Storage Architectures for
Big Data in the Cloud

Sam Fineberg
HP Storage CT Office/ May 2013
Overview

Introduction
What is big data?
Big Data I/O
Hadoop/HDFS
SAN
Distributed FS
Cloud
Summary
Research Areas
What is big data? (for this talk)

Applications that process large volumes of data (many terabytes-petabytes)
- Highly parallel – scale-out
- Scan oriented
- Regular inter-processor communications pattern

Examples
- MapReduce
- Pattern matching
- Log analysis
- Unstructured data analysis
Big data I/O

Common patterns
• Sequential scans
• Map/reduce
• Sorting
• Shuffle
• Index/graph traversal

Disk vs. SSD
• Performance improvement depends on access method and sequentially of access
• Do frameworks like Hadoop even make sense for SSD? Does SSD make sense over Ethernet?
• For now, let's focus on disks disks, SSD is another talk
MapReduce in Hadoop

HDFS on local DAS

Compute nodes are part of HDFS, data spread across nodes.
Hadoop Distributed File System - HDFS

Architecture
Java application, not deeply integrated with the server OS
Layered on top of a standard FS (e.g., ext, xfs, etc.)
Must use Hadoop or a special library to access HDFS files
Shared-nothing, all nodes have direct attached disks
Write once filesystem – must copy a file to modify it

HDFS basics
Data is organized into files & directories
Files are divided into 64-128MB blocks, distributed across nodes
Block placement is handled by the “NameNode”
Placement coordinated with job tracker = writes always co-located, reads co-located with computation whenever possible
Blocks replicated to handle failure, replica blocks can be used by compute tasks
Checksums used to ensure data integrity

Replication: one and only strategy for error handling, recovery and fault tolerance
Self Healing
Makes multiple copies (typically 3)
HDFS File Write Operation

1: create
2: create
3: write
4: write packet
5: ack packet
6: close
7: complete

3-way replication

Image source: Hadoop, The Definitive Guide Tom White, O'Reilly
HDFS File Read Operation

Image source: Hadoop, The Definitive Guide Tom White, O'Rei
HDFS on local DAS - Pros and Cons

**Pros**

- Writes are highly parallel
  - Large files are broken into many parts, distributed across the cluster
  - Three copies of any file block, one written local, two remote
  - Not a simple round-robin scheme, tuned for Hadoop jobs
- Job tracker attempts to make reads local
  - If possible, tasks scheduled in same node as the needed file segment
  - Duplicate file segments are also readable, can be used for tasks too

**Cons**

- Not a replacement for general purpose storage
  - Not a kernel-based POSIX filesystem
  - Incompatible with standard applications and utilities (but future versions of Hadoop are adding more other application models)
- High replication cost compared with RAID/shared disk
- The NameNode keeps track of data location
  - SPOF - location data is critical and must be protected
  - Scalability bottleneck (everything has to be in memory)
  - Improvements to NameNode are in the works
Other options - HDFS with SAN Storage

Storage Arrays

iSCSI or FC SAN

Hadoop Cluster

Compute nodes
Other options - HDFS with SAN Storage

Storage Arrays

iSCSI or FC SAN

Compute nodes

Hadoop Cluster
HDFS File Write Operation

Array can provide redundancy, no need to replicate data across data nodes.
HDFS File Read Operation

Array redundancy, means only a single source for data
SAN for Hadoop Storage

Data is in one or more arrays attached to data nodes through a SAN

- Looks like local storage to data nodes
- Hadoop still utilizes HDFS

Pros

All the normal advantages of arrays
- RAID, centralized caching, thin provisioning, other advanced array features
- Centralized management, easy redistribution of storage

Retains advantages of HDFS (as long as array is not over-utilized)

Easy failover when compute node dies, can eliminate or reduce 3-way replication

Cons

Cost? It depends

Unless if multiple arrays are used, scale is limited
- And with multiple arrays, management and cost advantages are reduced

Still have HDFS complexity and manageability issues
Tightly coupled DFS for Hadoop

**General purpose shared file system**
Implemented in the kernel, single namespace, compatible with most applications (no special library or language)
  - E.g., GPFS, Gluster, IBRIX, …

**Data is distributed across local storage node disks**
Architecturally like HDFS
  - Can utilize same disk options as HDFS
    - Including shared nothing DAS
    - SAN storage
  - Some can also support “shared SAN” storage where raw volumes can be accessed by multiple nodes
    - Failover model – where only one node actively uses a volume, other can take over after failure
    - Multiple initiator model – where multiple nodes actively use a volume

Shared nothing option has similar cost/performance to HDFS on DAS
Distributed FS – local disks

Compute nodes are part of the DFS, data spread across nodes.
Distributed FS – remote disks

Scale out nodes are distributed FS servers

Compute nodes are distributed FS clients

Distributed FS inter-node Protocol
Tightly coupled DFS for Hadoop

**Pros**
- Shared data access, any node can access any data like it is local
- POSIX compatible, works for non-Hadoop apps just like a local file system
- Centralized management and administration
- No NameNode, may have a better block mapping mechanism
- Compute in-place, same copy can be served via NFS/CIFS
- Many of the performance benefits

**Cons**
- HDFS is highly optimized for Hadoop, unlikely to get same optimization for a general purpose DFS
  - Large file striping is not regular, based on compute distribution
  - Copies are simultaneously readable
- Strict POSIX compliance leads to unnecessary serialization
  - Hadoop assumes multiple-access to files, however, accesses are on block boundaries and don’t overlap
  - Need to relax POSIX compliance for large files, or just stick with many smaller files
- Some DFS’s have scaling limitations that are worse than HDFS, not designed for “thousands” of nodes
What about the cloud?

Comparison to Big Data Clusters*

**Similarities**

• Scale-out, lots of compute instances
• Lots of storage, multiple options
• Highly networked

**Differences**

• Cloud is Virtualized – not designed for optimal per node performance
• Cloud is inherently Multi-tenant
• Cloud typically doesn’t use the fastest networking technology
  • Ethernet (not IB)
  • VLANs, security, …

* Note that we are not including dedicated big data offerings in this analysis
Cloud Storage

VM Host

SAS/SATA

VM

REST

Cinder + iSCSI/FC

Object Storage

VM Host

SAS/SATA

VM

VM

VM

VM

VM

Storage Array

© Copyright 2013 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.
Cloud storage options for big data

Amazon/OpenStack IaaS Storage

- Local disk
  - Files on VM host, accessed through LVM
  - Not persistent, typically not very high performance
- Remote persistent volume storage – EBS, Cinder
  - Remote volumes, typically iSCSI
  - Varying implementation and performance
- Object storage
  - REST based access, any application can access it
  - Limited access modes – sequential, write-once

Hybrid Options

- Dedicated cluster
  - E.g., Amazon Elastic Mapreduce
  - Similar to remote local DAS or remote DFS case
- Remote DFS
  - Clients are VMs running in cloud
  - Could be as simple as data on NFS
  - Or, tightly coupled remote DFS with storage on dedicated nodes
Summary

There are lots of storage architectures to choose from

What is the predominant access mode?
• Read or write oriented
• Sequential or random
• Local or remote

Need to consider entire workflow
• How does data feed into and out of the cluster
• Do multiple types of processing need to happen to data
• Will data be retained, where, how protected…
• Manageability is important – often trumps performance

Will the job run on a dedicated cluster or the cloud?
• Virtualized or not? Dedicated service?
• Local or remote disks?
• Block or object storage?
Research areas

What types of storage make the most sense for what workloads?
• SSD vs. disk, object vs. block
• Shared vs. dedicated storage
• Tightly or loosely coupled? POSIX or not?
• Caching

Create some hard data
• Choices are often based on religion more than data – hardware cost vs. TCO
• Consider entire chain, not just a micro benchmark
• Small improvements don’t matter if you have to give up manageability/ease of use

How does this all change in an NVM world?
• Does NVM address the entire supply chain, or just the data processing part?
• How can we take advantage of NVM without losing manageability?
Thank you