Fundamentals of Building an HPC Cluster

The King in Alice in Wonderland said it best, “Begin at the beginning ….” The general goal of HPC is either to run applications faster or to run problems that can’t or won’t run on a single server. To do this, you need to run parallel applications across separate nodes. Although you could use a single node and then create two VMs, it’s important to understand how applications run across physically different servers and how you administer a system of disparate physical hardware.

With this goal in mind, you can make some reasonable assumptions about the HPC system. If you are interested in parallel computing using multiple nodes, you need at least two separate systems (nodes), each with its own operating system (OS). To keep things running smoothly, the OS on both nodes should be identical. (Strictly speaking, it doesn’t have to be this way, but otherwise, it is very difficult to run and maintain.) If you install a package on node 1, then it needs to be installed on node 2 as well. This lessens a source of possible problems when you have to debug the system.

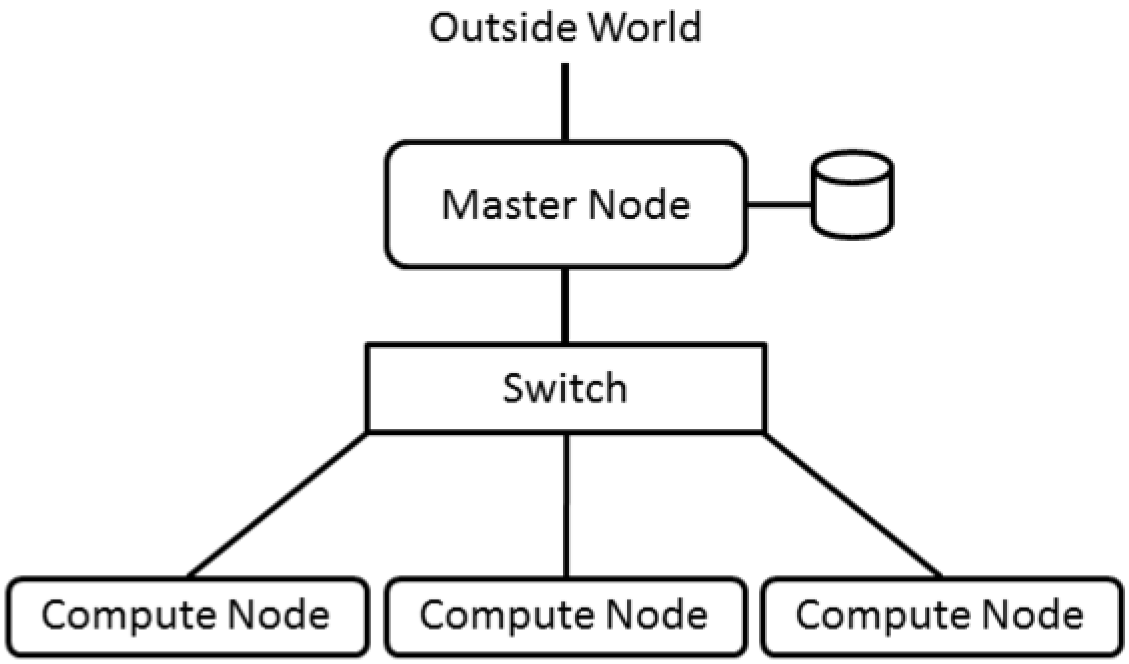
The second thing your cluster needs is a network to connect the nodes so they can communicate to share data, the state of the solution to the problem, and possibly even the instructions that need to be executed. The network can theoretically be anything that allows communication between nodes, but the easiest solution is Ethernet. In this article, I am initially going to consider a single network, but later I will consider more than one.

Storage in each node can be as simple as an SD card to hold the OS, the applications, and the data. In addition to some basic storage, and to make things a bit easier, I’ll create a shared filesystem from the master node to the other nodes in the cluster.

The most fundamental HPC architecture and software is pretty unassuming. Most distributions have the basic tools for making a cluster work and for administering the tools; however, you will most likely have to add the tools and libraries for the parallel applications (e.g., a message-passing interface [MPI] library or libraries, compilers, and any additional libraries needed by the application). Perhaps surprisingly, the other basic tools are almost always installed by default on an OS; however, before discussing the software, you need to understand the architecture of a cluster.

Architecture

The architecture of a cluster is pretty straightforward. You have some servers (nodes) that serve various roles in a cluster and that are connected by some sort of network. That’s all. It’s that simple. Typically, the nodes are as similar as possible, but they don’t have to be; however, I highly recommend that they be as similar as possible because it will make your life much easier. Figure 1 is a simple illustration of the basic architecture.



Almost always you have a node that serves the role of a “master node” (sometimes also called a “head node”). The master node is the “controller” node or “management” node for the cluster. It controls and performs the housekeeping for the cluster and many times is the login node for users to run applications. For smaller clusters, the master node can be used for computation as well as management, but as the cluster grows larger, the master node becomes specialized and is not used for computation.

Other nodes in the cluster fill the role of compute nodes, which describes their function. Typically compute nodes don’t do any cluster management functions; they just compute. Compute nodes are usually systems that run the bare minimum OS – meaning that unneeded daemons are turned off and unneeded packages are not installed – and have the bare minimum hardware.

As the cluster grows, other roles typically arise, requiring that nodes be added. For example, data servers can be added to the cluster. These nodes don’t run applications; rather, they store and serve data to the rest of the cluster. Additional nodes can provide data visualization capabilities within the cluster (usually remote visualization), or very large clusters might need nodes dedicated to monitoring the cluster or to logging in users to the cluster and running applications.

For a simple two-node cluster that you might use as your introduction to HPC, you would typically designate one master node and one compute node. However, because you only have two nodes, applications would most likely run on both – because why waste 50% of your nodes?

The network connecting the cluster nodes could be any networking technology, but the place to start is with wired Ethernet, which ranges from 100Mbps to 56Gbps; however, I’ll stick to the more common Fast Ethernet (100Mbps) or Gigabit Ethernet (1,000Mbps).

The network topology you use for clusters is important because it can have an effect on application performance. If you are just starting out, I would stick to the basics. A simple network layout has a single switch with all nodes plugged in to that switch. This setup, called a fat tree topology, has only one level and is simple and effective, particularly when building smaller systems.

As systems get larger, you can still stick to the fat tree topology, but you will likely have to have more levels of switches. If you re-use your existing switches, design the topology carefully so you don't introduce bottlenecks.

For smaller systems, Ethernet switches are pretty inexpensive, costing just a few dollars per port. Switches are going to be better than an Ethernet hub, but if all you have is a hub, then you can use it. Although hubs will limit performance, it won't stop the cluster from working.

Because you're interested in "high performance," you want to do everything possible to keep the cluster network from reducing performance. A common approach is to put the cluster on a private Ethernet network. The address space is unroutable, so the compute nodes will effectively be "hidden" from a routable network, allowing you to separate your cluster logically from a public network.

However, a good idea would be to log in to the cluster from a public network, and the way to do that when the cluster is on a private network is to add a second network interface controller (NIC) to the master node. This NIC will have a public IP address that allows you to log in to the cluster.

Only the master node should have the public IP address, because there is no reason for compute nodes to have two addresses. (You want them to be private.) For example, you can make the public address for the master node something like 72.x.x.x and the private address something like 10.x.x.x. The order of the network interfaces doesn't make a huge difference, but you have to pay attention to them when installing the OS.

You can give the master node two private addresses if you are behind a network address translator (NAT). This configuration is very common in home routers, which are also NAT devices. For example, in my home network, I have an Internet router that is really a NAT. It converts packets from a private network, such as 192.168.x.x, to the address of the router (the Internet) and vice versa. My simple clusters have a master node with a public IP of 192.168.x.x, and they have a second NIC with an address of 10.x.x.x, which is the cluster's private network.

Another key feature of a basic cluster architecture is a shared directory across the nodes. Strictly speaking this isn't required, but without it, some MPI applications would not run. Therefore, it is a good idea simply to use a shared filesystem in the cluster. NFS is the easiest to use because both server and client are in the kernel, and the distribution should have the tools for configuring and monitoring NFS.

The classic NFS approach to a shared directory is to export a directory from the master node to the compute nodes. You can pick any directory you want to export, but many times, people just share /home from the master node, although sometimes they will also export a new directory, such as /shared. The compute nodes also mount the shared directory as /home. Therefore, if anything in /home is local to each node, it won't be accessible.

Of course, you can get much fancier and more complicated, and you might have good reasons to do so, but in general you should adopt the KISS (Keep It Simple Silly) approach. Simple means it is easier to debug problems. Simple also means it's easier to reconfigure the cluster if you want (or need). With the architecture established, I'll turn to the software you'll need.