Newsreader virtual machine

Zuhaitz Beloki and Kike Fernández and Aitor Soroa
a.soroa@ehu.es

November 3, 2014

Contents

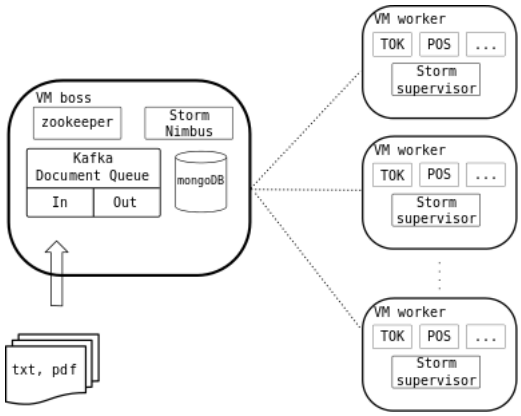

Figure 1: Main architecture of the NWR distributed pipeline

1 NewsReader architecture for distributed NLP processing

This document describes the distributed pipeline implemented within the NewsReader project. The pipeline follows an streaming computing architecture, where the computation is not limited to any timeframe. Instead, the pipeline is always waiting to the arrival of new documents, and one such new document arrives the NLP processing starts.

Figure 1 shows a general overview of the NewsReader distributed pipeline. The processing NLP modules and the software infrastructure for performing parallel and distributed streaming processing of documents is packed into virtual machines (VM). Specifically, we distinguish two types of VM into the cluster:

- One "boss" node, which is the main entry of the documents and which supervises the processing.
- Several "worker" nodes, which actually perform the NLP processing.

The boss node is the entry point for input documents. It contains a document queue where the input documents are stored and wait until they enter the pipeline. It also supervises the execution of the different stages of the processing pipeline. When the NLP processing is over, the boss node receives the processed NAF document into the output queue, and writes it in a specified directory.

The boss node implements a REST service that accept documents to the processing pipeline. For instance, the following command sends the document `doc.xml` to the processing pipeline[1]:

[1] In this document we will show many examples which are commands to execute from the CLI. If the command has to be executed from the host machine, we will use the '%' character in the beginning of the line. If the command is meant to be executed from the boss machine, we will use the '$' character.

```
% curl --form "file=@doc.xml" http://BOSSIP:80/cm_upload_text_file.php
```

Once the NLP processing is finished, the final NAF will be stored inside the boss VM, under the `docs/output` directory. In principle those documents should be sent to the KnowledgeStore (KS), but at the time being this last module which populates the KS with the processed NAFs is not ready.

The worker nodes are the nodes where the NLP modules are executed. There can be many worker nodes in a cluster, and therefore there can be many instances of the modules running in parallel.

This document is a reference guide that covers the most important steps to build a NewsReader architecture from scratch.

2 VM from scratch

We distribute a set of scripts with the aim of automatically create a fully working cluster for distributed NLP processing. We call these scripts "VM from scratch", as they create and configure the required virtual machines. The scripts are in a github repository, at this address:

`https://github.com/ixa-ehu/vmc-from-scratch`

The latest version of the documentation is also in the same repository, under the `doc` directory.

Creating a cluster using the scripts involves three main steps:

1. Create a basic cluster with the boss node and one worker node.
2. Make as many copies as required of the worker node.
3. Deploy the worker copies among different hosts and run the cluster.

The subsections below will describe how to use the provided scripts to achieve these goals in turn.

2.1 Create a basic cluster

The first step is to create the basic cluster using the `create_basic_cluster.pl` script. Please note that the VM images created by these scripts are huge (9 Gb memory), as they contain all the NLP modules of the Newsreader processing pipeline. Likewise, the machine to install those VMs need to have a large amount of RAM memory, as each VM, particularly the worker nodes, need circa 10Gb of memory to run.

The script is executed as follows[2]:

[2] In ubuntu machines, install the following packages before running this script

- libvirt-bin
- libguestfs-tools
- qemu-kvm

besides, you'll need to run

```
% sudo update-guestfs-appliance
```

```
% sudo create_basic_cluster.pl --boss-ip 192.168.122.111 --boss-name bossvm \\
  --worker-ip 192.168.122.112 --worker-name workervm1
```

The scripts needs four parameters:

- `-boss-ip` (**required**) the IP address of the boss VM[3].
- `-boss-name` (**required**) the hostname of the boss VM
- `-worker-ip` (**required**) the IP address of the first worker VM created by this script.
- `-worker-name` (**required**) the hostname of the worker VM.
- `-master-ip` (**optional**) the IP address of the IXA master machine. This machine is where the latest version of all NLP modules are.

Use the `-help` option to obtain information about the script's parameters (this option is present in all scripts described here).

The script will perform the following steps for creating the basic cluster:

- Copy an empty VM image from the IXA servers.
- Create the boss VM image using the empty VM image. In the example above, the hostname of the boss VM is "bossvm" ans its IP address is `192.168.122.111`.
- Create the worker VM image using the empty VM image. In the example above, the hostname of this worker VM is "workervm1" ans its IP address is `192.168.122.112`.
- The VMs have an unique user "`newsreader`" whose password if "`readNEWS89`". The user can `sudo` any root commands inside the VMs.

The next step is to turn on both VMs and start a synchronization process so that the required software (both the system software as well as the NLP modules) is properly installed in the newly created VMs.

The above script creates two XML definition files with the specification of the boss and worker virtual machines[4]. To (locally) run the VMs, run the following commands as root from the host machine:

```
% virsh create nodes/bossvm.xml
% virsh create nodes/workervm1.xml
```

Once the machines are up and running, we ssh into the boss VM.

```
% ssh newsreader@192.168.122.111
(pass: readNEWS89)
```

[3] In this example and in the examples below we will use private IP addresses for the boss and worker IPs. See section 2.5 for more information about IP addresses.

[4] See section 2.3 on the XML definitions of the VMs.

Once logged into the boss VM, wun the following:

```
$ sudo /root/init_system.sh
```

This command prepares installs all required software into both virtual machines (boss and worker). Specifically, the script performs the following steps:

- Install the required software into the boss VM. This involves the installation, among others, of the following packages:
  - *Zookeeper*
  - *Storm*
  - *Kafka*
  - *MongoDB*
  - *Puppet*
- Synchronize the boss VM with the IXA master VM to obtain all the NLP modules.
- Synchronize the worker VM with the boss VM to obtain a copy of all the NLP modules.
- Start the *puppet* agents on both the boss and worker VMs.

When the scripts finalizes the basic cluster is finally created and configured. The boss VM image is stored in the file `nodes/bossvm.img` and the worker image is `nodes/workervm1.img`. The cluster is already ready for processing documents, although it does not make much sense to use a cluster with a single worker node. Therefore, the next step is to create the required copies of the worker VMs.

2.2 Making copies of worker VMs

The next step is thus to copy the worker VM and create new worker nodes in the cluster. Before doing this, however, the boss and the worker VMs must be shut down. There are several ways to shut down the cluster. One possible way is, being into the boss VM, is to execute the following:

```
$ sudo pdsh -w workervm1 poweroff
$ sudo poweroff
```

The first command remotely shuts down the worker VM, whereas the second command shuts down the boss VM itself. In any case, check that the VMs are properly shut down by executing the `sudo virsh list` command from the host machine. The command should return an empty list of VMs:

```
% sudo virsh list
 Id    Name                           State
----------------------------------------------------
```

Once the cluster shut down, one can make as many copies as wanted of the worker nodes. The script `cp_worker.pl` accomplishes this task:

```
%  sudo ./cp_worker.pl --boss-img nodes/bossvm.img \
  --worker-img nodes/workervm1.img 192.168.122.102,workervm2
```

The script needs two parameters with the images of the boss VM and the first created worker VM (`workervm1` in out example). Then, the script accepts a space separated list of `IP,hostname` pairs. The above example just creates a copy of the worker VM, called `workervm2`, whose IP address will be `192.168.122.102`. Of course, it is very important that each VM receives a different hostname and IP address.

If we were to create more copies of the VM, we can just run the script again, the only requite being that the cluster VM can not be running. For instance, the following command will create two more worker nodes:

```
%  sudo ./cp_worker.pl  --boss-img nodes/bossvm.img \
  --worker-img nodes/workervm1.img \
  192.168.122.103,workervm3 192.168.122.104,workervm4
```

After this step we need to boot the cluster manually. Section 2.4 explains how to power on and power off the whole cluster at any time.

Creating copies of the worker without stopping the cluster

The requirement of having to stop the cluster before making copies of the workers may too hard to fulfill, in the case the cluster is running and processing documents. Therefore, we present is a method to create worker nodes without the need to stop the whole cluster. However, this method needs a copy if a worker image, and it is very important that this particular image is not running.

For creating a worker without stopping the cluster, execute the following:

```
%  sudo ./cp_worker.pl  --boss-img nodes/bossvm.img \
  --worker-img nodes/workervm1.img \
  192.168.122.105,workervm5
```

This will create the new `workervm5` node with IP address `192.168.122.105`. Now, we need to log into the boss node, and manually add this new address to the `/etc/hosts` file:

```
% sudo echo "192.168.122.105 workervm5" >> /etc/hosts
```

The next step is to stop the topology (see section 3) and run it again with the proper number of workers/CPUs.

2.3 Virtual machines XML definitions

The virtual machines specifications are defined in a XML document, one document per VM. These XML specification document is stored along with the VM image, i.e., if the image is in `nodes/workervm1.img`, the corresponding XML specification is `nodes/workervm1.xml`. You may want to manually edit the XML specification file to change some important things:

- The number of CPUs assigned to the VM (line 6, `/domain/vcpu` element). By default VMs are assigned 1 CPU but this can be easily changed. In fact, **we recommend to increase the number of CPUs per worker whenever possible**.
- The memory assigned to the VM (line 4, `/domain/memory` element). By default VMs are assigned 10Gb of RAM (needed mostly by the NED module).
- The exact location of the image files (line 22, `/domain/devices/disk/source` element). If you plan to deploy the worker VM on a different machine, you must change this value to the directory where the worker image file will be.

2.4 Booting and stopping the cluster

Boot the boss VM by running this into the host machine:

```
% sudo virsh create nodes/bossvm.xml
```

The host needs some time to start all the required services. To be completely sure that the machine is ready, run the following command inside the boss machine until it produces some output. First ssh into the boss VM, and then:

```
$ ps -aux | grep zoo | grep java
```

Once the boss is ready, turn on the worker nodes running the following inside the host machine:

```
% sudo virsh create nodes/workervm1.xml
% sudo virsh create nodes/workervm2.xml
...
```

The above commands will start the worker VMs in turn. Each VM needs some time to boot up (mostly due to the required time in loading the NED module). Section 2.6 explains how to know which machines are connected to the boss VM.

2.5 Deploying worker VMs into different host machines

In principle the worker VMs can be executed in any host machine, as far as the host has a 64 bit CPU. The main requirement is that the IP of the worker VM, as specified when creating the VM image, is accessible from within the boss VM. Likewise, the boss IP has to be accessible from the worker VM.

In the examples on this document all the IPs are private IPs, and therefore not accessible from outside the host machine. If you need to copy the worker VMs among several host machines, perhaps the best way is to use a bridged network configuration (not explained here). Alternatively, you can use iptables for allowing external access through ssh to the VM. In any case, talk with the IT people in your laboratory to correctly set-up the environment.

2.6 Knowing which machines are connected to the boss VM

When the cluster is up and running, we can consult any time which worker machines are connected to the boss. This is easily done by querying the zookeeper server on the boss VM. From inside the boss VM, run the command line "elinks" web browser:

```
$ elinks localhost:8080
```

Another option is to connect to the boss VM from the host machine, by putting the address `http://ip_of_boss:8080`. In the running example used in this document the boss IP is 192.168.122.111, we should put the address `http://192.168.122.111:8080` into the web browser.

3 Running the topology

The steps described in section 2 will create a cluster that is ready to start the NLP processing. For this, we need first to load a topology into the cluster. The topology is an abstraction that describes processing steps each document passes through. In principle the cluster can run any topology, the only requisite being that the NLP modules described in the topology are installed in the worker VMs.

Topologies are declaratively described in an XML document. There are already some topologies in the bossvm `opt/topologies/specs` directory. In this example, we will run the topology described in `opt/topologies/specs/test.xml`.

Apart from the topology definition, running the topology requires knowing the number of worker nodes in the topology, and the number of CPUs used. In the running example of this document we are using three workers. Suppose we also use two CPUs per worker[5]. So, the total of CPUs assigned to worker nodes is 6. This parameter is very important because it fixes the maximum

[5]Section 2.3 describes how to change the number of CPUs assigned to a node

parallelization we can achieve. If we have 6 CPUs working on our cluster, it makes little sense to have more than 6 copies of any NLP module running in parallel.

Inside the boss VM, run the following to run the topology:

```
$ opt/sbin/run_topology.sh -p 6 -s opt/topologies/specs/test.xml
```

This will load the topology and, as a consequence, the cluster will be ready to accept and process documents.

The script also accepts another optional parameter to specify the main configuration of the topology, the default value being `opt/etc/topology/topology.conf`. This configuration file is seldom changed within the newsreader project, and specifies values such as the port and addresses of various services such as zookeeper, kafka, mongoDB, etc. However, there is one parameter that need to be checked. The "host" values (`zookeeper.host`, `kafka.host`, `mongodb.host`) need to point to the hostname of the boss node (by default, `bossvm`).

3.1 Topology XML specification

The topology executed by the cluster is declaratively defined in an XML document. Here is an excerpt of a small topology:

```
<topology>
  <cluster componentsBaseDir="/home/newsreader/components"/>
  <module name="EHU-tok" runPath="EHU-tok/run.sh"
          input="raw" output="text"
          procTime="10"/>
  <module name="EHU-pos" runPath="EHU-pos/run.sh"
          input="text" output="terms"
          procTime="15" source="EHU-tok"/>
  <module name="EHU-nerc" runPath="EHU-nerc/run.sh"
          input="terms" output="entities"
          procTime="75" source="EHU-pos"/>
</topology>
```

The `<cluster>` element specifies the base directory of the NLP modules. Each module is described by a `<module>` element, whose attributes are the following:

- `name` (**required**): the name of the module.
- `runPath` (**required**): the path relative to `componentsBaseDir` where the module resides.
- `input` (**required**): a comma separated list of NAF layers required by the module as input.
- `output` (**required**): a comma separated list of NAF layers produced by the module.

- `source` (optional): the previous module in the pipeline. If absent, the attribute will get the value of the immediately preceding it according to the XML tree. If the module is the first node in the XML tree, and the source attribute is absent, the attribute gets no value at all.
- `procTime` (optional): the percentage of time this particular module uses when processing a document.
- `numExec` (optional): the number of instances of the module that will run in parallel.

The topology is defined by looking at the chains of `source` attributes among the modules, which have to define a directed acyclic graph. The module without parents is considered to be the first module in the pipeline. Likewise, the module without children is the last module in the pipeline. In principle the declaration may define non linear topologies, but at the moment its use is not properly tested, and thus not recommended.

The last two attributes are very important and require some tuning in order to obtain the maximum efficiency when processing the documents. There are two ways of calculating these factors. If the `numExec` parameter is present, this particular number of instances will be used. If `numExec` is absent, the system will use the `procTime` information, along with the number of nodes/CPUs to automatically calculate the number of instances of each module. The main idea is to execute as many copies as possible of the most resource demanding modules. Finally, if both `numExec` and `procTime` are absent, the system will run one single instance of the module.

4 Troubleshooting and tools

This section describes the most usual tasks performed by the cluster administrator to manage and monitorize the cluster, as well as when facing many problems that may occur. In principle the cluster should run flawlessly and without errors; it should just accept documents via the boss input port or inside the boss machine itself, and the documents should get processed, put in the output queue of the boss machine, and written to the `docs/output` in the boss machine. However, if something goes wrong there are some useful commands to help understanding what is happening.

4.1 Synchronizing the modules

The NLP modules are constantly being developed and upgraded into the IXA master machine. Obtaining the latest version of the modules from the master node into the cluster involves two steps. First, synchronize the boss node with the IXA master node; then, synchronize each worker node with the boss node.

For synchronizing the boss node with the IXA master, log in into the boss VM and run the following:

```
$ sudo ./update_nlp_components_boss.sh
```

For updating the workers there are two main options. One option is to log in into each of the workers and execute the following:

```
$ sudo ./update_nlp_components_worker.sh
```

The other is to update the worker nodes from inside the boss node, using the `pdsh` command. Being logged into the boss machine, run the following for each worker node:

```
$ sudo pdsh -w ip_of_worker /home/newsreader/update_nlp_components_worker.sh
```

4.2 Document queue

The boss VM contains two document queues, for input and output NAF documents, respectively. The input queue stores the documents to be processed until they are consumed by the NLP pipeline. The output queue stores the final NAF documents.

In principle, the usage of the queues is wholly automatic. The user sends a document to the input queue, and eventually the processed document is enqueued into the output queue. However, is important to have some basic notions about how these queues are implemented in case we want to consult the contents of a particular queue, or if we want to manually send or retrieve documents.

The queues are implemented using "Apache Kafka"[6]. Kafka is a messaging system that can be distributed among many machines and that provides fast and reliable access to the queue contents. Queues in Kafka are organized into *topics* and *partitions*. In NewsReader we have two topics, called "`input_queue`" and "`output_queue`", respectively, and each topic has one unique partition. Once a documents is sent to the queue, it stays there for a specific time frame, which by default is set to seven days. The input queue is attached to a consumer that continually pulls documents from the queue and sends them to the first stage of the processing pipeline. It also maintains an *offset* to the last consumed document in the queue.

Inside the boss VM there are some tools which can help with the management of the queues. Currently there exist the following tools:

`send_to_queue`

This tool is useful for feeding the input queue with documents from inside the boss VM. The usage is as follows:

```
$ opt/sbin/send_to_queue -f docs/input/input.xml
```

Please note that documents can be sent to the input queue from outside the boss VM, by using the following command:

[6] `http://kafka.apache.org/`

```
% curl --form "file=@input.xml" http://BOSSIP:80/cm_upload_text_file.php
```

this is meant to be the principal way to send documents into the NewsReader pipeline. The `send_to_queue` tool is an auxiliary tool for manually feeding the input queue with new documents.

ls_queue

Sometimes it is useful to know which documents are present in a queue (input or output), and the `ls_queue` command does exactly this. This is a typical example of usage (from inside the boss VM):

```
$ opt/sbin/ls_queue
doc18.txt  1209  9   input_queue
doc20.txt  1025  10  input_queue
doc23.txt  2020  11  input_queue
doc24.txt  347   12  input_queue
doc26.txt  789   13  input_queue
doc27.txt  257   14  input_queue
doc28.txt  1675  15  input_queue
doc29.txt  9837  16  input_queue
doc20.txt  721   17  input_queue
```

The tool displays 4 columns for each document in the queue, namely, the document name, the size, the offset number, and the queue name.

The above example showed the documents of the input queue waiting to be processed. The "`-a`" switch shows all the documents in the queue. Remember that Kafka keeps all documents in the queues for some time (by default, seven days). The following command shows all the documents in the queue.

```
$ opt/sbin/ls_queue -a
doc1.txt   762   1   input_queue
doc0.txt   4678  2   input_queue
doc3.txt   9876  3   input_queue
doc4.txt   189   4   input_queue
doc6.txt   875   5   input_queue
doc7.txt   8989  6   input_queue
doc8.txt   6735  7   input_queue
doc9.txt   9875  8   input_queue
doc18.txt  1209  9   input_queue
doc20.txt  1025  10  input_queue
doc23.txt  2020  11  input_queue
doc24.txt  347   12  input_queue
doc26.txt  789   13  input_queue
doc27.txt  257   14  input_queue
doc28.txt  1675  15  input_queue
doc29.txt  9837  16  input_queue
doc20.txt  721   17  input_queue
```

It is also possible to list the documents of the output query, using the "`-q`" command switch to select the queue name:

```
$ opt/sbin/ls_queue -q output_queue
doc1.txt_065a4fe5686a240b0569d35c6f04b025.naf 222715  21   output_queue
doc0.txt_0ff7e7f4e55ad14f085aada286f0b718.naf 219339  22   output_queue
doc3.txt_2ab255aac76c7b7110be1fe7cf9bfd0e.naf 214548  23   output_queue
doc4.txt_3762e590fe5ba62e96845491acda0656.naf 213492  24   output_queue
doc6.txt_98aef58c8b9b824006dd3c5a0d4d87cb.naf 212985  25   output_queue
doc7.txt_d0af8c39cd227492e6d6cf656be19491.naf 211940  26   output_queue
doc8.txt_45d6c827fbab1ef8f211a2a0ed2d448c.naf 210390  27   output_queue
doc9.txt_5cd1fbedf874d25391ba78ea789ebe90.naf 209767  28   output_queue
doc18.tx_621f460498502b42d4a9ddf2401349f9.naf 209104  29   output_queue
```

Note that these examples show documents consumed from the input queue are already processed and stored in the output queue.

`flush_queue`

Once the documents are processed, they are stored in the output queue. The boss machine contains a consumer that continuously pulls from the output queue and stores the documents in the `docs/output` directory. However, sometimes it is interesting to dump all the current content of a queue (input or output) into a directory. The `flush_queue` accomplishes this task.

```
$ opt/sbin/flush_queue -q output_queue -o path_to_docs
```

The parameters of the script are the following:

- `-o` (**required**): the directory to dump the documents.
- `-f` (optional): by default the script will not overwrite any existing document in the output directory. Set this parameter to overwrite the documents.
- `-q` (optional): the queue to flush. If omitted, the script dumps the document from the `output_queue` queue.

Logs

The cluster maintains a log of all the documents when passing over the several stages of the pipeline. Each time the document processing starts, or when it enters some particular stage of the topology (in any worker machine), a log entry is created. Besides, if a document processing fails a record is created with the name of the document and the module which caused the error.

All the log records are stored into the *mongoDB* database, and are accessible from within the boss node. To access the logs, one has to first enter the so called mongo shell. Being logged into the boss machine, run the `mongo` command:

```
$ mongo
MongoDB shell version: 2.4.6
connecting to: test
> use newsreader-pipeline
switched to db newsreader-pipeline
> db.log.find()
```

the `find` command will show all entries of the log database. Each log entry has the following fields:

- `_id`: internal ID.
- `tag`: the entry type. Possible values:
  - `DOC_START` -> the document "doc_id" enters the pipeline
  - `DOC_BOLT_RCV` -> the document "doc_id" enters the bolt "module_id"
  - `DOC_BOLT_EMT` -> the document "doc_id" exits the "module_id"
  - `DOC_BOLT_FAIL` -> the document name "doc_id" failed in bolt "module_id"
  - `DOC_FAIL` -> the document "doc_id" failed
  - `DOC_ACK` -> the document "doc_id" worked as expected
- `doc_id`: the document name.
- `module_id`: the NLP module.
- `hostname`: the node name.
- `timestamp`: the time stamp in mongoDB format.

Find documents with errors, also displaying the bolt which caused the error and the worker node

```
> db.log.find( { tag: "DOC_BOLT_FAIL" }, { _id:0, doc_id:1, module_id:1, hostname:1 } )
{ "doc_id" : "4MSN-CTV0-TX2J-N1W8_8bb9805598b55e10850852165981897f.xml",
  "module_id" : "FBK-time", "hostname" : "workervm1" }
{ "doc_id" : "4MR6-N790-TX33-71WM_bab251443f3e6869306c87f8173e11c4.xml",
   "module_id" : "EHU-corefgraph", "hostname" : "workervm2" }
{ "doc_id" : "4MRN-CMP0-TX37-G2P3_82436b4a93ef1826a280763ac2838a74.xml",
   "module_id" : "EHU-corefgraph", "hostname" : "workervm1" }
{ "doc_id" : "4MRP-5X50-TX2J-N2H8_d83f60a3c7a5e783b8e84fd5d13a9b69.xml",
   "module_id" : "EHU-corefgraph", "hostname" : "workervm3" }
...
```

Find successfully analyzed documents

```
> db.log.find( { tag: "DOC_ACK" }, { _id:0, doc_id:1 } )
{ "doc_id" : "4MR1-81M0-TWX2-W2WD_f5341438336a8f739fd3bb2192a3374b.xml" }
{ "doc_id" : "4MR2-D9G0-TX52-F26P_5484c46f08a6242518fa389f8f1e4851.xml" }
{ "doc_id" : "4MRC-JSD0-TX2J-N1V0_7a5dc0f87c195104251969c9ca100c6c.xml" }
{ "doc_id" : "4MRK-J360-TX2J-N39T_f51f321d7a7a60d8c802bfece498027b.xml" }
...
```

4.3 Starting and stopping the services

There are a number of scripts in the boss VM (under the `opt/init.d` directory) to start and stop many services. It is convenient to stop the puppet agent before stopping services:

```
$ sudo /etc/init.d/puppet stop
```

Likewise, the puppet agent needs to be restarted after the services are running again:

```
$ sudo /etc/init.d/puppet start
```

Here is a list of the provided scripts to start or stop the services. All these scripts require one command, `start` or `stop`, which starts or stops the service, respectively:

- `kafka_server`: it needs a parameter, `start` or `stop`. The script starts or stops the kafka service (document queues), respectively.
- `zookeeper_server`: start or stop the zookeeper service.
- `storm_boss_server`: start or stop the STORM nimbus service.
- `boss_servers`: this script start or stops all services in the boss VM.
- `wipe_all_boss`: this script is special and erases all documents of the queues. **Use it with care**! It is recommended to first dump the documents of the queues in some temporary directory before erasing the queues.

The worker services can also be started/stopped:

- `worker_servers`: start or stop all the services in the worker VM. Run this script inside the worker VM. Alternatively, run this script from inside the boss VM using the `pdsh` tool. Inside the boss VM, write the following:

  ```
  $ sudo pdsh -w ip_of_worker /home/newsreader/opt/sbin/worker_servers stop
  $ sudo pdsh -w ip_of_worker /home/newsreader/opt/sbin/worker_servers start
  ```