Detailed Outline of Hadoop

Brian Bockelman

Outline of Hadoop

- Before we "dive in" to an installation, I wanted to survey the landscape.
  - HDFS Core Services
  - Grid services
  - HDFS Aux Services
  - Putting it all together / demos

HDFS Core Services:

Namenode & Datanode

HDFS Core Services

- There are two core service, the namenode and the datanode.
  - Communication is done between the two via Java RPC
  - Each comes with a built-in webserver. Lots of the functionality is actually exposed through HTTP pages

HDFS Namenode

- The namenode is analogous to a PNFS server, plus a few unique design decisions.
  - Namenode is in charge of the namespace.
  - In charge of maintaining replica locations (and deleting/creating new ones as necessary)
  - Almost all client operations interact with the namenode at some point.

HDFS Namenode

- Namespace is kept on disk as a file named "fsImage".
- fsImage is read once, at startup.
- Changes to namespace go to a file named "edits" -- this is the journal
  - Multiple journals can be written out (on multiple HDDs, NFS, etc).

HDFS Namenode

- The entire FS is held in memory.
  - Any operation that does not change the namespace is memory-only.
- So, the edit log is a record of changes to the namespace.
  - Journal is periodically merged into a new copy of the fsImage

HDFS Journal

- The journal is merged by the namenode when it reaches a certain size (64MB?) or # of operations.
- Merging the namespace and the journal basically requires a replay; sometimes, there's not enough memory in the namenode to hold 2 copies of the namespace; then, external process merges.

HDFS Namenode

- Each file is decomposed into blocks.
  - Default: 64MB / block.
- The namenode has a value for the default, minimum, and maximum number of replicas each file can have.
  - But this number is set per-file.

HDFS Namenode

- Block decomposition has its benefits: No more hot-spots!! A single large file might be spread among 20 servers, making it hard to take out a single one with lots of reads!
- It has its drawbacks: If you lose multiple datanodes at once, you'll probably lose more files than with file decomposition.

HDFS Namenode

- File metadata includes:
  - Directory structure
  - File -> Block list mapping
  - Ownership
  - Atime/Ctime/Mtime.

HDFS Namenode

- Directory metadata also includes built-in directory quotas (both # of files and total space).
  - This is how we plan to manage the flood of users and keeping them from doing "bad things".

HDFS Namenode

- The namenode also keeps the map of block name -> block locations.
  - Unlike namespace, block locations are not persisted to disk.
- All block location information is kept centrally so a single node can make decisions about replications.

HDFS Namenode

- Block locations are "rack-aware": the namenode can be told which rack the data is on and make intelligent decisions based on that.
- Default policy for writes is one replica on the local node and the next 2 replicas on separate racks.
  - This is done so you can purchase commodity network & power gear.

HDFS Namenode

- The namenode automatically "hears from" a datanode in a heartbeat RPC from the datanode.
- If it has work for the datanode, it sends it in the heartbeat response.
- This is how the namenode does "housekeeping": delete invalid blocks, request new copies be made, etc.

HDFS Namenode

- This model for distributing work appears to scale well.
- Approximately every hour, the namenode requests the datanode send it a full block report.
  - Very useful in case if "something bad" happened in the last hour that could have caused inconsistency.

HDFS Namenode

- The client only talks to the namenode for namespace operations and the beginning of a file read.
  - In the beginning of a read, the client gets a listing of the file's block locations and a "lease" on how long that listing will be valid from the namenode.
- Client then talks directly to datanodes.

HDFS Datanode

- The datanode is analogous to a dCache pool.
- It stores blocks on a host and serves the data movement for client I/O.
  - HDFS datanodes serve data through HTTP (uncommon) and a RPC protocol (this is what your CLI uses).

HDFS Datanode

- Datanode is "well-behaved" and designed to run on worker nodes. The memory and CPU footprints are minimal, and it doesn't cause high loads.
- Can manage multiple mounts per datanode.
  - Extreme case: Our Thumpers have 48 HDD mounted individually (no RAID!)

HDFS Datanode

- The datanode maintains checksums of all its blocks.
- A rate-limited background thread does a continuous background verification of all checksums.
  - Bad blocks are auto-invalidated; namenode is told, and a new block replica is created.

HDFS Datanode

- Upon request (about 1 hr intervals), the namenode will ask for a block scan.
- This causes a scan of the directories to compare in-memory blocks against on-disk blocks.
  - One of the biggest scalability issues!
  - Going away in new releases; cost outweighs benefits.
  - Eyeing inotify as a solution.

HDFS Namenode

- For writes, the client talks to the namenode once for each new block allocated.
- The namenode will pick several nodes to participate in the write; these form a data pipeline so blocks will have multiple copies at write time.

HDFS Datanode

- Because of high costs for block scan, we currently recommend no more than .5M blocks per node.
  - Luckily, .5 million * 64M = 32TB.
  - So, our biggest nodes run around edge of comfort range for 1 process.
  - Will be getting better.

HDFS Datanode

- Datanode sends heartbeats at 3s intervals; will be declared dead after heartbeat hasn't been heard for 10 minutes.
- If dead nodes come back, then their blocks get added back into the mix.
  - So, if you corrupt files due to many nodes leaving at once, they can be fixed by bringing nodes back online.

Grid Extensions:

BeStMan & GridFTP

FUSE

- FUSE = Filesystem in USErspace.
- A simple, stable interface to create mountable filesystems.
- Facebook has contributed a FUSE filesystem for HDFS.
  - Uses HDFS's C-bindings; basically maps POSIX calls to HDFS calls.

FUSE

- FUSE is mounted on every WN + SRM node.
- This allows apps to see HDFS without knowing anything about it.
  - And doesn't destroy distributed I/O.
- This is how CMSSW uses HDFS.

FUSE

- The following work:
  - Reading, writing (appends only)
  - ls/mkdir/rm/rmdir/cp
  - Unix user/groups (uses the string names, not the UIDs/GIDs)

FUSE

- FUSE itself is rock-solid. Never any problems.
- FUSE-HDFS interface was brand-new when we started. A few growing pains (bugs) last Nov.
- No issues since December 2008. We think we've shaken out all the interface bugs.

SRM

- We layer the BeStMan SRM server on top of Hadoop.
- BeStMan just sees a POSIX-like FS; that's all it needs to do most SRM work.
- The "gateway" mode for BeStMan is a minimal set of SRM calls. This allows BeStMan to be stateless!

SRM

- Removal of complexity (no DB of space reservation, for example) allows better performance.
- Stateless; makes it easy to virtualize.
- It will also allow easy load-balancing (LVS or DNS based).

BeStMan SRM

- We've had a good collaboration with BeStMan folks.
  - Initially, some bugfixes for compatibility with SAM.
  - New protocol selector interface.
  - Upcoming FS API interface.
  - Upcoming GUMS caching.

Selector Interface

- If you implement a java interface, you can program in your own GridFTP selection algorithm.
  - Default: round-robin through listed GridFTP servers.
- We have a Ganglia-based interface based on server load and memory available.
- Don't like it? Write your own! It's easy! (?)

Ganglia interface

- Queries ganglia periodically (60s)
- Builds a probability distribution function for the selection of gridftp server.
  - P = 0 if server is down, too high load, or swapping.
- Selection picks randomly from the pre-defined distribution.

Space Reservations

- BeStMan gateway mode currently uses "static space reservations": will allow you to use space reservation commands, but does no accounting of usage.
- Recent versions introduced an interface where you can provide a script for BeStMan to run, and the output of that script will be used as space reservation information.

Space Reservation

- With this script-based space reservation system, it'd be possible to advertise a few quotas as reservations for a few large chunks of the namespace.
- Too tedious to setup up 1000 space reservations to use with /store/user.

Space Reservation

- We're hoping to build a new interface with BeStMan that would let it programmatically determine all the space reservations and query all the quotas.
  - That way, we could provide a module that auto-configures and auto-advertises quotas as space reservations on /store/user.
  - Good Idea? Still in planning stage.

BeStMan SRM

- BeStMan devs have done these interface improvements upon our requests.
- BeStMan now definitely scales to CMS needs. Mostly limited by underlying FS.
- In a month or two, it'll be fair to expect 25-50Hz per server.
  - Should scale linearly.
- Very happy with this component.

GridFTP

- Utilize Globus GridFTP server.
  - All the things you've come to love and expect from GridFTP on your CE apply here.
  - This includes PRIMA, Globus logfiles, xinetd-based launching, familiar error messages, etc.

GridFTP

- HOWEVER: Globus-gridftp writes data in small (possibly out of order) chunks.
  - Hadoop does not support out-of-order writes.
- Plus, GridFTP -> FUSE -> HDFS client is quite a few context switches!

GridFTP DSI

- Globus GridFTP actually has a modular architecture.
- The DSI (= Direct Storage Interface) component allows us to interface Globus with any storage system.
- Plus, HDFS provides us with stable C bindings.

Globus DSI

- The GridFTP-HDFS module does the appropriate translation between Globus and HDFS.
- Plus, it re-orders streams before writing them to disk.
  - This can cause quite some overhead: up to 100MB / GridFTP process.
    - But RAM is cheap!

Globus DSI

- The approach works well, esp. combined with the Ganglia GridFTP selector.
- No performance hit per-file.
- Have clocked they system at >9Gbps.
- Been used for LoadTests for 5 months.

Hadoop Auxiliary: Secondary NN, Balancer

Hadoop Secondary NN

- Really a misnomer: I call it the "checkpoint server"
- Its job is to query the namenode, download the fsImage + journal, and merge the two.
- It saves & rotates the previous checkpoints and uploads the result back to the NN.

Hadoop Secondary NN

- Offloads the processing requirements of the merge to a different node (is about as memory-hungry running as the namenode itself!)
- Also a convenient way to handle backups: just backup the rotated checkpoints to tape / backup server.

Hadoop Balancer

- Ignoring rack-awareness, Hadoop writes to random nodes and reads from least-loaded nodes.
    - Deletes from nodes with least % free.
- Randomly writing is great for performance, but leads to evenly distributing new data - ignoring free space % differences.
    - We want each datanode to have the same % of space used.

Hadoop Balancer

- The Hadoop balancer is a Hadoop script that replicates and deletes files as necessary so all datanodes have the same % of free space within a certain threshold.
  - Unlike other HDFS components, this is a script -- we write so often, we just use it as a cron job; it'll refuse to start 2 processes

Hadoop Balancer

- We view a well-balanced system as a healthy one; things don't go wrong if datanodes start to fill up, just sub-optimal.
- We hope to see this turned into a regular daemon.
  - Esp. as the script will sometimes get stuck in an iteration.

Demonstrations

Time for Demos

- Local FS layout
- Logfiles / locations
- Hadoop CLI
- Read/write from FUSE
- GridFTP server
- SRM copy