Dynamic Hadoop clusters on HPC scheduling systems

Michele Muggiri, Luca Pireddu*, Simone Leo, Gianluigi Zanetti

CRS4

August 27, 2013

Outline

Rising interest in Hadoop

- Hadoop provides an effective and scalable way to process large quantities of data
- MapReduce suitable for many types of problems
- Hadoop ecosystem also growing in other directions
  - e.g., fast DB-style queries on very large datasets
- Growing number of applications
- Success confirmed by the growing number of users

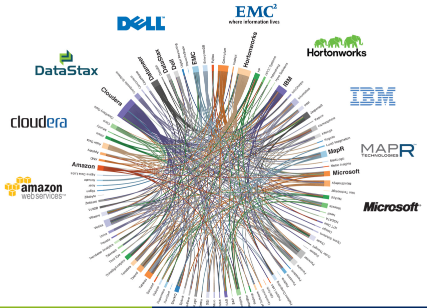


Image by Datamere

Hadoop’s goals

Hadoop has two main goals

Hadoop's goals

Hadoop has two main goals

- scalable storage
- scalable computation

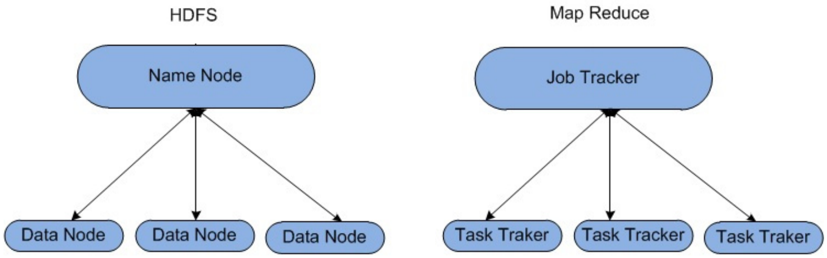
Hadoop's goals

Hadoop has two main goals

- scalable storage
- scalable computation

- Storage provided through Hadoop Distributed File System (HDFS)
- Computation provided by Hadoop MapReduce and other systems
    - For the scope of this work, for computation we focus on MapReduce

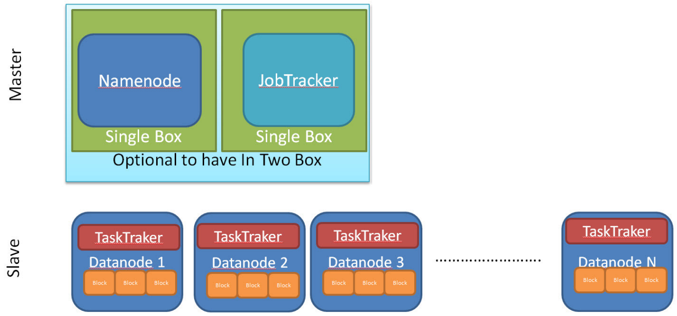
Hadoop 1.x architecture



- Two main subsystems, HDFS and MapReduce, each with a master-slave architecture
- HDFS has many DataNodes
  - store data blocks locally
- MapReduce has many TaskTrackers
  - run computation locally

Image courtesy of mplsvpn.info

Hadoop 1.x architecture



- Normally DataNodes and TaskTrackers are deployed together
- Quite complementary resource requirements
- Take advantage of data locality

Image courtesy of MSDN

Hadoop's use of resources

- Hadoop assumes it has **exclusive and long-term** use of its nodes
- It has its own job submission, queueing, and scheduling system

- This arrangement can make it complicated to adopt in some circumstances
- An important example: HPC centers, with shared clusters accessed via batch systems
  - Probably still one of the most ways to access private computing resources

Hadoop's approach to resource acquisition is decidedly in contrast with batch systems!

Adopting Hadoop

- Large, committed, operations have possibility of deploying dedicated clusters
- Others may not have the resources for a Hadoop cluster
- Some aren't sure about investing in one

And what about experimenting?

- Even setting up a temporary reasonably sized cluster
    - At worst will require sysadmin approval and intervention
    - At best will still require specific skills, which may not be easily accessible

Example application: DNA sequencing

An example of a user who has a lot of data to process but may not have Hadoop administration skills: bioinformatician!

- Interesting application of Hadoop is in processing genomic data
- Typical genomic processing workflow:
  - embarassingly parallel problems
  - mostly I/O bound
  - well suited for Hadoop
- Increasing number of Hadoop-based software for this type of work

Example application: DNA sequencing

How much data?

- Details depend on technology
- e.g., one run on Illumina high-throughput platform
- 10 days $\approx$ 400 Gbases $\approx$ 4 billion fragments $\approx$ 1 TB of sequence data

CRS4

CRS4 sequencing center

- CRS4 - largest sequencing center in Italy
- capacity of generating 5 TBases/month
    - i.e., about 25 TB of raw data
- Most processing performed with the Hadoop-based Seal toolkit

CRS4 computational capacity

- 3200 cores in its main HPC cluster
- About 5 PB of storage, most of which in a shared GPFS volume
- Managed with Grid Engine. Available to everyone at CRS4
- Runs a lot of MPI and standard batch jobs
    - *cluster cannot be entirely dedicated to Hadoop*

Hadoop allocation strategies

How can we allow Hadoop to exist in such a typical HPC setting?

- Various possible static and dynamic Hadoop allocation strategies
- Some may provide a suitable solution

Static allocation

Partition cluster: allocated part to HPC and part to Hadoop

- Works well if both partitions have regular, relatively high load
- Provides a static/stable HDFS volume

But

- not well suited for variable workloads
  - easily results in underutilization

Dynamic allocation

- Only occupy nodes when needed
    - Seems more reasonable strategy in shared HPC environments
- Not straightforward because HDFS uses node-local storage
- HDFS cluster cannot be reduced in size easily
    - data needs to be transferred off the nodes to be freed – slow!
- Number of nodes must always be sufficient to provide required storage space
    - idle cluster still occupies nodes

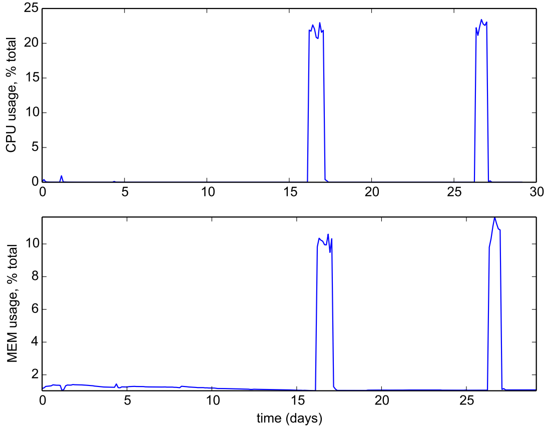
Yet, there are various possible flavours of dynamic allocation

Hadoop-on-Demand (HOD)

- Blocks of nodes allocated through a standard batch system
- HDFS and MapReduce started on those nodes
    - HDFS volume is temporary, so only useful for intermediate/temporary data
- Desired size of cluster must be decided at allocation time
- Cluster must be deallocated manually

Hadoop-on-Demand (HOD)

- allocation strategy exposed to human factors
  - given overhead/latency in allocating cluster users may be tempted to keep cluster allocated for longer than strictly necessary



Alternative approach

Alternative approach: *decouple Hadoop MapReduce and HDFS*

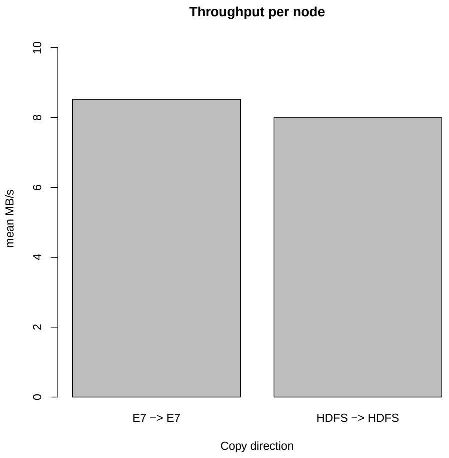
- MapReduce and HDFS may use different sets of nodes
  - can even choose to completely forego HDFS and use other storage systems
- More allocation strategies open up this way
- Drawback: risk losing data-locality

HDFS allocation

- Cluster-wide HDFS
  - Run HDFS daemons on all cluster nodes, alongside other task processes
- Dedicated block of machines to host an HDFS volume
  - Can even recycle older machines whose CPUs or RAM size are no longer competitive
- No HDFS: use some other parallel shared storage
  - use whatever is already in place
  - in addition to HDFS, Hadoop can natively access any mounted file system and Amazon S3

No HDFS

What's the price of foregoing HDFS? YMMV



- Use hadoop distcp to copy 1.1 TB of data
- 59 nodes, HDFS replication factor of 2
- Each bar is the mean of 3 runs

Warning!

- HDFS scales to 1000s of nodes
- This test only tests ~ 60
- Our nodes only have 1 disk

MapReduce allocation: per-job

- Acquire nodes, start JobTracker and TaskTrackers, run job, shut down and clean-up
  - Such a solution was implemented for SGE by Sun
- Lack of a static JobTracker nodes is not very simple for users and will not work with higher-level applications (e.g., Pig, Hive)

Static JobTracker, on-demand slaves

- Static JobTracker, dynamic cluster
- We've built a solution based on this strategy: *Hadoocca*

Outline

Hadoocca

- Hadoop MapReduce natively supports dynamically adding and removing slave nodes (Task Trackers)
  - a feature normally used to handle node failures
- Keep a static JobTracker server
- Monitor its queues
  - allocate task trackers as capacity as needed
- Two main components: Load Monitor, Task Tracker manager

Load monitor

- Monitors Hadoop JobTracker
- Periodically polls it for its map and reduce task counts:
  1. capacity
  2. running
  3. queued
- Currently implemented using JobTracker's command line interface
  - `hadoop jobs` program
- Based on number of queued tasks decides how many task trackers to launch

Scheduling formula

Scheduling decision is currently simple and intuitive

- Calculate the number of nodes required to put all tasks in running
- Try to allocate them, capping at a limit per scheduler iteration
- Iterate again after a delay and repeat the process

Scheduling formula

Roughly boils down to the following:

```
map_nodes = ceil( (n_map_tasks queued + n_map_tasks running)
                    / n_map_tasks_per_node)
red_nodes = ceil( (n_red_tasks queued + n_red_tasks running)
                    / n_red_tasks_per_node)
total_nodes = max(map_nodes, red_nodes)
capped = min(node_limit, total_nodes)
new_nodes = capped - (nodes_running + nodes_queued)
to_allocate = min(new_nodes, max_nodes_per_iteration)
```

- The system then queues `to_allocate` new nodes and sleeps until the next iteration

Scheduling formula

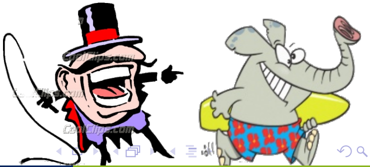
- Limiting the number of nodes requested per iteration slows down growth
- Play nice with neighbours and avoid flooding the cluster
- Avoid starting nodes unless they're really necessary
    - due to excessively quick tasks or errors causing crashes

Running task tracker nodes

When the Load Monitors tries to allocate a node, it actually queues a job through the batch system.

The job, once it starts running performs these steps:

- Verify that there are still tasks outstanding
- Uses default hadoop mechanism to start task tracker
- Keeps running, monitoring task tracker process
- When no tasks are running on task tracker for some time, terminates it
  - tell JobTracker node is to be excluded
  - use standard Hadoop tasktracker shutdown command
  - clean up scratch space



Running task tracker nodes

- We don't have a simple way to monitor the task tracker's operations
- Instead:
  - monitor daemon's local scratch space
  - specific directories are created while a task is running
  - by seeing which paths exist we know which tasks are running

Deployment at CRS4

- Use shared GPFS for storage
- Job tracker web interface accessible to all users
- **Multi-user setup**
  - use Hadoop's own mechanism for running multi-user cluster
  - `LinuxTaskController` with accompanying setuid-root binary
- Task processes with client's EUID
- Service daemons (JobTracker, TaskTrackers) run as a system user

Patches to Hadoop

Multi-user with GPFS required some small patches to Hadoop

- Hadoop MapReduce uses staging directories on shared FS to pass job info to task processes
- Some code assumes Hadoop runs as super-user and has full access to file system
    - enforces permissions that are too restrictive for Hadoop user and task user to both access data
- Patched Hadoop to relax those checks

Deployment layout

Cluster configuration and binaries

- stored on a shared volume mounted on all cluster nodes
- paths provided via environment variables loaded with “module”
  - e.g.,

```
$ module load hadoocca
$ hadoop jar MyJob.jar \
    file:///home/pireddu/data/input \
    file:///home/pireddu/data/output
```

Outline

Hadoocca

- An allocation strategy for dynamic Hadoop MapReduce clusters
  - doesn't require HDFS
  - analogous to Amazon's EMR
- Open source implementation
- Suitable for HPC centres who
  - don't want to dedicate portion of their cluster exclusively to Hadoop
  - are happy with a small- to medium-sized installation
  - have a suitable storage infrastructure

In production use

- We've been using Hadoocca at CRS4 for about 8 months
- Born as a prototype, but it's still running!

- Static JobTracker makes running Hadoop programs really easy
- Compatible with Pig and Hive
- Increased adoption of Hadoop at CRS4
  - With the addition of tools such as Pydoop script, it has become a common way to write simple parallel programs
- Without this type of solution it would have been quite difficult to bring Hadoop as a steady fixture at CRS4

Future development

- Release current prototype code
- Rewrite
  - support for Hadoop 1.x and 2.x
  - generalized queuing code (maybe DRMAA)
  - modular scheduling

Thank you

Thank you!

Questions?