Supporting a Hadoop Cluster with Unused PCs

Floriane Le Floch - Arthur Busser - Paul Dennetiere

December 17, 2016

Summary

Short riddle

This report is written in a font we all love to hate: Comic Sans MS. If you wish to read it as is, you are free to carry on. On the other hand, if you wish to read it in a much nicer font you will need to solve this short riddle:

You have a wall with 3141592653 doors that are initially closed and numbered from 1 to 3141592653. You will make 3141592653 passes in front of those doors. On each pass, you toggle (open if closed, close if open) certain doors. On the first pass, you toggle each door (#1, #2, #3, ...). On the second pass, you toggle every other door (#2, #4, #6, ...). On the third pass, you toggle every third door (#3, #6, #9, ...). So on and so forth until you reach the final pass where you only toggle the last door (#3141592653).

Once all your passes are complete, how many doors are open?

To obtain this report in a nicer font, visit the following link:

http://www.arthurbusser.com/hadoop/?answer=12345

Replace 12345 with your answer.

Note 1: You will not need brute force. In most programming languages, finding the answer only requires calling one or two functions.

Note 2: If you give up, you may submit 'justgivemethereport' as an answer.

Part 1

Introduction

1.1 Personal computers

In 2008, approximately 1 billion personal computers were in use worldwide. Today, that number stands above 2 billion[1]. In the United States of America, Japan, Germany, the United Kingdom, France, and Italy, the number of PCs gets closer to the number of inhabitants every year[2].

In 2016, more than half (50.1%) of the world's population was an Internet user[3]. In this Information Age, most computers are connected to each other and can easily communicate. Countless companies rely on computers for communication, marketing, accounting, storage, documents and reports, education, and research.

As things stand, personal computers used for work and/or personal activities add up to an immense amount of processing power and that amount is growing continuously.

1.2 The problem

Very few people use the full computing capabilities of their PCs. Even then, there are long periods of time every day when PCs are left unused. In many companies, almost every employee is provided with a PC. At night, desktop computers - and laptops too - are often left on, staying idle until their user returns in the morning. During this time, those PCs could be doing something.

[1] Source: Forrester Research

[2] Source: Computer Industry Almanac Inc.

[3] Source: http://www.internetworldstats.com/

The main problem with personal computers is that they are personal. The average person will never use the full potential of their PC, simply because most of the time they will not be using it. This results in a huge amount of processing power going to waste. If only the computer's capabilities could be shared somehow...

1.3 Our idea

As we stated above, most computers are connected and can easily communicate with each other. Many professionals have used this to their advantage, organizing computers into clusters that could tackle computation problems as a whole.

Our idea is to create a tool that, once installed on a personal computer, would wait for the PC to be unused to connect it to a remote computer cluster. The cluster could then use the new node's processing power to advance in its current calculations. When the PC would once again need to be used, it would disconnect from the cluster and continue as it normally does.

Part 2

Our solution: Setting up a DataNode on an unused computer

We want to make unused computers available for a Hadoop Cluster. In order to do that, we want to check at regular intervals if a computer is used or not (i.e. is in sleep mode). Our goal is to dynamically add sleeping computer to the cluster, in order to make more resources available.

In order to achieve this, we will have to ensure several things. This part will deal with the general structure, and what this structure induces for the cluster and added computers.

First we will have to make the computer we want to add able to communicate with the Master (NameNode). We have several options to achieve this:

- Virtualization
- Use only linux distributions
- Application containers

Only using linux distributions will greatly impact the interest of this project, so it is clearly not a solution. Virtualization does not seem good either, because of its need of resources, and the general weight of those solutions. That is why we will use an Application Container: Docker.

2.1 Docker

Docker defines itself as:

> "Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in."

Therefore by using Docker we will be able to use a linux application on every operating system encountered in the company. Moreover, by doing a snapshot of a running configuration, Docker allows us to deploy a solution very easily, and with a good velocity. Finally, the global strategy is to have a "Slave Image" for Docker: Every computer in the company will have Docker installed on it, and will start the Docker application every time the computer goes in sleep mode (we can easily do this through a C# script, running in background, checking the sleep mode, and launching the Docker application, or even using TaskLauncher included in Microsoft's OS).

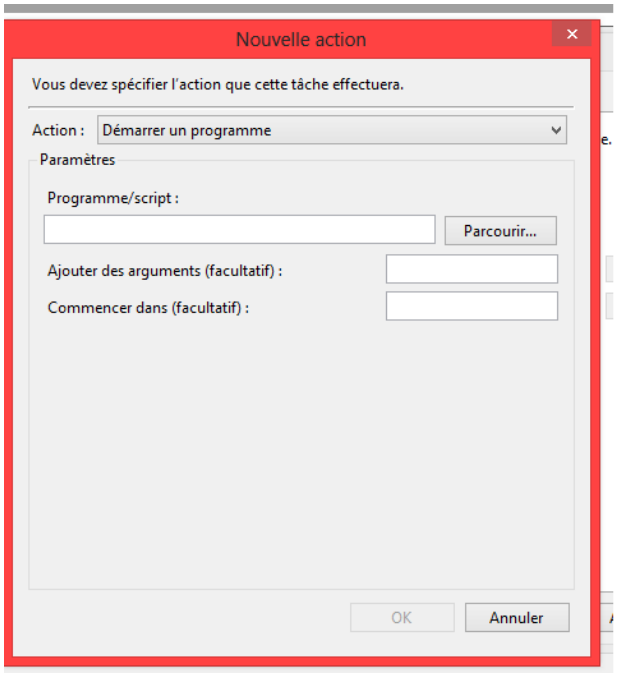

Figure 2.1: Screen capture of Microsoft's Task Launcher

2.2 Configuration

We supposed that a "classical Hadoop cluster" is already configured (meaning we do not install a full cluster). So our goal is to provide a Hadoop slave service on the computer, and to make it communicate with the Master.

We will first create an image of a running configuration. Such images are already available on Docker's Hub (Docker's image sharing website), but we will assume that we can not get such images: The first thing to do is to create a linux distribution image, a Debian for example. After that we install the Hadoop slave service on this Debian.

- Creating Hadoop user and password:

```
useradd hadoop
passwd hadoop
```

- Install Slave service on all slave by running on Master :

```
# su hadoop
$ cd /opt/hadoop
$ scp -r hadoop hadoop-slave-1:/opt/hadoop
$ scp -r hadoop hadoop-slave-2:/opt/hadoop
```

- Setup password-less connectivity from master to new slave.

```
mkdir -p $HOME/.ssh
chmod 700 $HOME/.ssh
ssh-keygen -t rsa -P '' -f $HOME/.ssh/id_rsa
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
chmod 644 $HOME/.ssh/authorized_keys
scp $HOME/.ssh/id_rsa.pub hadoop@IPPATH:/home/hadoop/
```

  Where IPPATH is the IP address of the new node. Do the same thing on the slaves, in order to exchange ssh keys between master and slaves.

- Then set a Hostname on new slaves in /etc/sysconfig/network:

```
NETWORKING=yes
HOSTNAME=slaveX.in
```

  Where X is the node Id (arbitrarily set, but unique)

- Run on every new slave the following, to make changes effective:

```
hostname slaveX.in
```

- Update /etc/hosts on all machines of the cluster with the following lines:

```
IPPATH slaveX.in slaveX
```

By using this configuration we are able to dynamically add a data node to an existing cluster. The only point of failure is on IP addresses: every node should have a different one, and use it in order to have a functional network. However, this IP address can be found on the host.

Finally, the new slave has to start HDFS by using this command:

```
./bin/hadoop-daemon.sh start datanode
```

This last command will have to be executed every time the Docker application is started.

2.3 Conclusion

In conclusion, docker is a tool (application container) that will allow us to run Hadoop on nearly every system, as well as allow us to have a configuration routine that practically does not differ from one machine to another. By porting this solution on every computer in the company, we can assume that every computer going in sleep mode, having a Docker image configured like above, will connect to the master, and be part of the Hadoop cluster.

However we can easily imagine that many difficulties will come up, and we will discuss them below.

Part 3

Foreseen difficulties

We expect to encounter several difficulties with our system. This part will enumerate them and provide a way to solve them.

3.1 Redundancy

When a computer is not used, it joins the cluster and some data is stored on it. But when the user comes back, this data will not be available for the cluster anymore. To face this availability problem, we have to prevent HDFS from storing all the data blocks on the temporary nodes. At least one copy, ideally more, of each data block should be stored on permanent nodes. Additional copies may be stored on temporary nodes at will.

To achieve this goal, we can modify the standard HDFS algorithm which chooses the nodes for the replication. The class in charge of this choice is : ReplicationTargetChooser (from org.apache.hadoop.hdfs.server.namenode). By default, the replica placement strategy is that if the writer is on a DataNode, the first replica is placed on the local machine, otherwise a random datanode. The second replica is placed on a DataNode that is on a different rack. The third replica is placed on a DataNode which is on the same rack as the first replca. If there are more replicas, they are spread across the rest of the racks. We can add two distinct groups to the regular parameters: temporary nodes and permanent nodes. Then during the replication, the algorithm chooses at least one target among the permanent nodes.

3.2 Blacklist

The JobTracker submit jobs to the different nodes through their TaskTracker. But if the TaskTracker fails too many times, it can be blacklisted and will not receive any jobs anymore (no impact on files replication). This is an issue because our computers can be used by a user at anytime and fail to execute its job. This node will then become unused for MapReduce and we lose the main advantage of this project. To solve this problem there are different possibilities.

Firstly we can manually remove a node from the blacklist :

```
hadoop job -unblacklist-tracker <hostname>
```

By executing this command every day for our temporary nodes written on the list, we can prevent them from being blacklisted for too long.

Secondly, we can prevent them to be blacklisted by safely removing them from the cluster. Usually in a company, most of the employees arrive at a predictable hour. We can plan to remove the temporary nodes a little before that time thanks to the decommissioning feature. This can also allow us to make sure that no data is lost because of a sudden shut down of the Docker container. To do that we need a script to execute the following steps dynamically:

- We have to add our nodes to the exclude group in the conf/hadoop-site.xml file. They will not be permitted to connect to the NameNode anymore. They will have enough time to finish their tasks and to be replicated on machines which are scheduled to remain active.
- Force configuration reload to make sure that HDFS is aware of the changes by running:

  ```
  bin/hadoop dfsadmin -refreshNodes
  ```

- Removing the temporary nodes from the exclude list so they can connect to the NameNode the next time.

This solution is efficient only if we can predict the time when computers will be used. We need to make sure than HDFS can handle the loss of a huge number of nodes simultaneously.