DAOS: An Architecture for Exascale Storage

Brent Gorda
High Performance Data Division
Intel

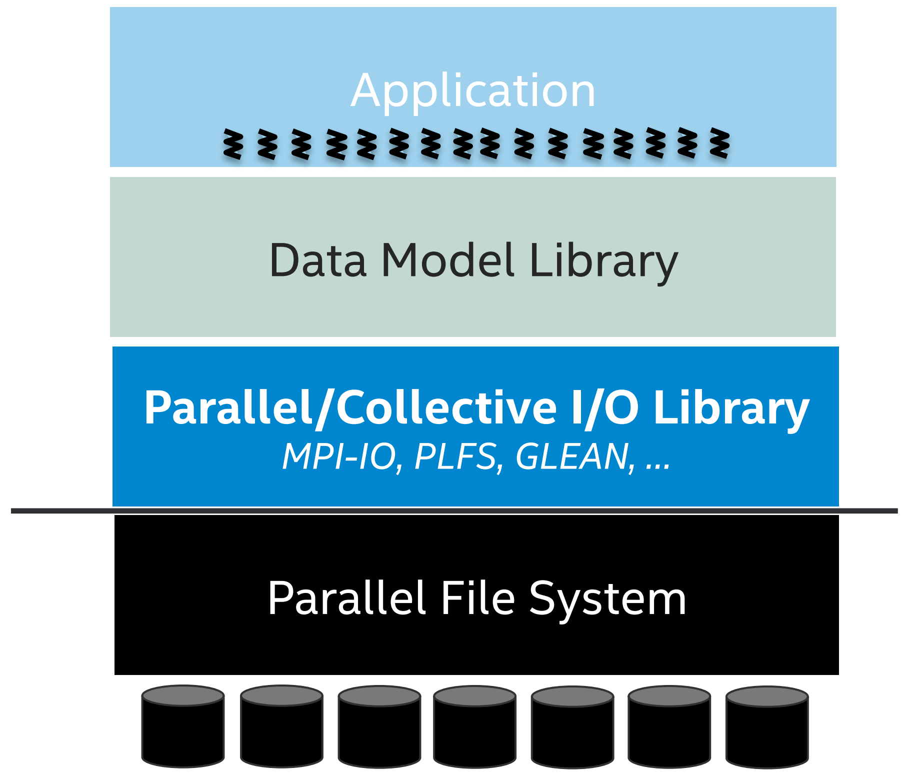
Legal Information

Dept. of Energy “FastForward” Program

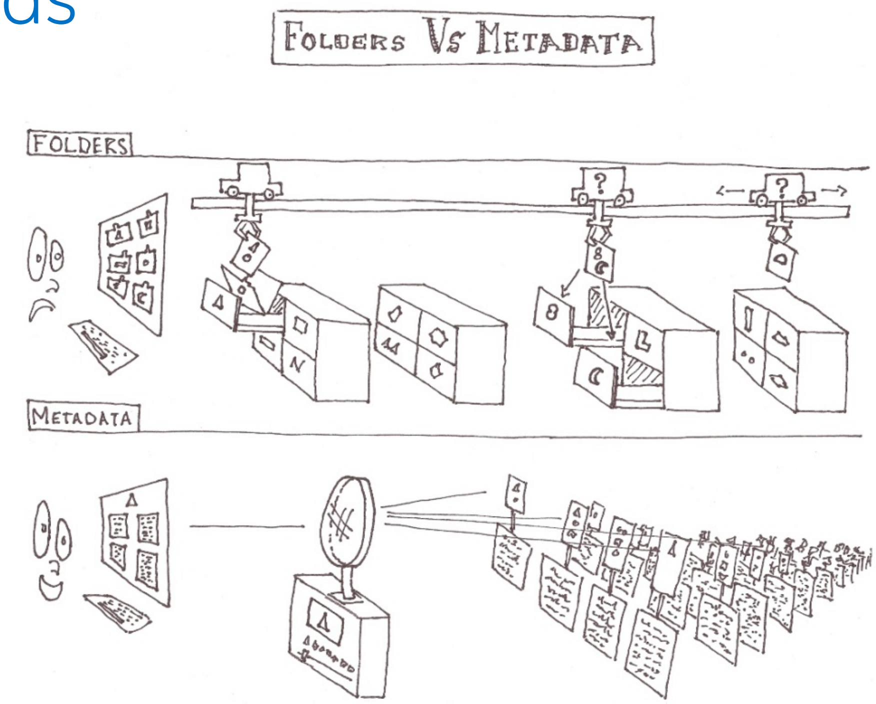
- Goal: Deliver Exascale computing ~2020
- FastForward RFP provides funding for R & D
- Sponsored by 7 leading US national labs
- RFP elements were Processor, Memory and Storage
- Whamcloud led group won the Storage portion:
  - HDF Group for HDF5 modifications and extensions
  - EMC for Burst Buffer manager and I/O Dispatcher
  - Cray for large scale testing
  - DDN for versioning object storage

Background: Posix running out of steam

- 1988 Standard for local file system
- Parallel computing with multiple kernel images has made it difficult to preserve semantics of standard
- Heroic efforts kept up with Top500
- Layers of SW libraries added to smooth out I/O (File per process) and to get beyond stream-of-bytes interface
- Survived Tera -> Peta, but don't expect it to lead going -> Exascale



Background: Emerging Trends



Increased computational power…

- Huge expansion of simulation data volume & metadata complexity
- Complex to manage and analyze

…achieved through parallelism

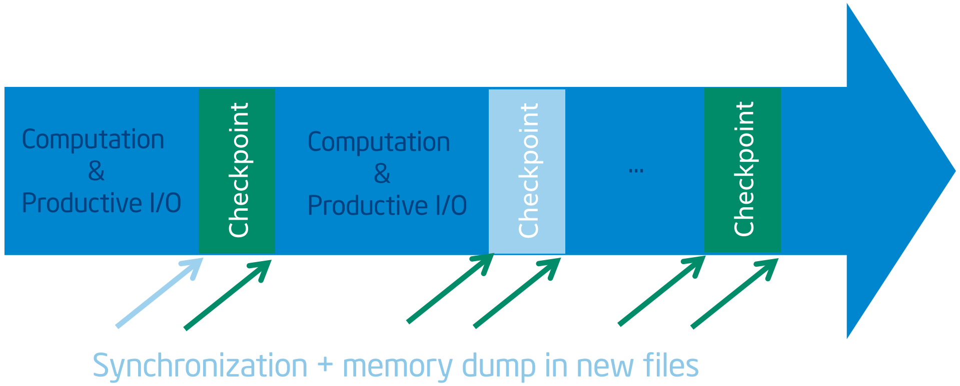
- 100,000s nodes with 10s millions cores
- More frequent hardware & software failures

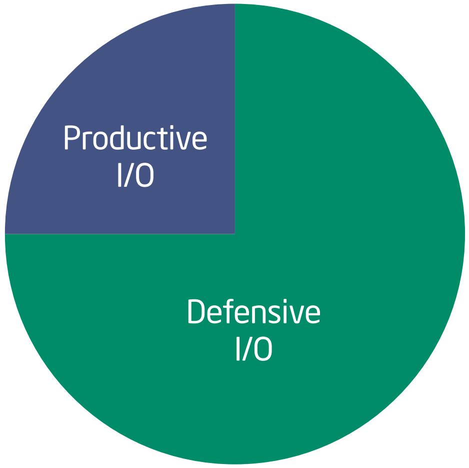
Tiered storage architectures

- High performance fabric & solid state storage on-cluster
- Low performance, high capacity disk-based storage off-cluster

Background: Primary HPC Driver is Checkpoint

Checkpoint/restart is the **primary** driver for **sustainable bandwidth** to parallel FS





I/O Distribution with Large Jobs

Disruptive Change with NVRam

NVRAM

- Byte-granular storage access
- Sub-µS storage access latency
- With ~µS network latency

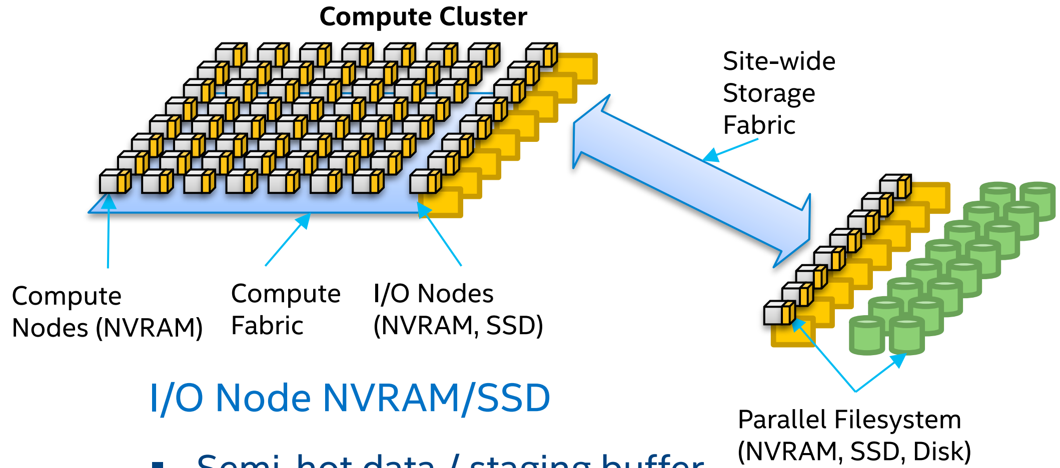
Conventional storage software

- Block granular access
  - False sharing
- High overhead
  - 10s µS lost to communications S/W
  - 100s µS lost to storage S/W
  - 1,000s µS lost to disk latency

New I/O stack requirements

- Minimal software overhead
  - OS bypass
    - Communications
    - Latency sensitive I/O
  - Security negotiated at container open
- Persistent Memory storage
  - Filesystem & application metadata
  - Hot data
- Block storage
  - SSD – warm data
  - Disk – lukewarm data

Storage Architecture



Compute Node NVRAM

- Hot data
  - High valence & velocity
  - Brute-force, ad-hoc analysis
  - Extreme scale-out
- Full fabric bandwidth
  - O(1PB/s)→O(10PB/s)
- Extremely low fabric & NVRAM latency
  - Extreme fine grain
  - New programming models

I/O Node NVRAM/SSD

- Semi-hot data / staging buffer
- Fractional fabric bandwidth
  - O(10TB/s)→O(100TB/s)

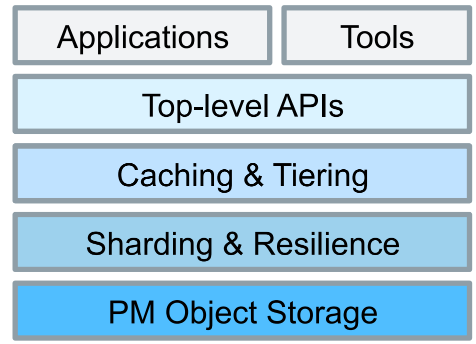
Parallel Filesystem

- Site-wide shared warm storage
  - SAN limited – O(1TB/s)→O(10TB/s)
- Indexed data

Archive

- Cold storage – O(100GB/s)→O(1TB/s)

Distributed Application Object Storage



Exascale I/O stack

- Extreme scalability, ultra fine grain
- Integrity, availability, resilience
- Unified model over site-wide storage

Multiple Top Level APIs

- Domain-specific APIs: HDF5*, SciDB,* ADIOS*
- High-level data models: HDFS, Spark, Graph A.
- Posix

Caching and Tiering

- Data migration over storage tiers
  - Guided by usage metadata
  - Driven by system resource manager

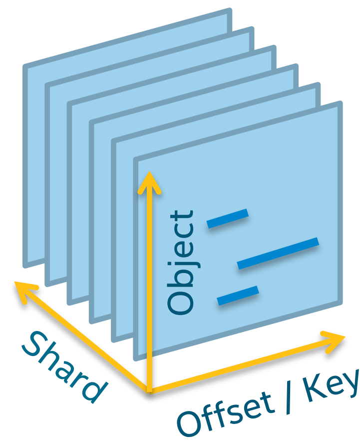
Sharding and Resilience

- Scaling throughput over storage nodes
- Redundancy across storage nodes

Persistent Memory Object Storage

- Ultra-low latency / fine grain I/O
- Fine-grain versioning & global consistency
- Location (latency & fault domain) aware

DAOS-M Object Storage



Multiple Independent Object Address Spaces

- Versioning Object PGAS
- Container = {container shards} + metadata
  - Container Shard = {objects}
    - Object = KV store or byte array
    - Sparsity exposed
  - Metadata = {shard list, commit state}
    - Minimal
    - Resilient (Replicated state machine)

Maximum concurrency

- Byte-granular MVCC
- Deferred integration of mutable data
- Writes eagerly accepted in arbitrary order
- Reads sample requested version snapshot

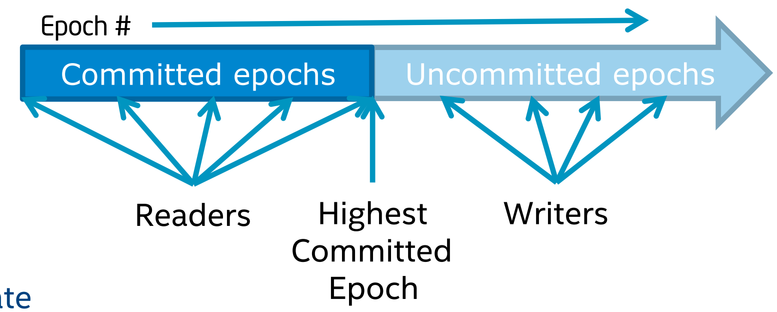
Distributed Transactions

- Prepare: Send updates tagged with version ‘t’
- Commit: Mark version ‘t’ committed in container MD
  - Version ‘t’ now immutable and globally consistent
- Abort: Discard version ‘t’ updates everywhere

Low latency

- End-to-end OS bypass
- Persistent Memory server
- Userspace fabric drivers

Epoch transactions



- Epoch based transactions
  - General purpose **A**tomic, **C**onsistent, ~~Independent~~, **D**urable update
    - Arbitrary numbers of collaborating processes and storage targets
  - Versions become visible on commit
    - Finish 'x' signals all writes in epoch 'x' done
    - Epoch 'x' committed when it & all prior finished
    - Readers see consistent data / atomic changes
    - Arbitrary rollback
- Multi-version concurrency control
  - Byte granularity to eliminate false block sharing conflicts & alignment sensitivity
  - Eliminates blocking, locking and other unnecessary serializations
    - Writers don't block readers: readers can always read HCE
    - Readers don't block writers: read from immutable version whereas write to new transactions
    - Writers don't block writers: they operate on different transactions
- Leveraged by I/O middleware
  - Consistency: end-to-end integrity, replication & erasure coding
  - Versioning: incremental replication / sync

Global Namespaces

System Namespace

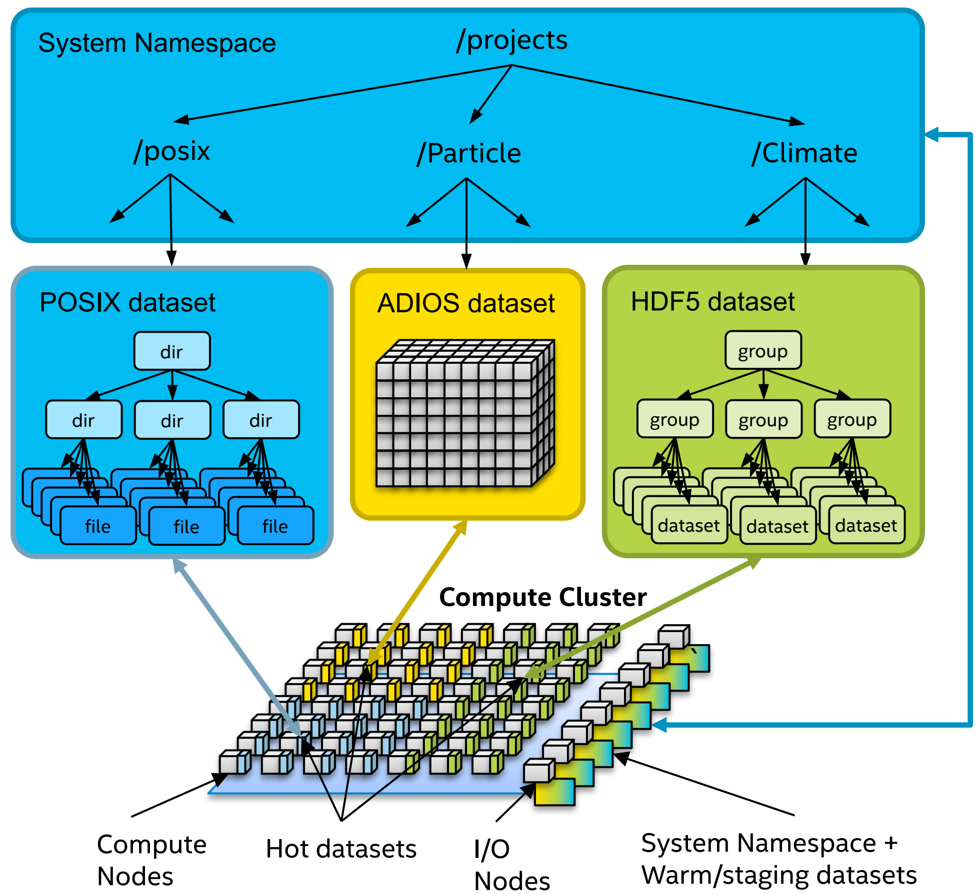
- “Where’s my stuff”

Object Namespaces (Containers)

- “My stuff”
  - Entire simulation datasets

Data Movement possibilities

- Data shared on the system – In situ
- Checkpoints need not traverse network



intel