Облачное распределенное хранилище данных для виртуальных машин (и не только)

Yet another Conference, 1 октября 2012

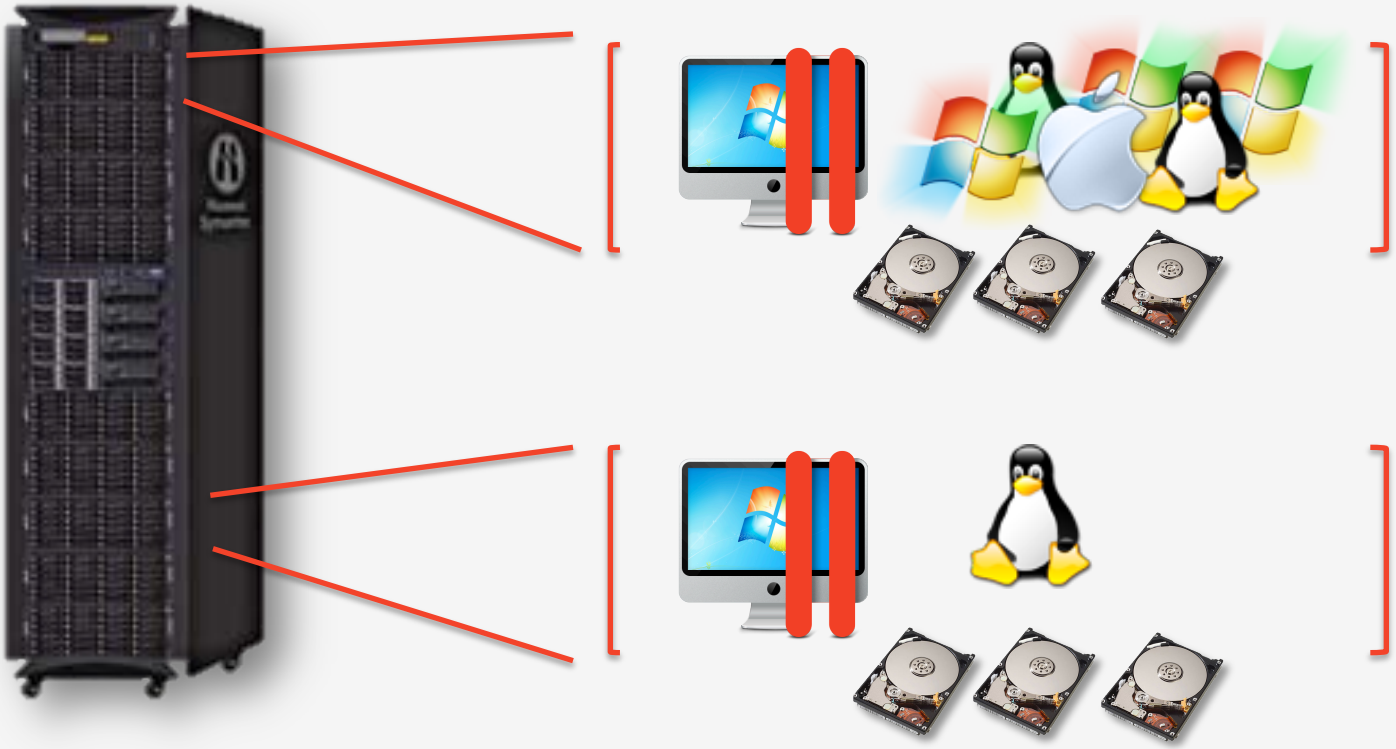
Parallels™

Profit from the Cloud

Kirill Korotaev
VP of Advanced Research, Parallels

Prehistory

Virtualization & needs

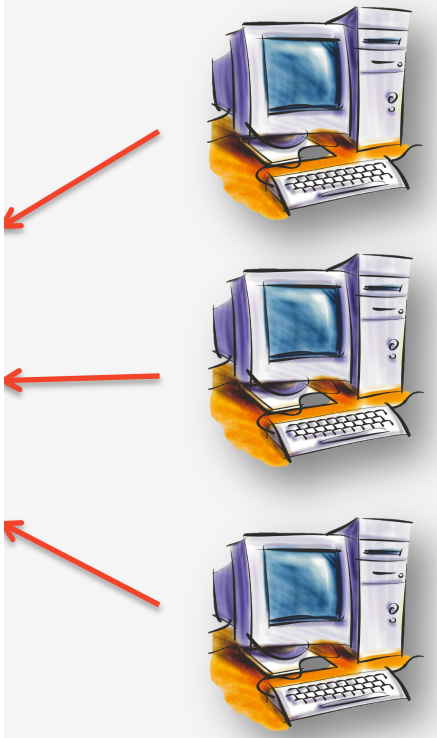


1. VM migration

2. HA on failures

SAN storage


Redundancy
High availability
Monitoring
Self healing
Reliability
Fiber channel
Performance



local

100$

How to get best of 2 worlds?

SAN

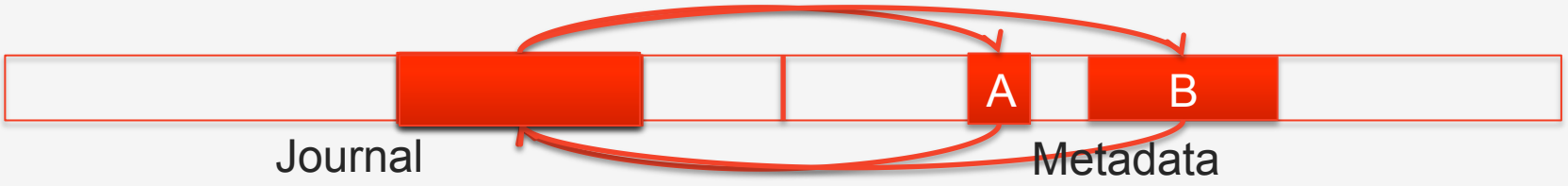
100,000$

Idea of maximum simplification

Focus on **needed** capabilities is a key to success

From required capability to consistency

Consistency - “some well defined and understandable **order**” of operations. **Required** for real file systems (NTFS, etx3/4,…) cause they use and rely on these properties of real disks.



- Immediate/Strict consistency
- Sequential consistency (all observers see the same order)
- Eventual consistency (no time guarantees, most object storages)
- Weak consistency (only synchronization primitives define order)

We want grow on demand also…

- Grow on demand means ability to allocate more then locally available (>HDD or no free space)
- Requires split of data into pieces
- Probability(any of N computer failure) is increasing with N ☹
- So redundancy/replication and auto recovery is a **MUST**
- But recovery can be done in parallel (~1TB in 10mins)
- Good news is that MTTF ~$1/T^2$, where T is recovery time.
- So let's split all objects to chunks (64Mb)
- Replicate chunks, not objects
- Spread chunks over the cluster to reduce recovery T
- Need to take into account topology of cluster ☹

More ideas

Simplifications:

- No need for POSIX FS
- Optimize for big objects only
- Can assume infrequent/slow metadata updates

Major assumption:

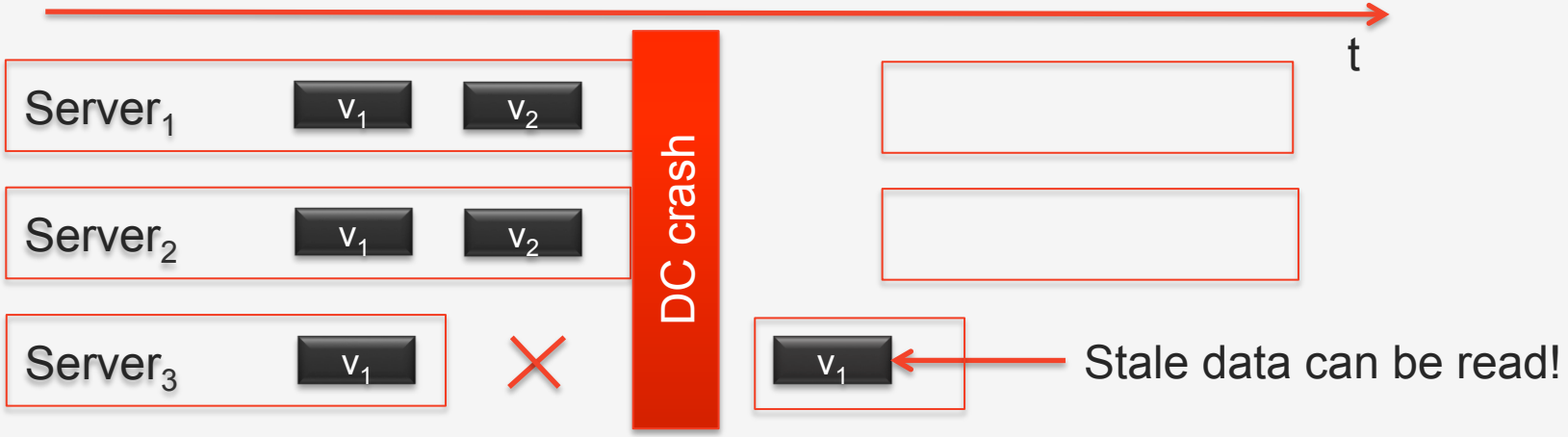
- Shit happens: servers die, even DC can crash

Why consistency is not easy? Replication…

Simplest case: object is fully stored on a single node

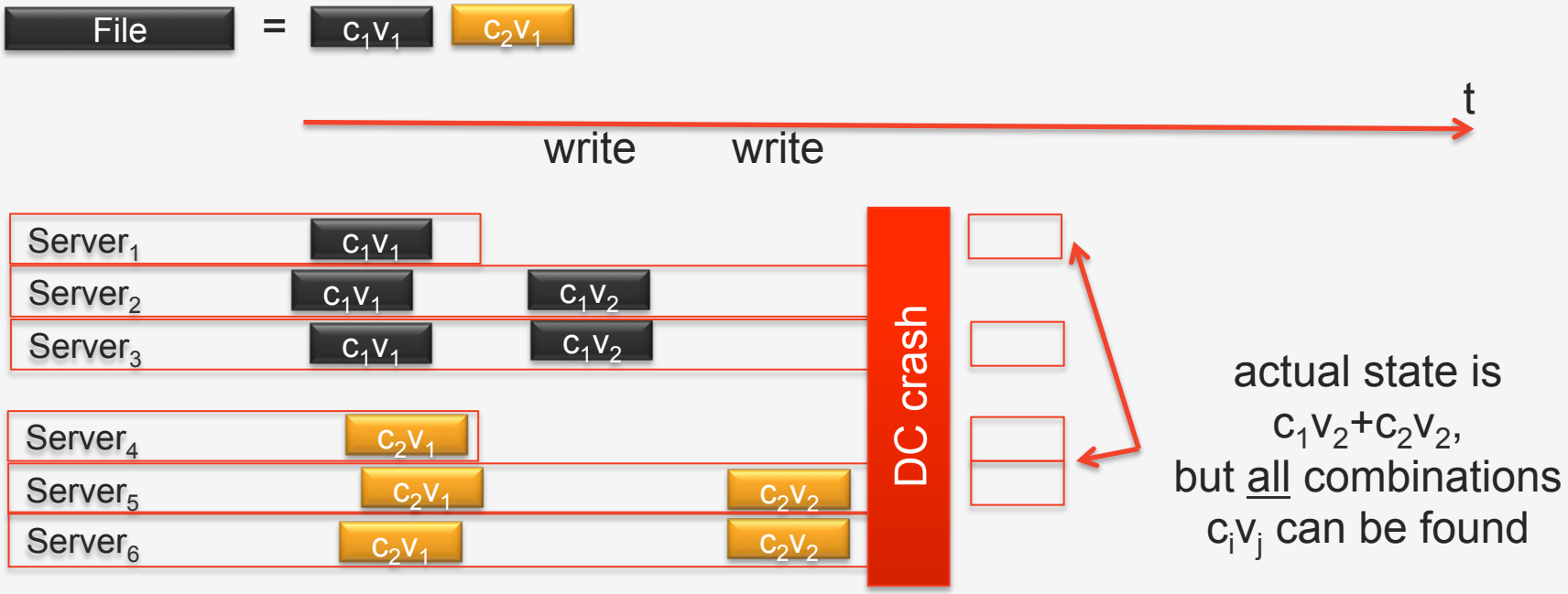
Object

Next step: object is replicated, i.e. multiple instances present



Why consistency is not easy? Data striping…

Splitting data into chunks leads to similar issues as replication:



Note: c_1v_1 or/and c_2v_1 should never be read!

So why consistency is not easy?

Data versioning is crucial and should be attributed to data, plus heavily checked on operations!

Problems with versions:

1. Tracking versions requires transactions support and sync operations. SLOW! ☹
2. Can't store versions near data only! Node can not decide alone whether it's uptodate or not.

Solution:

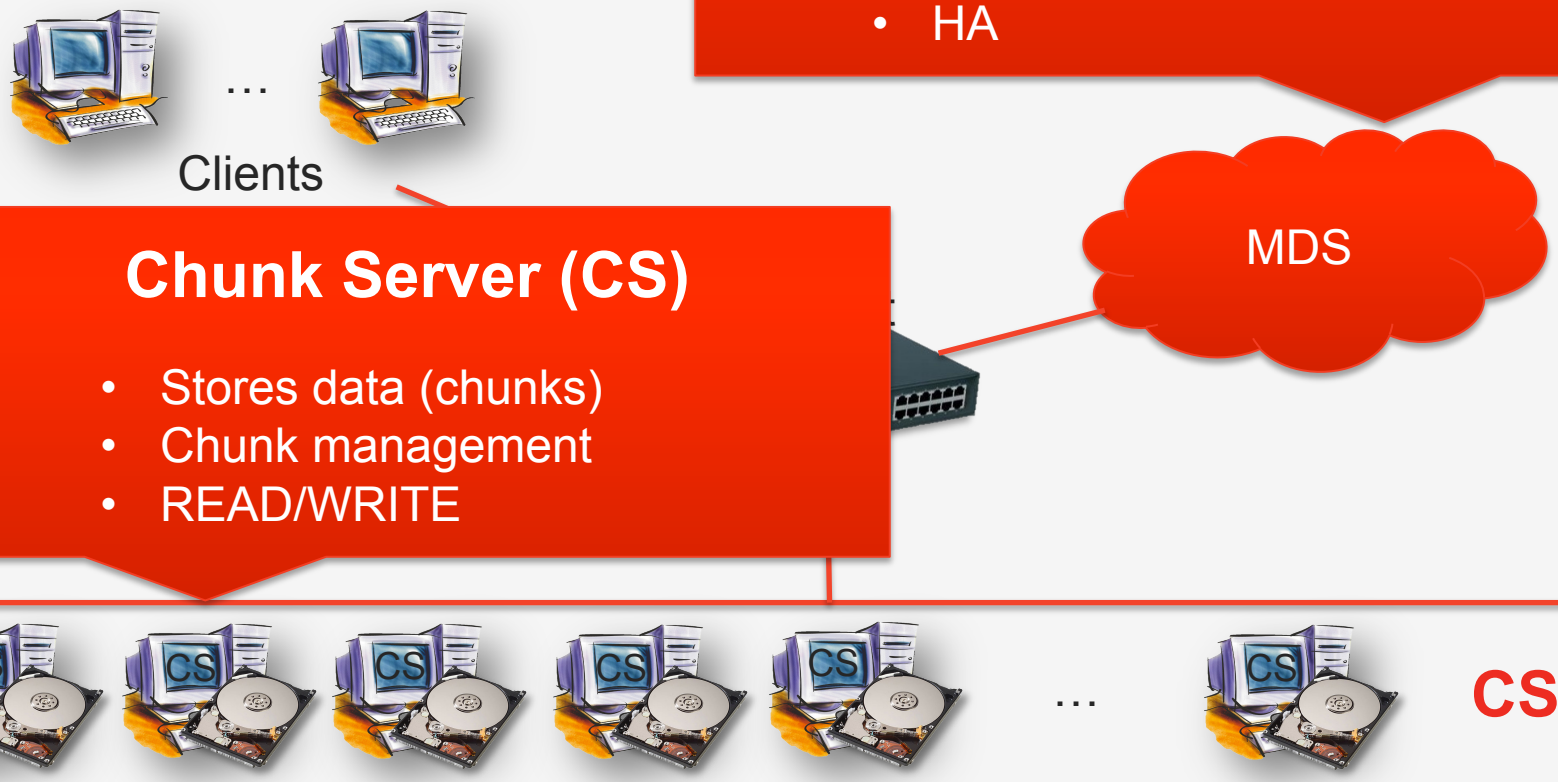
1. Let's update version only when some server can't complete write! And leave it constant on successful data modifications.
2. To solve #2 let's store versions on metadata servers (MDS) – authority which tracks full cluster state. It should be reliable and HA.

Design

Overall design

Meta Data Server (MDS)

- Metadata database
- Chunks and versions tracker
- HA



Chunk Server (CS)

- Stores data (chunks)
- Chunk management
- READ/WRITE

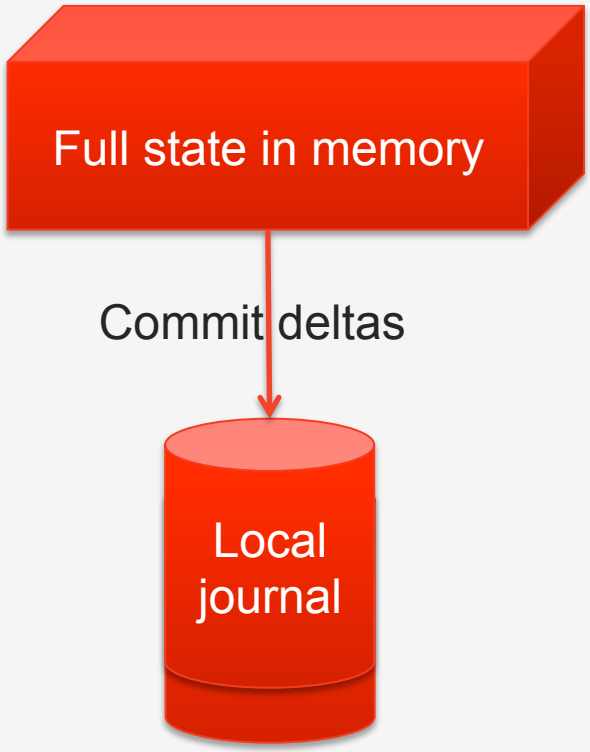
Magic (MetaData) Server HA

- Single point of failure => need HA…
- Ouch, but we don't want replicated DB, MySQL or Oracle… It's a nightmare to manage. Not scalable.
- Database adds noticeable latency per chunk access

We have to create our own replicated DB for MDS

Meta Data Server database

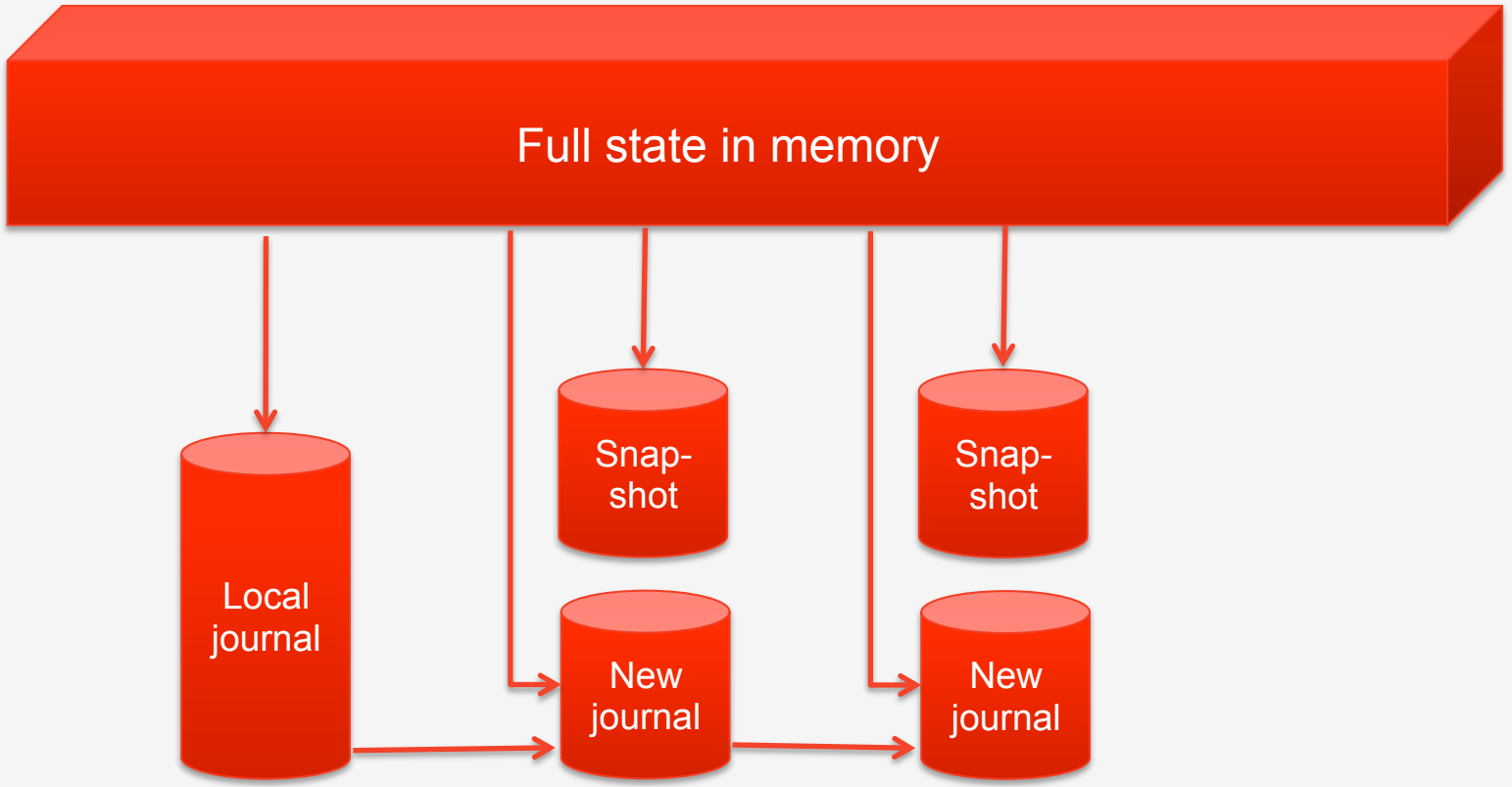
Ideas from GoogleFS:



- ~128 byte per chunk
- 64Mb RAM describe ~32 Terabytes chunks

- Journal stores modification records
- It's growing, so need compacting method
- To compact in background, need memory snapshotting
- Journal and state can be synced to outdated nodes

Meta Data Server database compacting



PAXOS algorithm

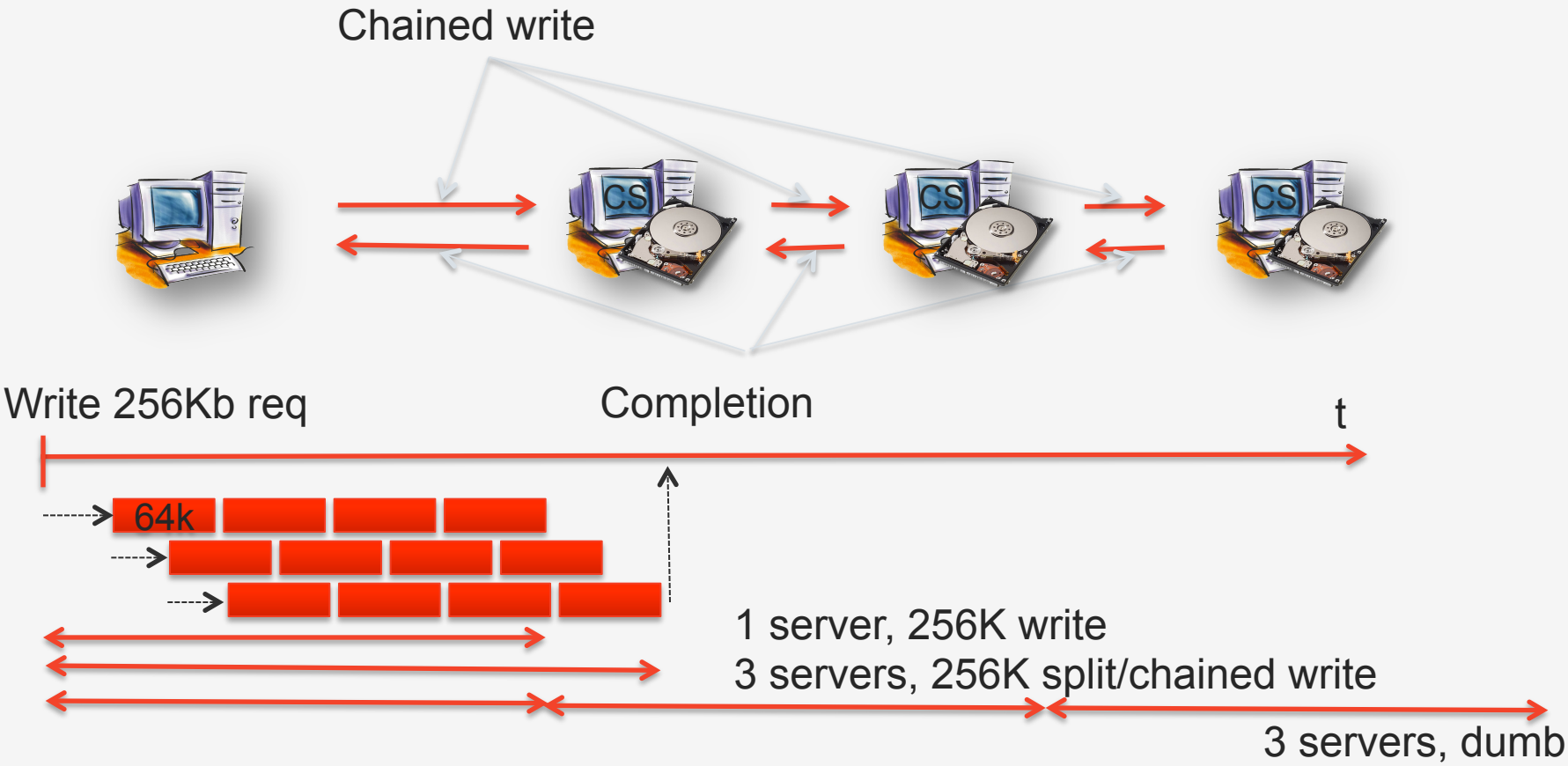
The Part-Time Parliament, Leslie Lamport:

- The Island of Paxos, parliament government
- Legislators are traders, travel a lot and often not present in chamber
- They used non-reliable messengers for notifications and communication
- Decree is adopted using ballots, need majority of votes
- Legislators are each having it's own journal of records
- Consistency of decrees is required
- Add/removal of legislators is needed
- Progress is needed

Performance

Chunk server performance tricks

Write requests are split (64k) and chained:



SSD caching

- Bursts performance > 10x times
- Allows to implement data check summing and scrubbing
- Reason: major difference from object storages – performance and latency are very noticeable and important. Latency is not hidden by WAN.

Summary

Final result

Storage system:

- With file system interface, scalable to Petabytes
- Suitable for running VMs and Containers, Object Storage, shared hosting and other needs
- SAN like performance over ethernet networks:
  - 13K IOPS on 14 machines cluster (4 SATA HDD each)
  - 600K IOPS with SSD caching
  - 1TB drive recovery takes 10 minutes!

Some experience to share

- Asynchronous non-blocking design
- QA: unit tests and stress testing are the must
- QA: how to emulate power off? SIGSTOP+SIGCONT/KILL.
  - It hangs connections and avoids RESETs as if host disappeared
- Drives/RAIDs/SSDs lie about FLUSHes.
- SSD write performance depends on data. Beware compression inside.
- Checksum everything (4GB/sec) and validate HW memtest86

Some experience to share

- All queues should be limited and controlled, like in TCP congestion control is required (both for memory limit and latency control, i.e. IO queue length)
  - One client can fire Nx4K random requests and effectively it's equivalent to 1MB requests. Congestion should be calculated correctly (taking into accoung quality of queue).
- In addition to queues limit FDs/connections etc.
- Linux sync() / syncfs() is not usable
- Linux fdatasync() is 3.5-4x faster then fsync(), but should not be used when file size changed.
- Replication should be done by pairs

Thank You

Kirill Korotaev dev@parallels.com

Try Parallels Cloud Server 6.0 beta2 at
http://www.parallels.com/product/pcs