Introduction to Commodity Linux clusters

To show all **commodity parts** that go into making a **cluster**, both HW and SW

Ezio Corso
Maria Verina

Overview

- **Part I:** *functions* + building blocks + architecture
- **Part II:** Maria Verina will explore further various aspects + specialised lectures on specific topics.

- Top-Down approach:
  - State **what** scientific/engineering user wants to achieve
  - Break it down into constituent parts
- Highlight ***non-functional*** requirements from ***operations perspective:***
  - Other HW/SW added
  - Making of a ***service*** delivered to users
- Highlight engineering of non-functional requirements, for a given service level i.e. High Availability

What are we delivering?

Main objective:

Run MPI parallel code written by user, that processes large dataset off central NFS storage, and saves the results back to central NFS storage.

Function of a Cluster

- In principle can be achieved ***without*** any cluster:
  - PC connected in GETH
  - Install Linux + OpenMPI
  - Login to each machine
  - Execute ***mpirun*** to enable MPI
  - Wait for code to complete
  - Switch off

- No Resource Manager, no Infiniband, no expensive and fancy HW: you get your result.

Function of a Cluster

Operational issues:

- You may do it once in a while, but ***doing it repeatedly*** becomes ***highly time consuming***

- Already with ***dozens of hosts*** it is ***cumbersome***: no amount of ssh/scripting will do

- If ***multiple users*** need to run the code:
  - it is ***problematic to organise/schedule/optimise*** activities
  - ***Use resources***: i.e. some may need 4 nodes for 12 hr, others may need 200 cores for 24 hours, etc.

- Having ***multiple application*** makes it ***unmanageable to setting up*** and handle the ***code execution environment***

Function of a Cluster

Technical issues:

- MPI calculation performance will be awful: ***GETH latency***.

- If ***MPI breaks in a single node***, there is no way other nodes will know and the ***application hangs***: MPI does not have facilities to communicate error conditions and recover from them.

Function of a Cluster

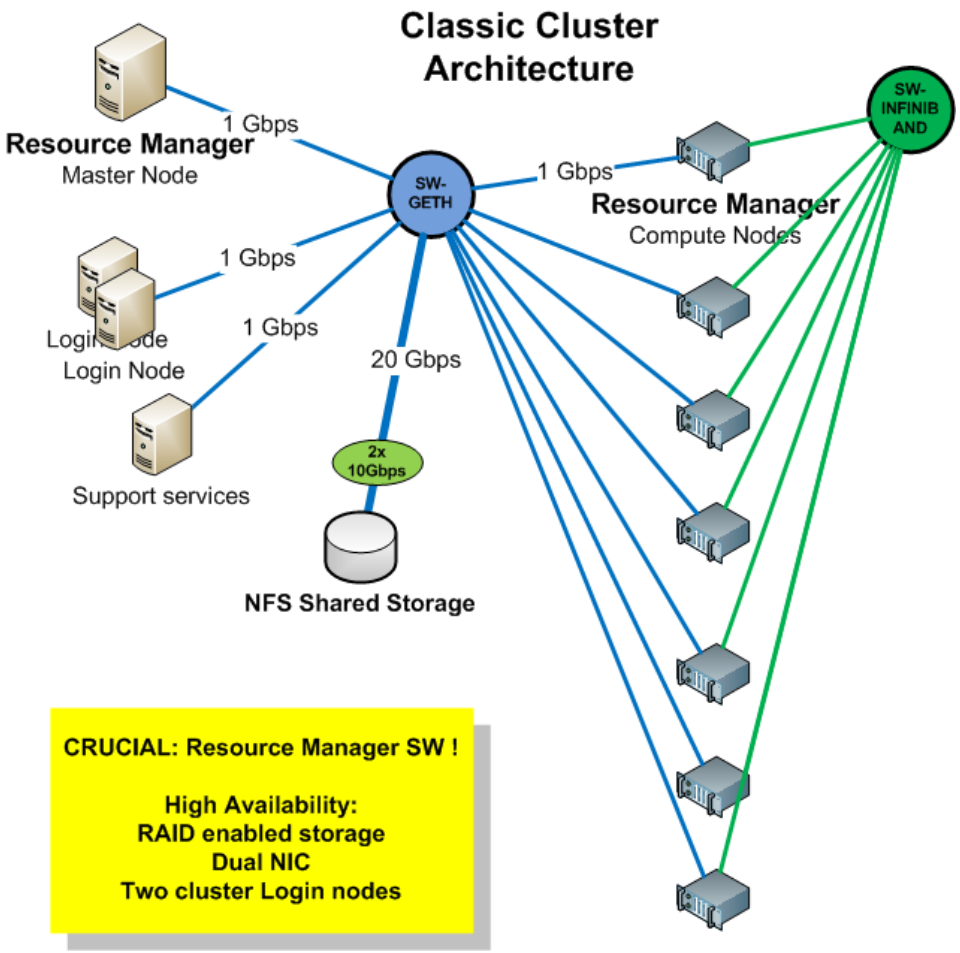
There really are operational goals
because
we are delivering a service

- Efficient use of resources
- Efficient concurrent use by multiple users
- Efficient management of multiple applications
- Availability of the service that users now rely on
- Technical handling of specific dynamics of MPI.

How to technically deliver all this?

The Classic Cluster Architecture

The Classic Cluster



The Classic Cluster

- Install and use a Resource Manager such as Torque/MAUI, IBM LSF, etc.
- **Job:** the code that must run + request for specified resources
- **Compute Node:** hosts where the code runs
- **Master Node:** host that co-ordinates the execution.
- Logical access to resources organised in **Queues:** jobs will be placed in queues.
- RM will analyse requested resources and schedule Job according to set policies.
- At runtime, RM grabs only needed nodes, switch on MPI, invoke the command line, executes specified commands, keep an eye on MPI in case one of the daemons in one of the nodes breaks.
- If resources are available and compatible with the requests, multiple jobs will run in parallel in their own set of resources
- When Job finishes, resources given back to pool, and new Jobs compatible with the resources are scheduled
- So: you efficiently manage multiple users, using multiple applications, as well as the caveats of MPI

The Classic Cluster

- You ***now*** also need to manage the ***lifecycle of the cluster***
  - Add / remove nodes
  - Reconfigure queues / policies
- ***With constraints:***
  - Running jobs have been computing for hours
  - Queues already hold hundreds or thousands of Pending jobs
- RM ***must*** handle it gracefully: e.g. Torque/MAUI can independently restart scheduler or restart queue, no effect on running jobs, no effect on queued jobs.

Take-away commodity 1:

- RM to co-ordinate resource utilisation
- Computation accesses central NFS storage for data
- Dedicated network for MPI computation
- ***And*** planning + technical management of the Cluster Lifecycle

The Lifecycle of SW in the cluster: what does it imply?

The Lifecycle:

compile/deploy + set env var before running + run + unset env

Cluster Software Management

- Cluster software: ***one-off sw*** developed by a single user for personal needs + ***stock sw*** used by the wider user community.

- One-off sw:
  - More practical for users to run it from their ***home directory***
  - ***Implies having to mount user homes from all compute nodes!***
  - Compiling may still require availability of libraries and modules installed and available to wider community

Cluster Software Management

- ***Stock SW: Physical deployment and access***
  - One approach: have rsync/scripts/ssh to each node and copy the compiled sw/library or run the standard package manger of the distribution for the requested package.
  - Quickly becomes problematic to maintain: has the copy been successful on all nodes? What if a node is reinstalled? How do I know the state of each node?
  - Another common approach: ***have all sw installed in a shared directory***, ***mounted from all compute nodes***: allows access and use of widely available scientific packages and libraries, perl/python and modules, etc.

- ***Stock SW: Runtime environment***
  - Setting the runtime environment possibly ***per user*** and ***per application***
  - *MODULE* sw allows loading and unloading the env variables as needed.

Cluster Software Management

- ***Stock SW: Compilation of Source code***
  - May require: other libraries of different versions + specific HW + to be root
  - One approach: have a separate host configured with the required sw for compiling
  - But keeping in sync with prod is error prone and overhead
  - ***Another approach: use directly one of the compute nodes since it is already configured***

Take-away commodity 2:

- There is a Central Repository Architecture:
  - Mount SW Directory through NFS from all Compute Nodes
  - Mount User Home through NFS from all Compute Nodes

- Compiling chores:
  - -Very cumbersome due to library, version and HW, dependencies
  - Needs to be planned.

- User needs for workable solution / procedure that fits enough with work habits

How do we support users?

Documentation and user support

- Add a ***wiki*** that users can use for ***self help***
- Add a ***ticketing system***
- Be ***transparent about specs*** to users to see what the cluster can do
- Prepare a ***mailing list*** for sending out messages
- All this implies:
  - Mailman
  - Plone
  - RT
  - Make it all HA or decide on Backup policy and restore procedure!

Take-away commodity 3:

- When delivering a commodity cluster you also need to deliver the user support infrastructure

How to physically deliver the previous three objectives?

To deploy and configure the SW and HW that implements the Classic Cluster Architecture

Deployment infrastructure

- First deployment may be small so may be carried out manually a node at a time, but:
  - Usual to add Compute Nodes in batches throughout the years
  - The initial install may be very large
  - Some nodes may need reinstalling (HW failure or Upgrade)
- A manual approach is **error prone** and takes a long time
- Needed: an infrastructure to **automate** the **node deployment + cluster stack** on top.
- Engineering of solution must care for these two concerns.

Deployment infrastructure

- It involves:
  - OS deployment + configuration (hostanme, IP, DNS, time, routing, mount points, specific communication protocols)
  - SW deployment + configuration
  - Resource Manager deployment + configuration + monitoring
- ***Decide*** if nodes must reach internet or not.
- ***Decide*** on the names + IP addresses to use.

Deployment infrastructure

How is it engineered:

- Then setup DHCP + DNS server: organisational effort to collect MAC addresses
- Use PXEBoot protocol/server to netboot hosts ***without*** OS,
- Configure the hosts for netboot beforehand (manually done on each one!)
- Configure OS installation software i.e. Kickstart, prepare beforehand all the configuration and deployment files as requested.
- Prepare images of the OS
- Probably need a TFTP or NFS storage to deliver all these files
- Amongst the post-install steps also include installation of monitoring sw such as Ganglia
- This implies a central repository from which all packages are downloaded from: install a local repository/mirror.
- This also means the NFS/TFTP server needs to be accessible from all nodes
- ***An option is to concentrate all these functionalities in the master node***

Take-away commodity 4:

Deployment infrastructure is unavoidable, given the size of the task.

It is highly complicated, much more than the actual cluster!

All of this is needed to deliver an implementation of Classic Cluster Architecture

Going beyond

- Recap of Classic Architecture:
  - MPI + Infiniband HW + NFS storage for data processing
  - SW deployment architecture with shared NFS storage
  - Operating Model: Compute Nodes mount the directory and SW run from there

What are the limits and alternatives?

Going beyond

Considering Application Lifecycle:

- Architecture perfect for one-off kind of codes/ usages
- For stock applications used more like tools for specific ***day-to-day work*** i.e. Quantum Espresso, regCM, Turbolence, etc.
  - Classic architecture still applies.
  - But a ***time-constrain*** for delivering of results may push to other approaches.

Going beyond

- **“Scaling up” the Architecture**
  - **Data processing** from network may **stress** the Storage
  - Consider not only storage capacity **but especially IOPS consumed**
- e.g. Quantum Espresso with multiIO enabled: all processes will write data concurrently! It will consume all your IOPS! The amount of data is small ~ 10GB but IOPS will be exhausted easily.
- All users of that storage will be affected: if **sharing homes**, login will be slow; if run other sw such as VMs, they will be slow; if downloading stuff from the Storage, it will be slowed down even if plenty of network bandwidth.

To address it, **scale vertically**:

- Upgrade HW: disk technology (rpm+protocol+SSD), controller
- Review RAID policy (2x1TB Disk in Mirror <> 8x 250 GB RAID 1+0)
- Consider Parallel Filesystem (GPFS): concurrent access from multiple clients to different chunks of the same file
- Data Sharding: use separate storage for separate applications

Going beyond

- Generally for performance you need to leverage app specific data access/processing patters, and architect around that.
- For example:
  - HazelCast for memory caches
  - Hadoop+HFS
  - Cassandra

Going beyond

- **Memory Caches** based architectures
  - E.g. you have ***the same*** 1 TB data to be analysed for a different aspect
  - Classic Arch: network trip + read from local disk = stressed + slow
  - You could have 1 server with 1 TB RAM (about $80.000)
  - But if I tell you there are 10.000 clients per second accessing 500 KB chunks?

To address it, use **Memory Cache software**:

• Orcle Coherence, Hazelcast, etc.

• Build a cluster with 100 Nodes, each with 20 GB RAM

• The SW will manage a second copy of the data in case a node fails

•The cluster has special Access Nodes e.g. 10x, each serving 1000 clients

• It scales horizontally and linearly: add more Access Nodes and serve more clients

Example:

• Big Pharma and Biotech use Open Source pattern recognition software, within one of the largest Oracle Coherence deployments of various TB RAM

Going beyond

- Hadoop and HFS based architectures
  - You may have large data sets in the order of 100 TB
  - You must process them within 24 hours
  - The network will be the bottleneck: even at 10Gbps it will take a little more than 24hr just for the data to travel once through the network.
  - The Classic Architecture will work, but most all time will be spent shuffling data around.

To address it, ***consider Hadoop+HFS***:

- Files are split into large chunks and spread over hundreds of nodes.
- The Scheduler and the filesystem HFS are tightly integrated
- The code that must be run on the data, is ***executed*** where the data is
- A catch is Hadoop has a special processing primitive, that your needs must fit.
- So data no longer goes around, but rather the code goes where the data is.

Going beyond

- **Cassandra and NoSQL** based architectures
  - You need to process Astronomy data stored in a SQL database
  - It's a huge DB with a few TB of highly normalised data
  - Direct processing within the DB with Stored Procedures will be unworkable: DBs scale vertically
  - You could dump the DB into file format, and feed it to your Classic cluster
  - The highly normalised state will lead to data expansion once it needs to be processed

To address it, consider **Cassandra and NoSQL**:

• Store the data in a Cassandra cluster

• You will need to reconsider how to structure it, as a certain degree of denormalisation is needed

• Once the cluster is in place: it scales horizontally and linearly

• Add more nodes until your storage needs are met

• If 10 nodes deliver data to 1000 concurrent clients, 20 nodes will serve 2000 clients.

• **Essentially you do away with the File Processing paradigm, in favour of direct Data Processing**

Thanks you!

- Questions?