Web Services and Cloud-Based Systems

Assignment 2.2

Tom Peerdeman & Laurens Verspeek
10266186 & 10184465

April 24, 2015

1 Pumpkin

Just as in the previous assignment, we registered an init scripts as images with oneimage and we added these images to our VM template file. We used three init scripts to contextualize the VM: init-pmk.sh, mount.sh and start-pmk.sh. They are configured to run in the order: mount, init, start, to ensure the correct state of the machine. We also made images of the pumkin config file and the python seed files for the workers for pumpkin and added these images to our VM template. This way we can easily say which seeds to include in the VM.

init-pmk.sh
The init script installs all the necessary packages (python, numpy, scipy, nltk (for the classifier) etc..) needed to run pumkin. Then it clones pumpkin from github.com and switches to the tracula branch and installs pumpkin. Next, it copies the mounted config file and the mounted python scripts (worker seeds) to the correct folder.

mount.sh
Our Ubuntu image by default only has 2.3 Gb disk space available. The tweets themselves are download in the process of running the workers. These tweets come in a file of over 2GB size. Knowing this the 2.3GB is clearly not enough. So besides the Ubuntu disk image, we also created a volatile filesystem of 4 Gb in the VM template:

```
DISK = [ TYPE   = fs,
         SIZE   = 4096,
         FORMAT = ext4,
         TARGET = sdb ]
```

By default however the contextualization script only mounts the main disk (the VM image) to / (the file system root). There are no options to set the mount point of any other disk devices. To solve this we had to mount this new storage device ourselves. In the mount.sh script which is run at startup of the VM before any other contextualization, this filesystem disk image is then mounted from device */dev/sdb* to */root/tweets*. The tweets are stored at *~/tweets/tweets2009-06.txt*. So when pumpkin is run from the root user this expands to */root/tweets/tweets2009-06.txt*. Thus this mount point is conveniently also the location where the tweets are stored, and thus solves the storage problem.

start-pmk.sh
The start script is designed to run the pumpkin framework, configured to run the sentiment analysis seeds. However if we would just start it in the script the contextualization would block on running pumpkin. Also there is no way to monitor the progress and output of pumpkin. To solve this we used tmux. The start script creates a new tmux session, this session is named *pmk*. In this tmux session the script will now start the pumkin script using the following command:
python DRHarness.py –supernode –broadcast –taskdir /home/ubuntu/pmk-seeds/ -c /home/ubuntu/pumpkin/pumpkin.cfg –gonzales –endpoints="tcp://*.*"
By sshing into the VM we can attach to the tmux session via the *pmk* name. This way we are able to follow the progress of pumpkin once it has started.

2 Problems

2.1 Invalid load key

While running Pumpkin, the error "ERROR: Invalid load key '[" kept popping up on the screen. We lost a lot of time trying to fix this error, as we had to debug and print the several states of the workers. After waiting for the next lab session, we showed our error, but the assistants could not help us any further. Several hours later we finally found out what the problem was: We had to run Pumpkin with the –gonzales option. This somehow fixed this weird error. This in contrast of the start method of the pumpkin-workflow.pdf file available on blackboard. This document states that pumpkin should

be started like:
python DRHarness.py –supernode –taskdir ~/pmk-seeds –broadcast -c pumpkin.cfg
Which we assumed to be correct, however it was not.

2.2 Pumpkin randomly stops processing

When the invalid load key error was solved we got another problem, which we actually still did not solve yet. When running all the worker seeds on one single VM the process would start correctly. However after a couple tweets are processed the whole thing just stops. Listed below is a section of the output. Not listed here is the prints of the initialization of pumpkin, and the reset counter messages up to 185.

```
INFO - reset counter, total: 185
INFO - reset counter, total: 186
INFO - reset counter, total: 188
INFO - reset counter, total: 189
INFO - reset counter, total: 190
INFO - reset counter, total: 206
INFO - reset counter, total: 277
INFO - reset counter, total: 366
INFO - reset counter, total: 456
INFO - Removed stale entry for seed clda1503:filter at tcp://10.102.0.231:7900
INFO - Discovered remote seed: clda1503:filter at tcp://10.102.0.231:7900
INFO - Removed stale entry for seed clda1503:filter at tcp://10.102.0.231:7900
INFO - Discovered remote seed: clda1503:filter at tcp://10.102.0.231:7900
```

After the reset counter message of 456, no more reset counter messages are printed. Only the discovery and removal of other group VM services are printed. This indicates to us that either one of the three workers just stopped, and thus stopping the whole sentiment analysis process. We can confirm that the process is actually stopped by looking at the CPU usage of the python process. When this phenomenon occurred the process usage was around 5%, while before the sentiment analysis stopped the CPU usage was around 90%. The freeze happens within seconds of pumpkin initializing. The actual stopping point differed for consecutive test, but none of them reached reset counter 1000. We cannot explain this phenomenon, we can only assume it has something to do with either us wrongly configuring pumpkin, or interference from other groups.

3 Results

Single VM
As stated in the problems section the run where all seeds ran on a single VM never completed successfully. The collector did output some data, however since it only successfully ran for a couple of seconds, the data is very sparse, as can be seen in figure 1.

Two VM's
When using two VM's we have to decide which seeds we will run on one VM and which on the other one. In the problems section we have seen that the tasks are mainly bottlenecked by the CPU. It would therefore make the most sense to run the most CPU intensive seed on it's own VM. If we examine the seeds we see that the inject and collector seeds tasks are interacting with the disk. The filter seed does the actual processing, it would thus make sense to run the filter seed on a separate VM.

Unfortunately due to previous problems we did not have time to fully test the two VM setup. Therefore we also couldn't create the plot for the two VM setup. However the VM templates (ubuntu-pmk-injectcollect.one and ubuntu-pmk-filter.one) are included in the tar. This also means we don't know if the 'freeze' of the one VM setup also occurred in the two VM setup.

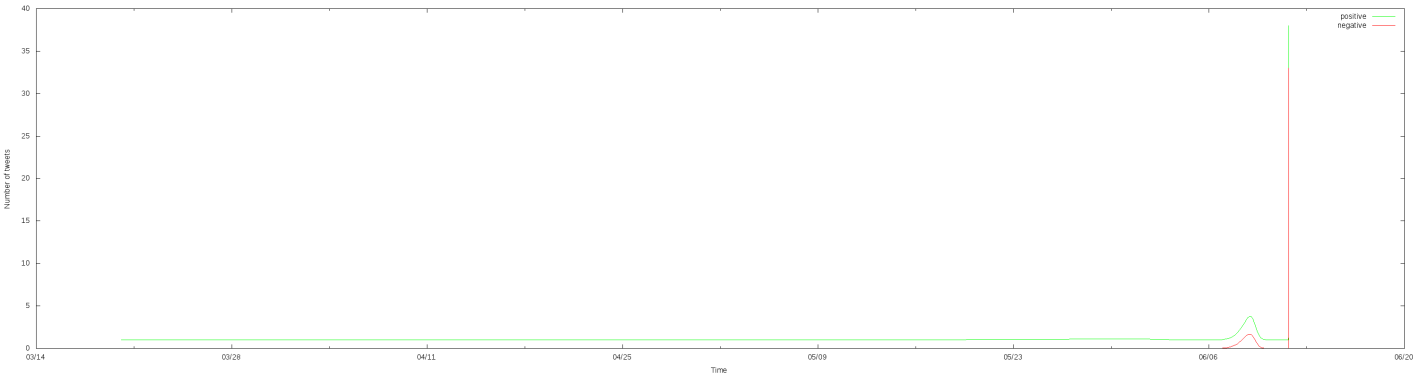

Figure 1: Tweet statistics - all seeds on one VM

Seed	CPU usage	RAM usage
Collector	80%	6%
Filter	99.5%	26.4%
Injector	20%	6%

Table 1: Performance metrics of the python process - one VM per seed

One VM per worker
For some reason, again we cannot explain it, the freeze bug does not occur when running all seeds on different VM's. As the test took quite a while, as it, for some reason, could only process about roughly 1000 tweets per second, we stopped it after 5 and a half hour. After this time it had processed about 9 million tweets. The resulting plot is provided in the tar as tweetstats.png. We found out that the tweets file has 74288337 lines of data. If we take a guess of 5 lines per tweet, that makes 14857668 tweets, meaning that we were just under a third of the tweets processed. It would have probably taken over 3 more hours to complete, time that we did not have, due to the deadline.

With the seeds separated we were also able to execute some performance measurements for the individual seeds. The results for the individual python processes are shown in table 1. The data confirms the assumptions made for the two VM setup. The filter seed is the computation intensive one. It also requires a lot of memory, probably to keep the classifier in memory. The injecting seed also seems to require quite a bit of CPU power. This can be explained by the tweetinject.py script parsing the tweets file to only inject actual tweet data.