Hadoop Overview

onTarget
TRAINING SOLUTIONS

Agenda

- History
- Design Decisions
- Hadoop IO
- MapReduce overview
- YARN overview
- YARN basics (scheduling policies, queues, preemption)

Brief History

- 2004 Apache Nutch (open source crawler)
- 2004 Google release MapReduce papers
- 2005 Work to port MR to Nutch
- 2006 Hadoop project was born by the good pieces from Nutch

- 2008 Sort 1TB in 209s (Hadoop)
- 2008 Sort 1TB in 68s (Goggle)
- 2009 Sort 1TB in 62s (Yahoo via Hadoop)
- 2014 Sort 100TB in 1.406s (Spark)

onTarget
TRAINING SOLUTIONS

Hadoop Design Decisions

- Hardware Failure - handle hw failure transparent to the user
- Streaming Data Access - designed for Batch processing in mind, from specialized applications
- Large Data Sets - use smaller number of larger files than bigger number of smaller files
- Simple Coherency - write-once-read-many
- Data Locality - run code where data is
- Portability

onTarget
TRAINING SOLUTIONS

Hadoop Design Decisions (applicable)

- Use larger files (1G+)
- Streaming Access -- you'll read most of the file, nor a particular part of it, bandwidth > latency
- For low latency - check HBase
- Smaller number of files - file meta is stored in memory, 1G files is too many
- HDFS block size 128MB (vs hdd 512b, desktop FS 4k) - seek should take ~ 1% of seek + read
- Block size determines mapper parallelism (at least for MR)

Hadoop Design Decisions (cont)

- Queries generally traverse all data
- No indexing
  - If you read most of the data anyway
  - If you update most of the data anyway
- Runs Code where Data is
- MR vs RDBMS:

	RDBMS	MapReduce
query	Predictable (indexes)	Ad-Hoc (full read)
update/insert	Point update/insert (small)	Mass update/insert (big)

Hadoop IO (organization)

- Namenode
  - stores FS tree, permissions, blocks->datanode (non-persistently)
  - reads block->datanode info from datanode on startup
  - if it dies => the FS is GONE
- Datanode
  - stores blocks
  - stores block to file id + offset mapping, reporting to namenode periodically

It is extremely basic!

Hadoop IO (cont)

- Namenode backup is essential
  - Active standby + QLM for logs. At worst it's like cold start (30min)
- Federation - split FS in namespaces, each in separate namenode (like unix mounts)
  - Share Datanodes across namenodes1
- Java API for access
  - Also support for S3, Azure, Google Cloud, local, Web
  - C bindings (use JNI)

onTarget TRAINING SOLUTIONS

Hadoop IO (Reading)

- FileSystem talks to namenode, retrieves locations of first few blocks (urls of datanodes sorted by proximity)
- FSDataInputStream talks to datanodes, reading data, contacting namenode if it needs more addresses (for further blocks)
  - retries with next datanode in case of read/network issues, ignores failing node in the future
  - performs checksumming verification of read data, reports failures to namenode
- namenode is used only for metadata, served from memory (fast), actual data transfer is done in parallel with all datanodes, so there is no bottleneck for large files

onTarget
TRAINING SOLUTIONS

Hadoop IO (write)

- Write Flow
  - App writes to first node, nodes form pipleine and write to each-other in step
  - If a node fails, the data is written to less nodes, and later replicated by namenode (min repl and max repl)
- Checksumming
  - CRC-32C for every block
  - CRC-32C for every 512b (configurable) -- 1% overhead
  - writing verifies checksums at the end of a pipeline
  - reading also verifies checksum and notifies datanode, which keeps log of failed reads (to detect bad disks)
  - DataBlockScanner periodically reads data and verifies it's checksums

Hadoop IO (cont)

- Coherency model
  - Last written block might not be visible
    - hflush() // wait for nodes to reply
    - hsync() // wait for nodes to write to disk (from mem)
- Balancer
  - Makes sure Datanodes are not too empty/too full
- Formats
  - SequenceFile
  - Avro
  - ORCFile
  - Parquet
  - bzip2/lzo+index

MapReduce overview

- Map phase, Shuffle (from MR framework), Reduce phase
- Each phase has k:v pairs as I/O, configurable types
- Example
    - (offset-of-line, line)
    - MAP phase
    - (year, temp)
    - MR Framework (shuffle)
    - (year, [temp1, temp2, ..., tempN])
    - REDUCE
    - (year, max-temp)

onTarget
TRAINING SOLUTIONS

MapReduce Overview

- MapReduce job:
  - Input files
  - Map and Reduce tasks (class with one method)
  - Configuration
- Flow
  - [map] Input data is split in *splits* (pieces of equal size)
  - [map] A mapper task is run on each split (with data locality in mind)
  - [map] Output of mapper is stored on local machine (no repl, faster)
  - [shuffle] Sorts by key after each mapper, “shuffle” keys to reduce machines (#reducers is configurable)
  - [reduce] reducer task is run (depending on # reducers)
  - [reduce] output is duplicated via HDFS
- Combiner
  - “reduce” output of map job before shuffle to avoid unnecessary copying

MapReduce overview

- Pros
  - Simple model
  - Very optimized implementations
  - Battle proven by Google
- Cons
  - IN/OUT only via HDFS (slow)
  - Sorting (in shuffle) is not always required
  - Building bigger pipelines is cumbersome
- Spark was born to address these ^^ :)

YARN Overview

- Evolved from MapReduce v1
- Resource manager for (Apps) Hadoop
- Resource request: (Memory, CPU, Locality)
- ResourceManager and NodeManager (similar to Namenode and Datanode)
- Locality hierarchy - topology fully configurable:
  - Prefer node with data block
  - (opt) Fallback to rack with node with data
  - (opt) Fallback to any rack

YARN Overview (Scheduler)

- FIFO scheduler
  - Executes requests in the order received, no application overlap
  - Simple and predictable
  - Good for small clusters where each app utilized the whole cluster
- Capacity scheduler
  - Divide cluster resources into queues
  - Submit jobs to particular queues
  - 2 jobs from different queues may run concurrently, but 2 jobs on the same queue wait for each other
- Fair scheduler
  - Supports queues, but would also use unused queue’s resources, possible preemption