Object Storage & Storage Analytics & EOS

Dirk Duellmann, IT-DSS

Object Stores

- Basic Idea
  - store user defined variable length byte sequences (objects) rather than fixed length blocks
    - abstracts lower level of media handling - like a file
    - but with constrained data modification semantics
      - eg create, read, delete - but **no update**
  - Usually implements media redundancy
    - using distributed object replicas or erasure-encoding
    - **no (local) RAID**
  - identified by object ID
    - simpler semantics than eg posix file name semantics
    - **no (scalable) iteration over namespace**
- Goal:
  - locally clustered store which scales better than Posix/NAS
  - in access performance, price and operational effort

For example

- CEPH
  - redundant object storage with client side calculated placement decision (CRUSH)
  - RADOS - native access
    - S3 / Swift via gateway -> scalability impact?
  - additional consolidation possibilities for sites
    - block storage (eg for VMs) used in AI project
    - CEPH file system
      - not yet supported - but “almost awesome”
- Interest from several projects to evaluate
  - CASTOR: match high-speed tape drives to “slow” disk cache for migration/recall

Object -> Cloud Storage

- Semantically similar
  - but accessed via http extensions like S3
    - tie-in with existing http caching components like SQUID
  - trivial namespace scaling via bucket separation
    - user chooses placement via object name (url)
  - commercial storage-as-service offerings and quasi-standard via Amazon docs exist
    - advantage: if “standard” service offered by a larger set of sites is needed
    - likely more suitable for volume scalability than single client performance
      - this depends more on the backend implementation than the access protocol

Server-less Object Storage

- Eg Seagate Kinetic Drive
- Single disk talks object storage protocol over TCP
  - replication/failover with other disks in a networked disk cluster
  - open access library for app development
- Other vendors are (re-)evaluating this approach
  - Why now?
    - shingled disk technology comes with natural match to semantic constraints: eg no data/metadata updates
  - Early stage with several open questions
    - port price for disk network / price gain via reduced server CPU?
    - standardisation of protocol/semantics to allow app development at low risk of vendor binding?

Caching Strategies

- ..are being discussed since many years. Progress has been steady, but not rapid:
  - – Discussion and benefit analysis is complicated by
    - missing / changing / differing access pattern knowledge
    - different cache layers affecting/hiding each other
      - – in a way which is rather opaque to end-users, service providers and framework developers
- In-process cache (TTreeCache) has an enormous benefit
  - – to reduce # of round-trips and repeat reads in a single process via protocol independent, pre-calculated vector-reads
  - – … and hence invalidated previous assumptions/optimisations
- Second biggest change (only enabled by above) is federated access
  - – which needs to be coherently integrated/evaluated wrt caching
  - – remote reading infrequently used data is more effective than attempting to cache

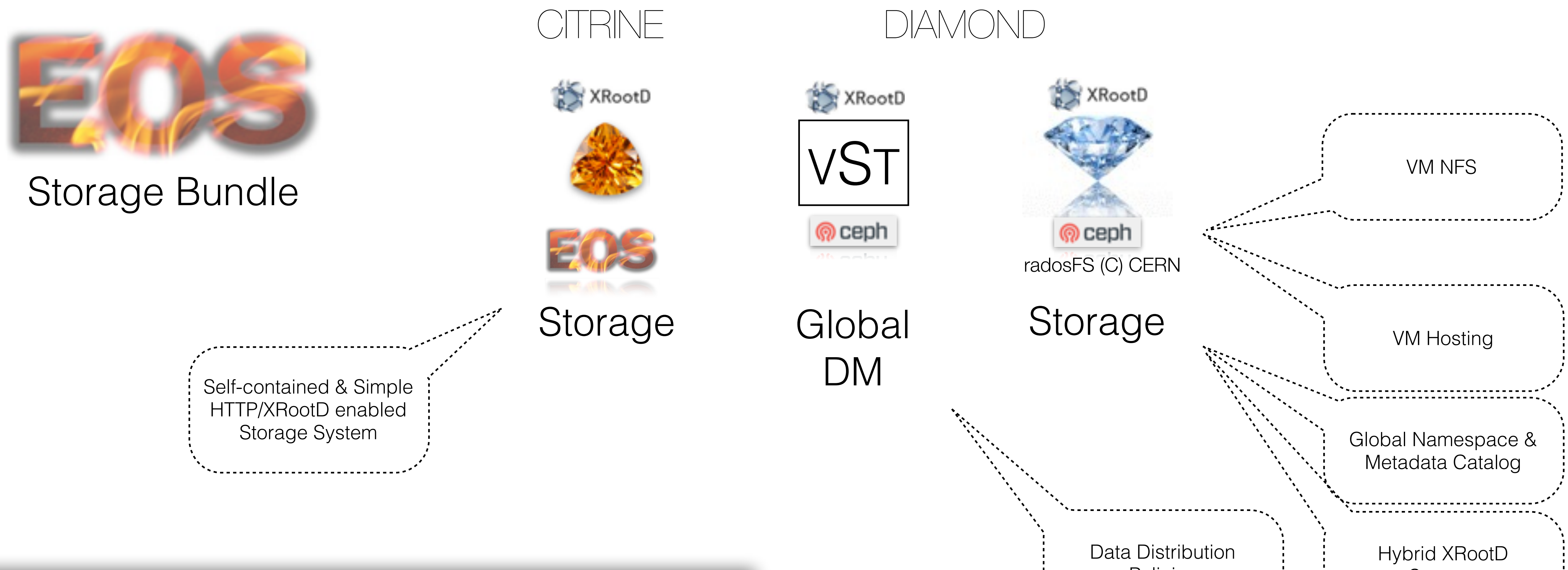
Some existing or planned cache components

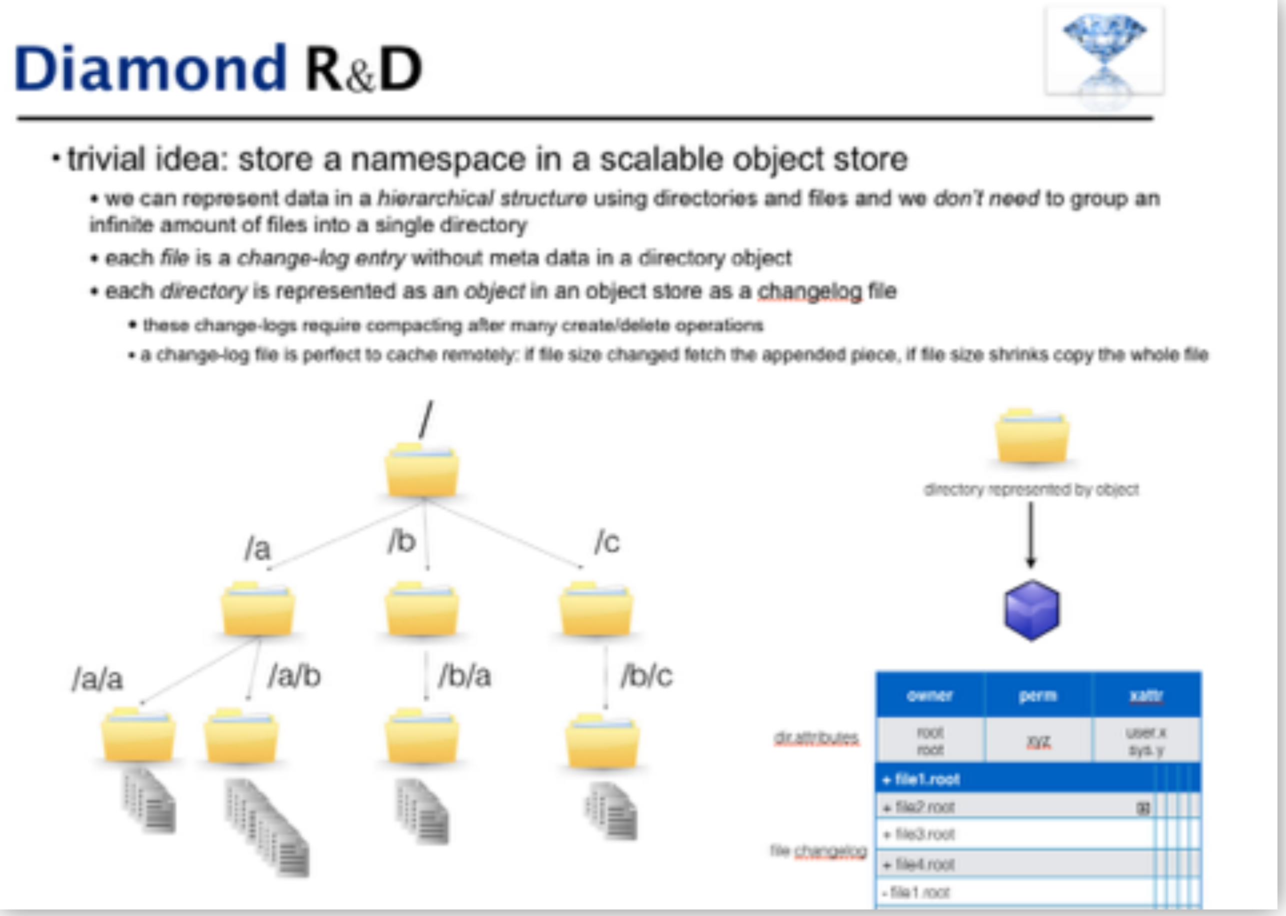
Where	What	Why	Who	How	Size	Lifetime	Accessed
Disk Server	FS cache	reduce repeated disk IO	OS/VM	pull	GB RAM	hours	kHz
Site (managed)	File Placement (SE + Catalog)	push popular data to avoid transfer I/O wait	content: exp storage: site	push	10-100 TB (disk)	months	10-100Hz
Site (unmanaged)	Proxy/CDN (eg SQUID, Xroot proxy, {Event Proxy})	reduce latency for repeat reads increase bandwitdh via tree hierarchy	storage: site optionally: exp push	pull	10TB??	weeks/months	10-100Hz
	may come with file/block/{event} granule - efficiency depends on popular fraction of cache granule						
Worker Node	Async read-ahead	increase CPU/IO overlap	job	async pull	GB (RAM)	job lifetime	<Hz
	persistent version of above	reduce repeat reads between jobs (eg user laptop case)	user	pull	10 GB (disk)	weeks?	<Hz
	FS cache for file:// access or WN download	reduce repeated disk/net IO	OS/VM	pull	GB RAM	hours	100 Hz
Process	TTreeCache	reduce network/disk round-trips	root + exp framework	pull	10-100 GB (RAM)	job lifetime	<Hz
	usage currently different between experiments and partially implemented in exp frameworks						

Ideally we would look at this with an overall throughput-increase/$ perspective **- but we still miss a lot of analytics to get there**

EOS

Disk Storage @ CERN

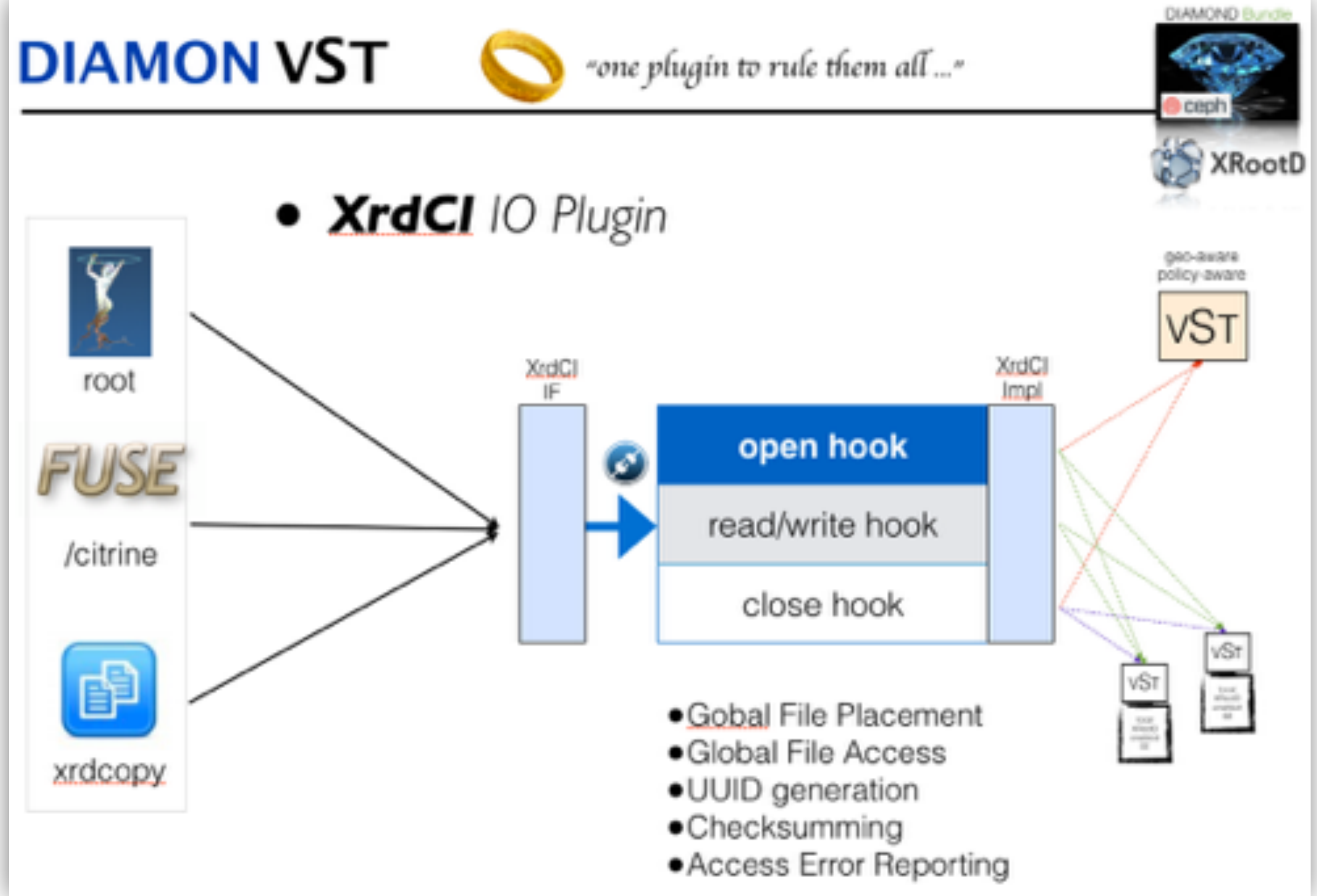




Diamond R&D

- trivial idea: store a namespace in a scalable object store
  - we can represent data in a *hierarchical structure* using directories and files and we *don't need* to group an infinite amount of files into a single directory
  - each *file* is a *change-log entry* without meta data in a directory object
  - each *directory* is represented as an *object* in an object store as a changelog file
    - these change-logs require compacting after many create/delete operations
    - a change-log file is perfect to cache remotely: if file size changed fetch the appended piece, if file size shrinks copy the whole file

owner	perm	xattr
root root	xyz	user.x sys.y
+ file1.root		
+ file2.root		
+ file3.root		
+ file4.root		
- file1.root		



DIAMON VST

"one plugin to rule them all ..."

- ***XrdCl*** *IO Plugin*

- Gobal File Placement
- Global File Access
- UUID generation
- Checksumming
- Access Error Reporting

- Focus for R&D
  - CITRINE
    - archiving interface
    - web portal extension for full administration
    - release deployment & feedback & validation
  - DIAMOND
    - global/cloud DM XRootD client plugin
    - global/cloud DM VST server
    - global/cloud DM scheduling
    - global/cloud DM FSCK
    - global/cloud DM
    - low-latency meta-data query engine
    - web portal to monitor & configure global storage orchestration
    - release deployment & feedback & validation