Building a cluster with two computers

Student:
Kastriot **Kadriu**

Supervisors:
Dimitar **Borisov**
Dimitar **Pilev**

Table of contents

Introduction

For the last decades, the way we access computer systems has been constantly changing due to the increasing demands of hardware and software use. As such, there has appeared the need for more rational use of those systems. The concept of rational use of computer systems is a matter which has been being worked on for ages. There is a linear proportion between resource-needs and human needs. As the single-core processors reached their limits, we had to move on with dual-cores and while the same happened to those, quad-core processors were introduced. Nowadays, the latter of processors mentioned can serve individual-use just well, but the industries are working with terabytes of data which these type of processors can't handle (properly). For these reasons, enterprises have to get creative in the way they manage the data by connecting computer systems with each other thus building clusters.

In the following paper, we will discuss the concept of resource-sharing between two (or more) computer systems which creates, what is called, a cluster.

What is a cluster?

A cluster is a set of connected computers (which are referred to as *nodes*) which is programmed to solve computational problems. Two of the most popular ways a cluster can function is by Distributed computing or Parallel computing. While those two methods are different, they also overlap each other thus creating no clear distinction between each other. In Distributed computing, a problem is divided into tasks, each of which is solved by specific nodes. In this type of computing, each node has its own local memory and they communicate each other by message passing. In Parallel computing, a problem is carried out simultaneously. The nodes here share a memory which they use to exchange information.

Building a cluster

While building a cluster, there are two issues that need to be overlooked. The first one is the hardware selection and the other being interconnection design.

Hardware selection

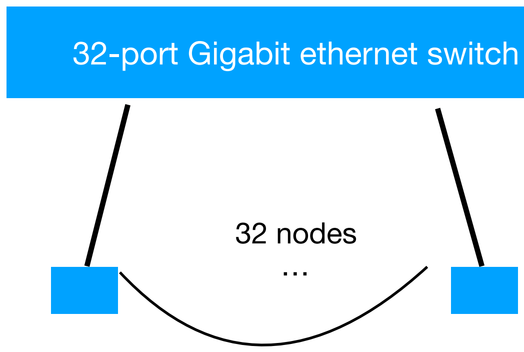
While selecting nodes, it is important to look at their specifications. Processors and memory have a big role in the cluster performance, so the main investment should go there. There's no need for any output devices, and no fancy graphic card is needed either. While the base principle is to get the better components possible, price is usually a huge factor when building a cluster meaning sometimes one should settle for a lower-end component.

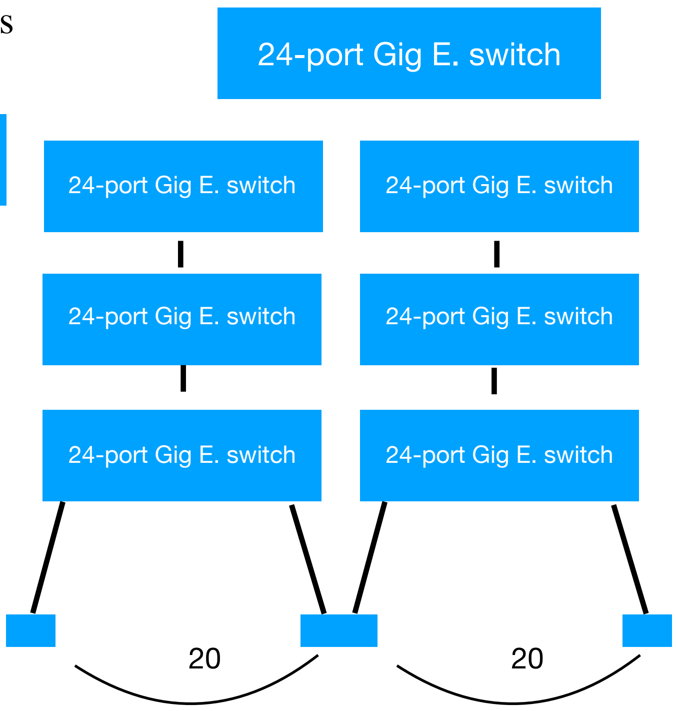
Interconnection design

Another issue is interconnection design. Before starting designing the actual topology of the nodes, networking type should be used. Some of the networking type can be: Fast Ethernet, Gigabit Ethernet, Myrinet, Infiband. The differences between those types of network are in bandwith, latency and price. The latter making a huge decision in which type should be chosen.
Another thing that should be decided is the ip addresses that are going to be used. A good practice is to use IP ranges that are reserved for private networks such as 192.168.0.0-192.168.0.255. These addresses are permanently unassigned, not forwarded by Internet backbone routers and thus do not conflict with publicly addressable IP addresses.

So far, the network type and network addresses have been decided, however there still need to be a way for nodes to be connected with each other. For that, a selection between Hubs and Switches should be made. A hub distributes a signal to all nodes whereas a switch distributes the signal only to the destined node.
In cases where we have a cluster consisting of two nodes, an Ethernet cable can be used to connect those nodes.
Network hardware design can be done in different ways such as:

- One big switch
- Complicated switching technology
- Tree structure of hubs and switches





Case study: Building a database cluster using MYSQL

A database cluster is a cluster built to get maximum numbers of performance, throughput and storage. Usually, a database cluster is made of one (or more) main node(s) that manages the other nodes - called *management node*, the nodes to store data - *data nodes*, and the nodes in which SQL queries will be executed - *SQL nodes*.
For compatibility reasons, this case study was chosen to be made using MySQL Cluster technology, specifically NDB engine.
MySQL Cluster is a distributed database system with *data nodes* storing data and API nodes for query execution such as *mysqld*. It is managed by a special *management node*. MySQL Cluster is designed for applications dealing with big data and requiring high-availability. As a precondition, the number of data nodes in this cluster must be a multiple of two (2, 4, 6, 8).
Data is automatically partitioned across the data nodes, although the user can define its partitioning only *by key*. By default it uses a hash over the table's primary key to partition data.
While in a distributed system, the data can be fragmented horizontally, vertically or a mix of the latter two, in MySQL Cluster the data is fragmented only horizontally - called *sharding*.

Working environment

This study was made using two computers at my university lab. Those two computers were connected with a 32-port Gigabit Ethernet switch on my university ISP. While the specs are not as good (Intel Core 2 Duo CPU @ 2.93 ghz - 2gb ram) they can serve just as well for demonstration purposes. Both of these computers run Debian OS 9.0.

Installation

To install MySQL Cluster you need to download its package from https://dev.mysql.com and install it as you would install any other Debian packages. On the internet there are a lot of resources which can tell you how to install the mysql packages.

Debian 9 comes with mariadb pre-installed. In order to install (and use) mysql, mariadb needs to be uninstalled first. That can be done by deleting the mariadb repository.

Setting up the servers

Our cluster environment looks like this:

Management Node	193.30.231.245
SQL Node	
Data node 1	193.30.231.196
Data node 2	

The management node should have a **config.ini** file in which all configuration parameters are written. To do that we first have to create the directory and then the file:

```
shell > mkdir /var/lib/mysql-cluster
shell > cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our setup, the **config.ini** should look as follows:

```
[ndbd default]
NoOfReplicas=2  #Number of replicas
DataMemory=80M #Memory allocated for data storage
IndexMemory=19M #Memory allocated for index storage
[ndb_mgmd]
HostName=193.30.231.245 #Hostname or IP of the management node
DataDir=/var/lib/mysql-cluster #Directory to save the data - Must be created if it doesn't
exist
[ndbd]
Hostname=193.30.231… #Hostname or IP of the date node
NodeId=2 # Node ID for reference
DataDir=/usr/local/mysql/data/ndbd-1 #Directory to save the data - Must be created if it
doesn't exist
```

```
ServerPort:50501
[ndbd]
Hostname=193.30.231…
NodeId=3
DataDir=/usr/local/mysql/data/ndbd-1
ServerPort:50502
[mysqld]
#SQL Node Options
NodeId=2
Hostname:193.30.231.245
```

Next we need to configure SQL and data nodes. On all the machines containing those nodes a file named **my.cnf** needs to be created. In our case, SQL node is located in the same node as the management node, and both of data nodes are in the same location.

```
shell > vi /etc/my.cnf
```

The **my.cnf** file should look like this:

```
[mysqld]
#Options for mysqld process
ndbcluster #run NDB engine
[mysql_cluster]
#Options for NDB Cluster processes:
ndb-connectstring=193.30.231.245        # location of management server
```

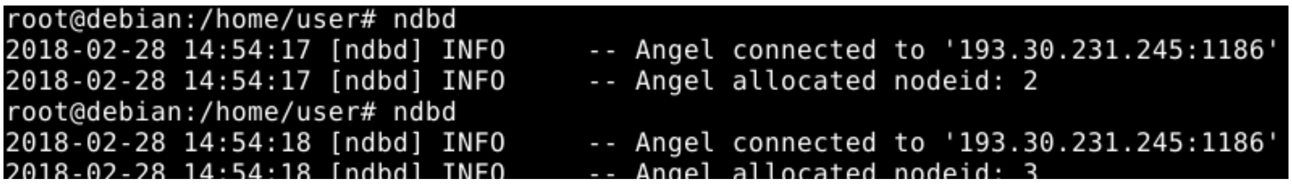
If all goes well, we should be able to start our NDB cluster. But to check if all *went* well, start *mysql* and run the *SHOW ENGINES;* command. The field for NDB engine should have values: DEFAULT or YES.
Starting the NDB cluster for the first time should be as follows.
Run the **ndb_mgmd** command with the following parameters:

```
shell > ndb_mgmd --initial / -f /var/lib/mysql-cluster/config.ini / --config-dir=/var/lib/mysql-
cluster/
```

The *initial* parameter can be replaced with *reload*. Both of their purpose is to delete the cache and force ndb_mgmd to read the updated **config.ini** file.

Start the data nodes by running the following command in the respective machine:

```
shell > ndbd
```



```
root@debian:/home/user# ndbd
2018-02-28 14:54:17 [ndbd] INFO     -- Angel connected to '193.30.231.245:1186'
2018-02-28 14:54:17 [ndbd] INFO     -- Angel allocated nodeid: 2
root@debian:/home/user# ndbd
2018-02-28 14:54:18 [ndbd] INFO     -- Angel connected to '193.30.231.245:1186'
2018-02-28 14:54:18 [ndbd] INFO     -- Angel allocated nodeid: 3
```

And then finally, on the *mysqld* machine start the SQL service:

```
shell > mysqld
```

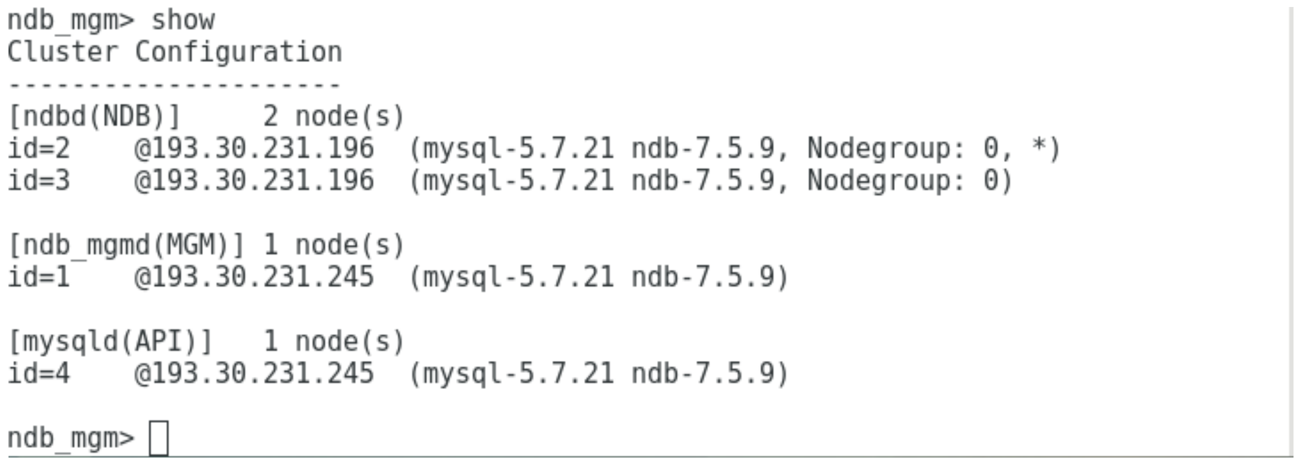
If mysqld is ran before starting the ndbd nodes, it will automatically shutdown.

Next run the following commands:

```
shell > ndb_mgm
```

```
ndb_mgm > show
```

This is what you should be getting:



```
ndb_mgm> show
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @193.30.231.196  (mysql-5.7.21 ndb-7.5.9, Nodegroup: 0, *)
id=3    @193.30.231.196  (mysql-5.7.21 ndb-7.5.9, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @193.30.231.245  (mysql-5.7.21 ndb-7.5.9)

[mysqld(API)]   1 node(s)
id=4    @193.30.231.245  (mysql-5.7.21 ndb-7.5.9)

ndb_mgm> 
```

If you get the results above that means the connection between hosts has been done successfully.
Another step is to convert mysql tables to ‘engine=ndbcluster’:

```
use mysql;
ALTER TABLE mysql.user ENGINE=NDBCLUSTER;
ALTER TABLE mysql.db ENGINE=NDBCLUSTER;
ALTER TABLE mysql.host ENGINE=NDBCLUSTER;
ALTER TABLE mysql.tables priv ENGINE=NDBCLUSTER;
 ALTER TABLE mysql.columns priv ENGINE=NDBCLUSTER;
ALTER TABLE mysql.func ENGINE=NDBCLUSTER;
ALTER TABLE mysql.proc ENGINE=NDBCLUSTER;
ALTER TABLE mysql.procs priv ENGINE=NDBCLUSTER;
SET GLOBAL event scheduler = 1;
CREATE EVENT ‘mysql‘.‘flush priv tables‘ ON SCHEDULE EVERY 30 second ON
COMPLETION
PRESERVE DO FLUSH PRIVILEGES;
```

What could go wrong

NDB Cluster by default uses port 2202 for connection with data nodes. Since both of data nodes are located in the same place, we will have trouble connecting the second node. For that we need to find free ports. To open connections in all port is dangerous, but instead we can open specific ports to receive connection from our management node. That can be done by using iptables and the following command in our data nodes location:

```
shell > iptables -I INPUT -p tcp --dport 50501:50502 -s 192.30.231.0/24 -j ACCEPT
```

To save our changes, we need to run the following command as root:

```
shell > service iptables save
```

And that’s why the ndbd section in our **config.ini** file should have the *ServerPort* parameter.

We should create two different data directories or else our nodes will be stuck in the *starting* phase.
NDB Cluster by default uses port 1106 for connection with management node. Another issue that can appear is the second node not connecting (in the same machine as in our case). Since two instances from the same machine want to connect with the same destination, the instance that makes the connection last will be refused. To treat this issue is to do the same thing that we did in our *data nodes* machine and that is by opening new ports depending on our needs in our management node machine.
MySQL/d not connecting - Sometimes, you might have troubles connecting mysqld or starting mysql. This is mostly an issue of not finding the sockets. To fix that you need to find the sockets and in the **my.cnf** file in the machine containing SQL node you add those parameters:

```
socket=/run/systemd/journal/socket
```

And the respective file looks like this:

```
[mysqld]
ndbcluster
ndb-connectstring=193.30.231.245
datadir=/var/lib/mysql
user=root
socket=/run/systemd/journal/socket
[mysql_cluster]
ndb-connectstring=193.30.231.245
```

In our case, we also used the *user* parameter to allow mysqld to be ran on root since mysql doesn't naturally allow the deamon service to be ran on root privileges but we had to do that because that's where we installed it.

Working with data in this cluster

If we want our tables to use the NDB Cluster all we need to do is add the "engine=ndbcluster" to our query. This will ensure that the data entered in this table will be partitioned across our data nodes. Also, we can change the tables that already exist to use the NDB Cluster with *ALTER TABLE table_name engine=ndbcluster.*

Conclusion

This study was made for demonstration purposes. While configuring the cluster wasn't easy, once you get familiar to the environment and the parameters it won't be hard either. People use NDB Cluster for its ease in maintaining and administrating. It does have its limitation such as a poor performance in massive parallel write requests. However, a database designer can always get creative and solve those limitations by building special architecture setups such as replication, partitioning, etc.
Nevertheless, this is a great alternative for database clustering especially for someone who has little to no experience.

References

1. MySQL NDB Cluster official documentation. https://dev.mysql.com/doc/refman/5.6/en/mysql-cluster.html
2. Raith, Michael. MySQL Cluster – Evaluation and Tests