Fenrir: Cluster Middleware

Siddhant Agarwal Divyang Mittal Shrey Shrivastava

Ritik Kumar Kanishk Singh

March 2021

1 User level features

These are the features that the user will see, i.e. will be perceived by the user only. Most of the internal system related features and architectures will be discussed in the subsequent sections. For a user, a cluster should look like a single machine where he/she submits a set of jobs (tasks) and gets the results. He does not really care about where the job is running. Often though, the user may be concerned with how fast the job completes. Cluster middlewares often give some flexibility to users to prioritize their jobs. Here are some features that we will be providing to the users,

1. The user can submit a job interactively or as a batch where a job will consists of multiple tasks.
2. The tasks can be both serial or parallel.
3. User can specify the minimum requirements (memory, cpu cores, gpu memory etc) while submitting a job.
4. User will be a part of a specific user group which will have a quota of the number of jobs that can be submitted simultaneously and a quota on the resources. Moreover, the group will also define the priority of the jobs, i.e., some user groups can have a higher priority over the others.
5. User can be a part of multiple groups, but it has to then mention the group from which it is submitting the job.
6. User can kill any job that it had sent, but it cannot modify the job properties (minimum requirements, group etc).
7. User can check the job status (the amount of memory or compute resources it has been allocated currently).
8. All files will be stored in an external file system (which itself will be a distributed structure like NAS). So, the user can anytime access the output logs while the file is running.

9. User can submit jobs only in the master node.
10. User can access the file system from the master node itself. All the files related to the job must be present in the external (or global) file system.
11. A user cannot see (and definitely cannot kill) the jobs of any other user.

2 Administrative features

A cluster is generally managed by an administrator. The admin definitely should have privileges to manage the system resources well. The privileges that an administrator will have apart from the general ones as a user are:

1. The admin can view and kill the jobs of any user.
2. The admin can view the status of all the compute nodes and can forcefully migrate jobs from one compute node to the other.
3. The admin checks for any misbehaving jobs and can kill them.
4. The admin can create and assign groups to the user. The admin decides on the quota for different groups.

3 System design features

We will now list the various design features that will be required to implement the cluster middleware. These features can be be broken down to features related to general architecture, job scheduling and fault tolerance. Here we have written job scheduling but this includes everything from the scheduling primitives to load balancing.

3.1 General architecture

1. The compute nodes cannot be accessed externally. Infact, the user does not know anything about the compute nodes.
2. The middleware can handle compute nodes with heterogeneous hardware resources but they should be running the same OS.
3. Compute nodes are connected in a completely connected topology.
4. A database which is stored in the external file system holds the records of all the users and groups.
5. Compute nodes can communicate between each other. An interface will be provided for this. This will enable the user to run parallel jobs on several nodes.

3.2 Job scheduling and load balancing

1. As already mentioned, jobs can have a minimum resource requirement as determined by the user. Moreover, the group will also be mentioned. If the user does not mention any minimum requirement, a default value (independent for a group) will be chosen.
2. The master node determines the maximum resources based on the job priority, type of job, user group etc.
3. Job priority will be determined by the user group and the job wait time. It will be ensured that a job does not wait infinitely, i.e. there will be no starvation.
4. The master node can preempt any job. It can also migrate a job from one set of compute nodes to the other.
5. A higher priority job can preempt a lower priority job.
6. The master perform dynamic load balancing. It will regularly check the status of the compute nodes and if required, will migrate jobs from some compute nodes to the other.

3.3 Fault Tolerance

1. Two master nodes (active - active).
2. File system will itself implement replication among its nodes to tolerate faults. This will although be abstracted from the working of the middleware.
3. If a compute node crashes, the master node will schedule all the jobs in that compute node to all other compute nodes.
4. The master node will take periodic checkpoints of the compute nodes and will log the transactions between checkpoints.
5. The nodes will send a heartbeat message at regular intervals to the master node.