled from the same technology toolbox

- We need solidly engineered building blocks from this toolbox
- We need to validate early with our real application workload



<http://lonelychairsatcern.tumblr.com>

Backup Slides

Source of Hardware Bandwidth and Capacity Numbers

Method and entries marked † from Patterson (2004) [▶ Link](#)

Year	Hard Disk Drives		DRAM		Ethernet Bandwidth
	Capacity	Bandwidth	Capacity	Bandwidth	
1993			16 Mibit/chip†	267 MiB/s†	
1994	4.3 GB†	9 MB/s†			
1995					100 Mbit/s†
2003	73.4 GB†	86 MB/s†			10 Gbit/s†
2004			512 Mibit/chip	3.2 GiB/s	
2014	6 TB	220 MB/s†	8 Gbit/chip	25.6 GiB/s	100 Gbit/s
Increase	×1395	×24	×512	×98	×1000

†http://www.storagereview.com/seagate_enterprise_capacity_6tb_35_sas_hdd_review_v4

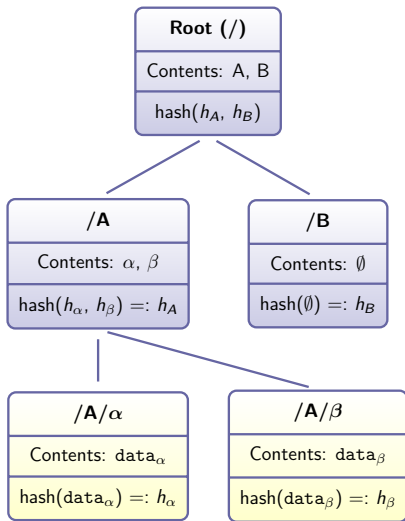
HDD: Seagate ST15150 (1994)†, Seagate 373453 (2004)†, Seagate ST6000NM0034 (2014)

DRAM: Fast Page DRAM (1993)†, DDR2-400 (2004), DDR4-3200 (2014)

Ethernet: Fast Ethernet IEEE 802.3u (1995)†, 10 GbitE IEEE 802.3ae (2003)†, 100 GbitE IEEE 802.3bj (2014)

High-end SSD example: Hitachi FlashMAX III (2014), ≈2 TB, ≈2 GB/s

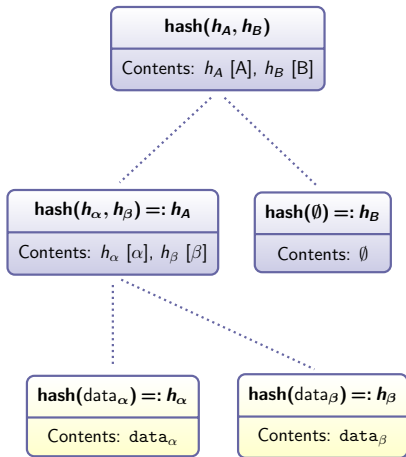
Data Integrity and File System Snapshots



Merkle tree

- Hash tree with cryptographic hash function provides **secure identifier for sub trees**
- It is easy to sign a small hash value (data authenticity)
- Efficient calculation of changes (**fast replication**)
- Bonus: versioning and data de-duplication

Data Integrity and File System Snapshots



Merkle tree

\oplus

Content-addressable storage

- Hash tree with cryptographic hash function provides **secure identifier for sub trees**
- It is easy to sign a small hash value (data authenticity)
- Efficient calculation of changes (**fast replication**)
- Bonus: versioning and data de-duplication
- Full potential together with content-addressable storage
- **Self-verifying data chunks**, trivial to distribute and cache

Survey Articles



Satyanarayanan, M. (1990).
A survey of distributed file systems.
Annual Review of Computer Science, 4(1):73–104.



Guan, P., Kuhl, M., Li, Z., and Liu, X. (2000).
A survey of distributed file systems.
University of California, San Diego.



Agarwal, P. and Li, H. C. (2003).
A survey of secure, fault-tolerant distributed file systems.
URL:
<http://www.cs.utexas.edu/users/browne/cs395f2003/projects/LiAgarwalReport.pdf>.



Thanh, T. D., Mohan, S., Choi, E., Kim, S., and Kim, P. (2008).
A taxonomy and survey on distributed file systems.
In *Proc. int. conf. on Networked Computing and Advanced Information Management (NCM'08)*, pages 144 – 149.



Depardon, B., Séguin, C., and Mahec, G. L. (2013).
Analysis of six distributed file systems.
Technical Report hal-00789086, Université de Picardie Jules Verne.



Donvito, G., Marzulli, G., and Diacono, D. (2014).
Testing of several distributed file-systems (hdfs, ceph and glusterfs) for
supporting the hep experiment analysis.
Journal of Physics: Conference Series, 513.

File Systems



Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B. (1985).
Design and implementation of the sun network filesystem.
In *Proc. of the Summer USENIX conference*, pages 119–130.



Morris, J. H., Satyanarayanan, M., Conner, M. H., Howard, J. H., Rosenthal, D.
S. H., and Smith, F. D. (1986).
Andrew: A distributed personal computing environment.
Communications of the ACM, 29(3):184–201.



Hartman, J. H. and Osterhout, J. K. (1995).
The Zebra striped network file system.
ACM Transactions on Computer Systems, 13(3):274–310.



Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000).

OceanStore: An architecture for global-scale persistent storage.
ACM SIGPLAN Notices, 35(11):190–201.



Quinlan, S. and Dorward, S. (2002).

Venti: a new approach to archival storage.
In *Proc. of the 1st USENIX Conf. on File and Storage Technologies (FAST'02)*, pages 89–102.



Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003).

The Google file system.
ACM SIGOPS Operating Systems Review, 37(5):29–43.



Schwan, P. (2003).

Lustre: Building a file system for 1,000-node clusters.
In *Proc. of the 2003 Linux Symposium*, pages 380–386.



Dorigo, A., Elmer, P., Furano, F., and Hanushevsky, A. (2005).

XROOTD - a highly scalable architecture for data access.
WSEAS Transactions on Computers, 4(4):348–353.



Weil, S. A. (2007).

Ceph: reliable, scalable, and high-performance distributed storage.

PhD thesis, University of California Santa Cruz.