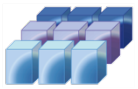
Exascale File Systems

Scalability in ClusterStor's Colibri System

Peter Braam

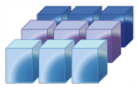
Mar 15, 2010

ClusterStor

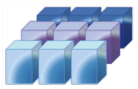


Content

- A few words about Lustre
- System overview
- Management
- Availability
- I/O
- Scalable communications

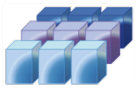


Lustre



Lustre

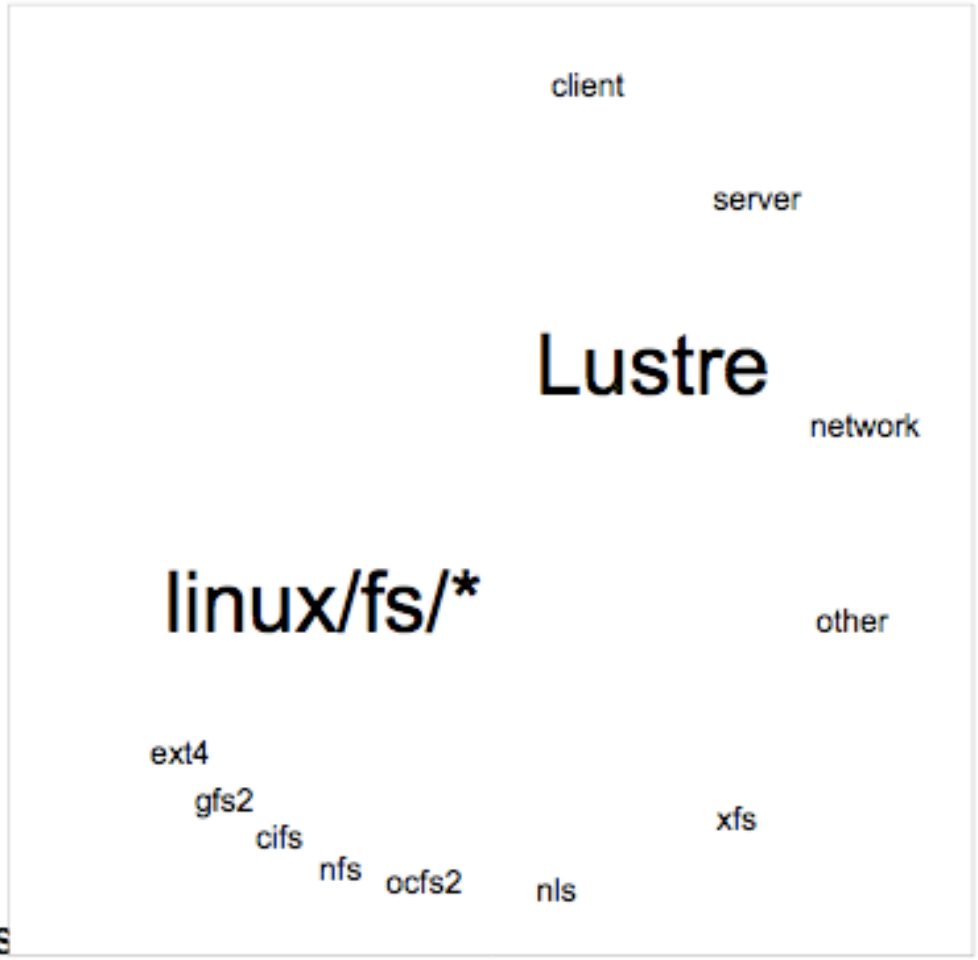
- I started this in 1999, left Sun late 2008
- A few questions
  - How successful is Lustre?
  - Why not evolve Lustre to exascale system?
  - What key differences to expect?



Lustre is doing well: Top 500 June 2010

- 59/100 run Lustre
- 22/100 run GPFS
- 3/100 run Panasas
- 1/100 run CXFS
- 15 undetermined

Engineering
Lines of Code



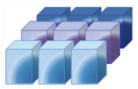
- Lustre – 257 KLOC
- Total of all in-tree linux filesys

Source: Slide 3 of E. Barton presentation at LUG 2010

Some History

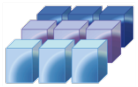
- DOE: paid much, provided requirements, QA
- Version interoperability key to increasing cost
- Lack of focus on QA is biggest regret

	Event	Release	Comments
1999	Lustre Project Started @CMU		
2001	Cluster File Systems founded		
2003		Dec: Lustre 1.0	~$3.8M to develop 1.0
2004	Nov: Lustre on #1 system (BGL)	Apr: Lustre 1.2 Nov: Lustre 1.4	~$5.9M to get to #1
2007	250 paying customers, 25 OEMs	Apr: Lustre 1.6	~$10M for 1.4 to 1.6 upgrade
2009		May: Lustre 1.8	2 more years of R&D for 1.6 to 1.8



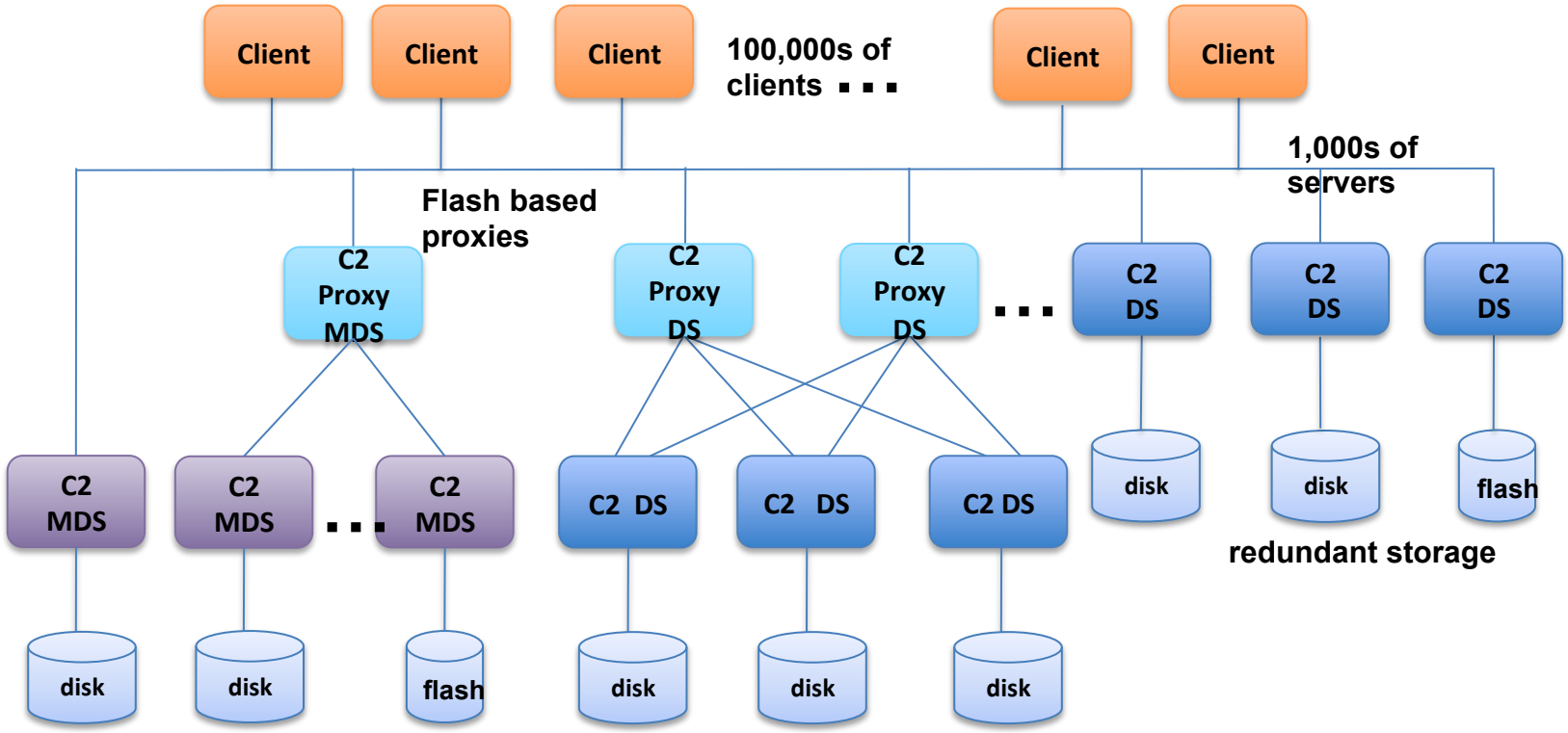
Key differences Colibri - Lustre

- Focus on quality, diagnostics, usability
- Utilize flash storage
- Stop following 40 year old UFS paradigms
  - Use embedded databases
  - Introduce powerful interfaces
- No dependence on RAID storage, failover



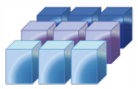
System overview

Colibri (aka C2) deployment



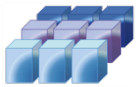
3rd party storage: RAID arrays, JBOD, Internal storage, with or without flash

- Complete file service solution
- Clients have "cluster intelligence"
- Scales enormously (HPCS and beyond)



A few elements to start

- Object store is used for everything
- Metadata is in embedded databases
  - Schema breaks away from UFS (finally)
- PCI flash storage: integral design element
- Architecture should scale to 100 PF range
  - After that I see too many disks again….



Management

Management

- What was learned in the past?
- Observing a cluster is crucial
  - FOL: file operations log
  - ADDB: analysis and diagnostics database
  - A data mining & visualization tool
- Simulations
  - Simple language expressing I/O patterns
  - Simulator that processes the FOL traces
  - Editor for FOL for simulations
  - Server load simulations

ADDB & FOL

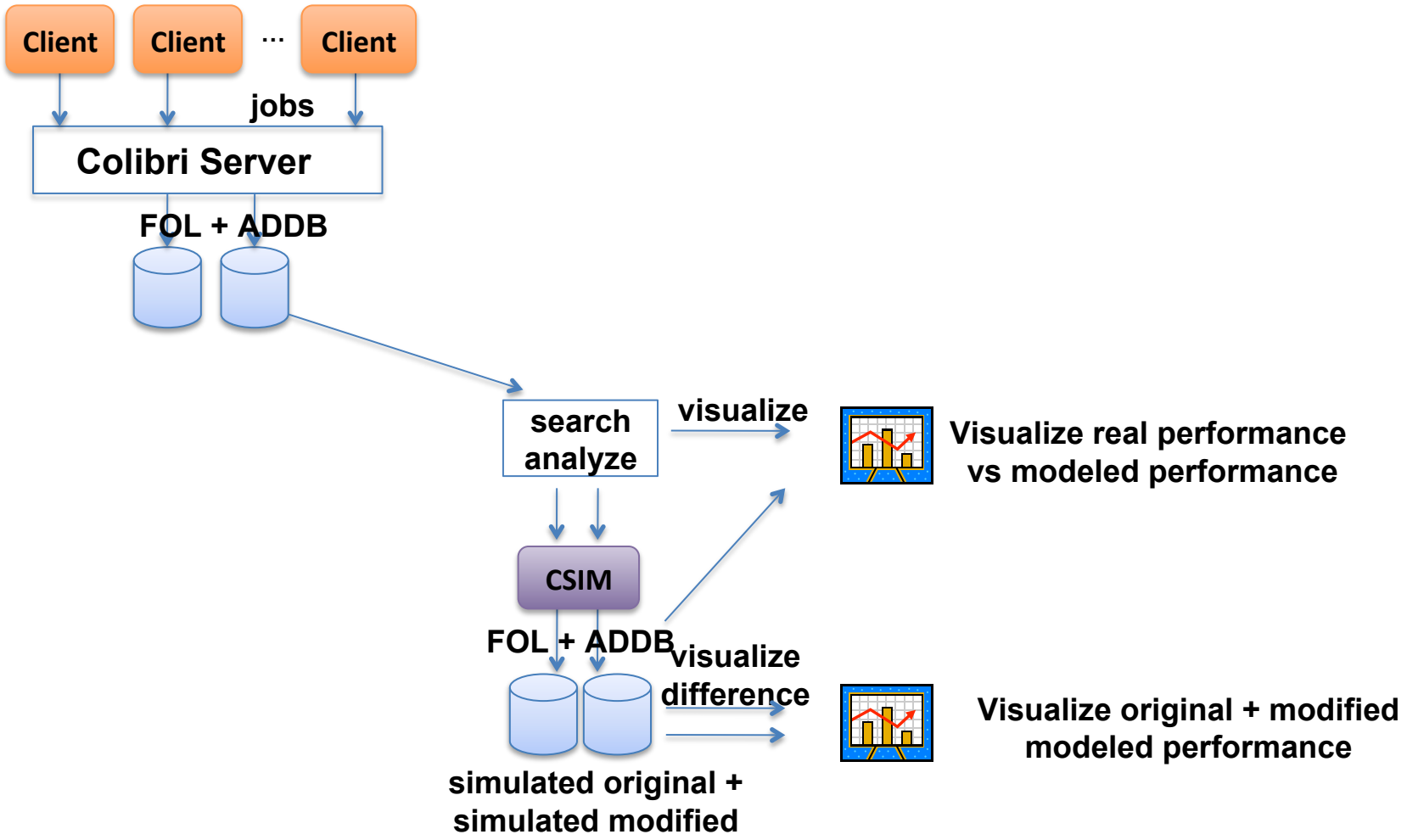
FOL

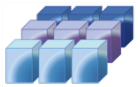
- File Operation Log
- Contains FOP
  - packets
- Transactional
- Complete
- Useful for data mgmt

ADDB

- For each FOL record
  - One ADDB record
- Contains
  - Analysis & diagnostics data
- Eg:
  - Which client did what?
  - How long did it take
  - What network?
  - Cache hits / disk queues

Example - Simulation Environment

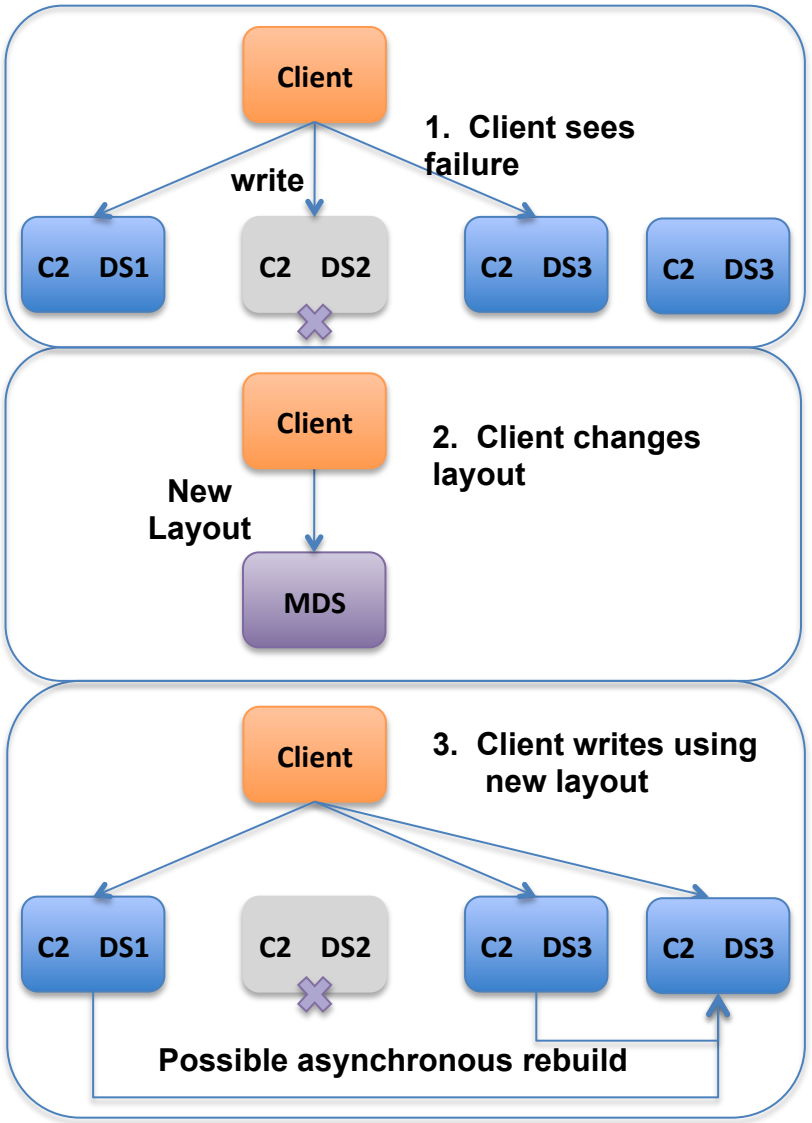


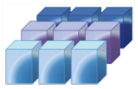


Failures & availability

No failover

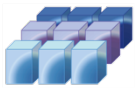
- Failures will be common
  - in very large systems
- Failover
  - Wait for resource
  - Doesn't work well
- Focus on availability
  - No reply (failure, load)
  - Adapt layout
  - Asynchronous cleanup





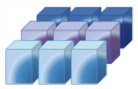
Metadata - Data Layout

- Almost all layouts will be regular
  - Use formulas, do not enumerate objects & devices
  - Use references between metadata tables
    - Avoid multiple copies
- After failures, data layout becomes complex
  - Failures can move 1000s of different extents in a file
  - May clean this up asynchronously
    - FOL knows about it



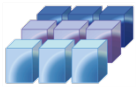
Data placement

- Data layout is central in architecture
  - Redundancy across the servers in network
  - RAID in data servers has little value
- Parity de-clustered in network
  - All drives:
    - Contain some data of objects
    - Contain some spare space
  - 1-5 minute restore of redundancy for failed drive
    - with bandwidth of cluster

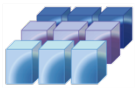


Availability

- System @ full bandwidth 99.97% of the time
  - Disk rebuilds are not the problem
  - Server losses are a problem
- Systems must have network redundancy
  - AKA server network striping
- No purpose for shared storage



Caches and scale



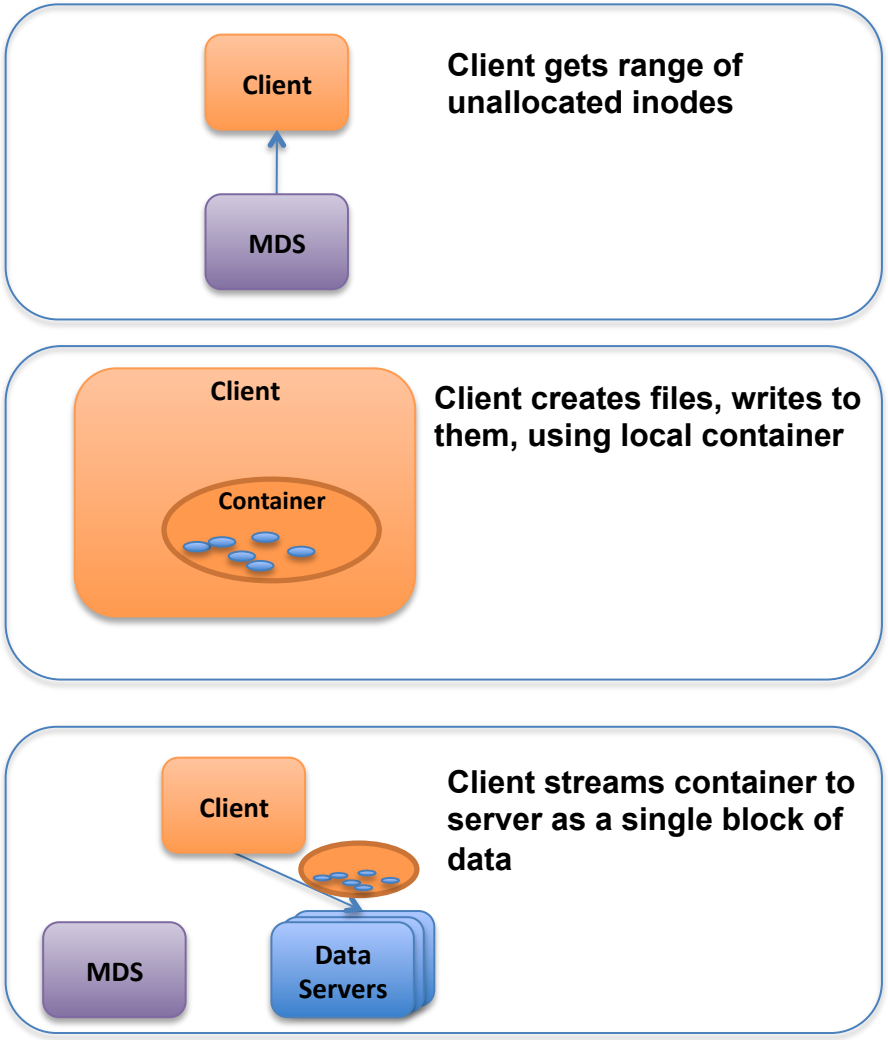
Flash, cache & scale

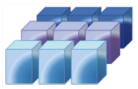
- {Flash,Disk} based servers = {FBS, DBS}
- 2010
  - Bandwidth: FBS $/GB/sec == 0.25 DBS $/GB/sec
  - Capacity: FBS $/GB == 5x DBS $/GB
  - FBS form factor –> PCI flash becomes commodity
- Choose
  - FBS for bandwidth, capacity ~ RAM of cluster
    - FBS bandwidth high enough for e.g. fast checkpoint
  - DBS Bandwidth = 0.2 FBS Bandwidth

Containers – forwarding caches

- Containers are like a logical volume
- Can contain
  - Metadata objects
  - File fragments
  - Files
- New feature – "include" operation
  - a container moves into a larger container
  - "include" does ***not iterate over contents***
- Metadata is more complex
  - Not problematic, we use an embedded DB anyway

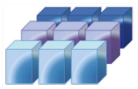
Container streaming





Several uses for containers

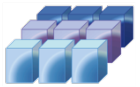
- WB caches in clients
  - Advantage – no iteration over contents
- Networked caches to act as I/O capacitor
  - Get flash for bandwidth,
    - Capacity ~ RAM of entire cluster
  - Get disk for capacity
- Intermediary storage, e.g. I/O forwarding
  - Physically non-contiguous container with small stuff
  - Streams to contiguous container inside a bigger one
- Data management



Oh, you need to read?

- Physics – disks are slow
- Two common cases:
  1. Everyone reads the same
  2. Everyone reads something different

- Case 2 is best handled as follows:
  - Use FOL or otherwise track what files will be needed
  - Pre-stage them in flash
- Case 1 is addressed with scalable comms

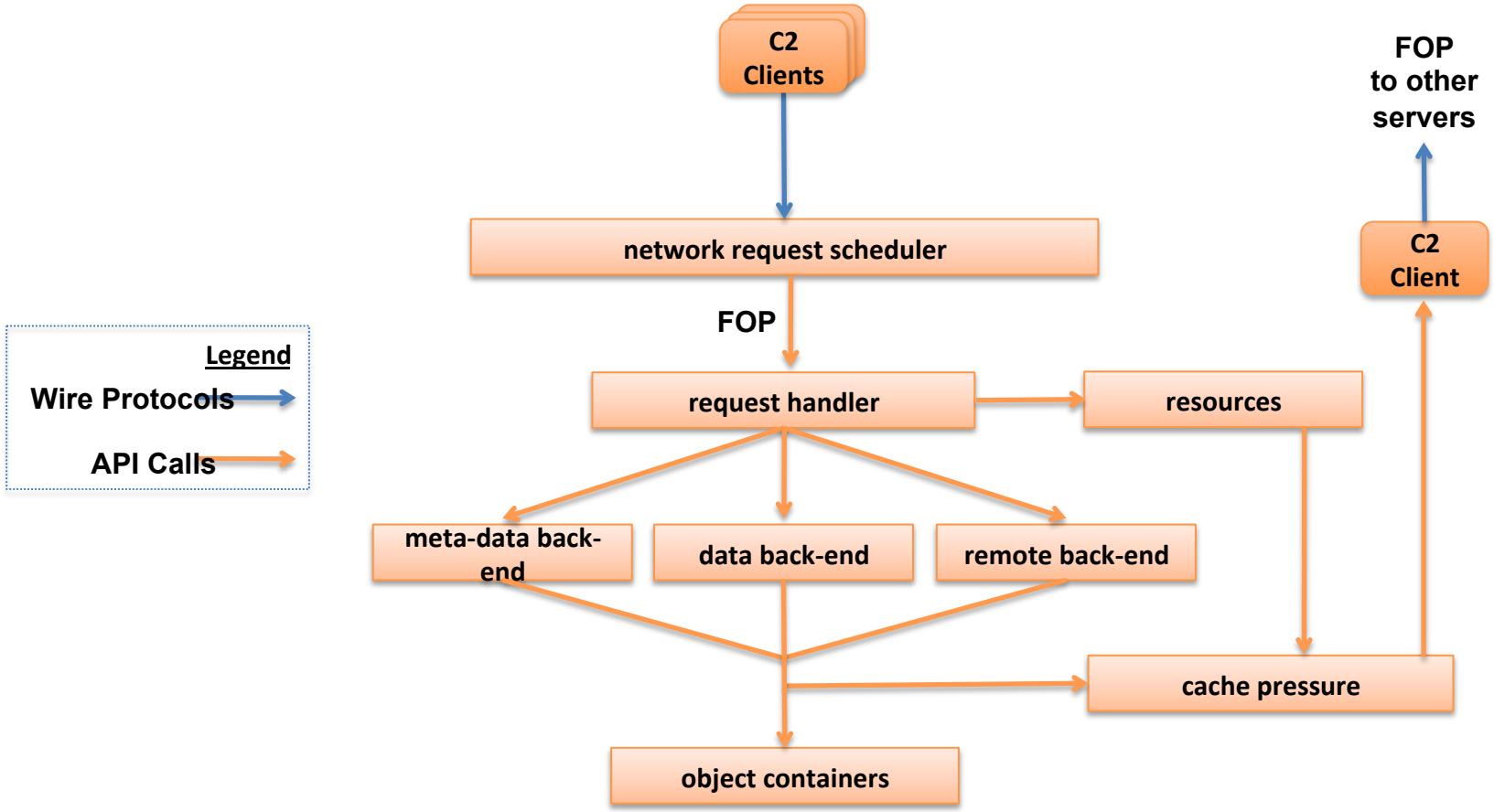


Scalable communications

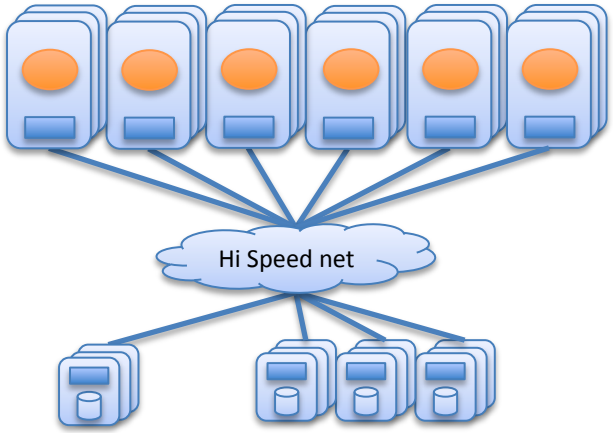
Colibri resources

- Colibri manages resources
  - Locks, file layouts, grants, quota, credits ….
  - Even inodes and data are resources
- Resource communications need to scale
  - E.g. a million processes may need to agree on an adapted file layout
  - All nodes need to read the same data
- Hierarchical flood-fill mechanism (not new)

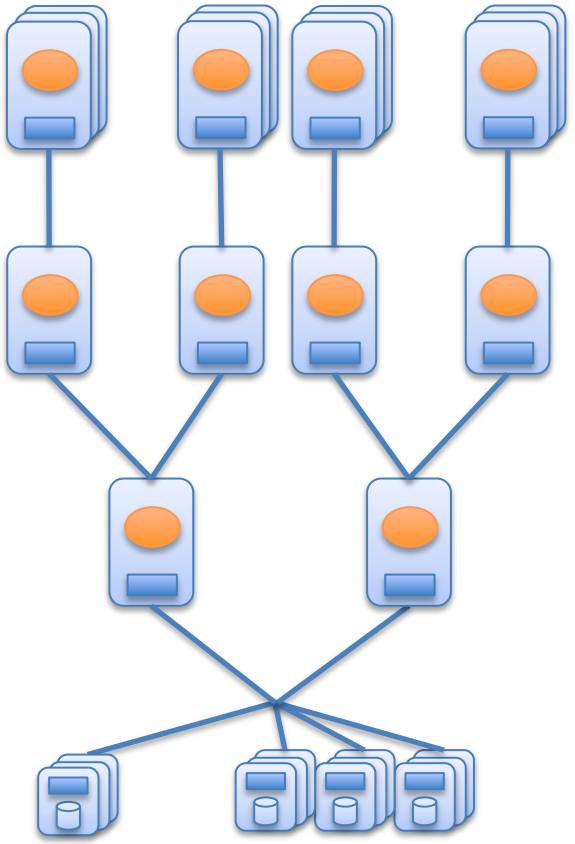
Architecture decomposition



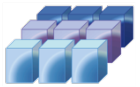
Scalable communications



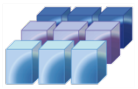
Physical Organization of C2 Cluster



Logical Organization for Resource Management

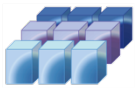


Use case example – HDF5 file I/O @100PF



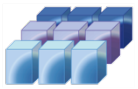
Lessons from benchmarking

- 1 TB FATSAS drives (Seagate Barracuda)
  - 80 MB/sec bandwidth with cache off
  - 4MB allocation unit is optimal
- PCI flash and NFSv4.1 RPC system
  - 10 Gige IPoIB connection
  - DB4 backend
  - 100K transactions / sec aggregate, sustained
    - Update 2 tables and using transaction log
    - One server



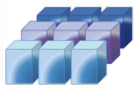
100 PF system

- Requirements
  - 100 PB capacity
  - 1 trillion files
  - 10 TB/sec IO throughput
  - Checkpoint or result dumps of 3 PB
  - 10M MDS operations / sec



100 PF Solution

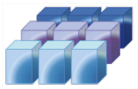
- 500 servers, each acting as MDS and DS
- Disk capacity 500 x 8TB x 40 dr = 160 PB raw
  - BW ~ 20,000 x 80 MB/sec = 1.6 TB /sec
- PCI flash
  - capacity 500 x 6 TB = 3 PB
  - BW/node: 25 GB/sec, aggregate: 12.5 TB/sec
- Network 4x EDR IB – effective BW 25 GB/sec
- MD throughput aggregate: 50M trans / sec
  - 1 copy of MD remains in flash
  - 10^12 inodes x 150 B = 150 TB, or 5% of flash



HDF5 file I/O – use case

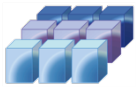
- Servers detect ongoing small I/O on part of a file
- It chooses to migrate a section of the file and the file allocation data into a container stored on flash
  - This involves revoking the file layout from all clients and sending the new layout (scalable comms)
- During migration, small I/O stops briefly
- Now 100K iops are available to flash
- When file is quiescent, data migrates back

About ClusterStor



A few other notes…

- ClusterStor market focus is “large data”
- Colibri will be delivered in increments
- The 4th increment is the exascale file system
- The 2nd is a pNFS server
- The 0th, 1st, 3rd and 5th are a secret ☺



Thank you!