ID2210 - Working with the cloud

Jim Dowling, Alex Ormenisan

March 2018

1 Introduction

These days, cloud computing is part of any IT company, be it giants like Google, Facebook, Spotify and even start-ups.Two of the main benefits of the cloud is that it is managed and you can request resources on demand. As such many companies prefer to invest all their resources on their specific software products and not on maintaining the infrastructure for the product.

It is thus important to get contact and understand the different technologies behind cloud infrastructure. This project, part of the ID2210 - Distributed Computing, Peer-to-Peer and GRIDS, focuses on components of the Google Cloud Platform(GCP) such as compute, storage and network. You will examine these components, deploy the necessary architecture and benchmark their performance.

2 GCP

Google Cloud Platform, offered by Google, provides a number of cloud services. In order to access the platform you will need a gmail account and will need to contact us to add you to a project which includes a number of Google credits.

3 Task 1 - External storage support 8p

1. Investigate the different types of storage support that cloud providers (GCP) supply. The storage types we are interested here are the ones that can be accessed from outside the cloud without having a vm active within the cloud. Write a short description(1/2p) of the limitations and characteristics of each storage type inspected.
2. Choose one storage type and provide code for accessing(writing/reading) the storage externally(without a vm). Write a short section(1/2p) on investigated relevant API options.
3. Benchmark the storage operations(write/read) performance. Report on relevant statistics both for single transfers and parallel transfers(same source/destination).

4 Task 2 - Setting up a vm 7p

1. Create a vm using the web consoles. Report(1/2p) the required steps and the relevant configuration steps you took.

2. Investigate APIs for remote creation of vms. Write a short(1/2p) description section.

3. Write simple code to access the vm from your own laptop through TCP/UDP. Include in the report section any relevant vm configuration changes(if any) that were required.

5 Task 3 - Internal storage support 8p

1. Investigate vm access support to the external storage from task 1. For example GCP allows mounting(limited) the storage buckets as disk for a vm. Report(1/2p) on your findings.

2. Benchmark access to the external storage(task 1) from within a vm. Report on relevant statistics both for single transfers and parallel transfers.

3. Benchmark access to local file system(ssd) from within a vm.

6 Task 4 - Simple Network 7p

1. Benchmark TCP/UDP communication. You can use tools like iperf or custom made java/scala programs.

2. Benchmark network in the same availability zone / across availability zones / across regions. Benchmark single transfers and parallel transfers. Benchmark at least 3 different availability zone configurations and 3 different region configurations.

7 Task 5(Bonus) - Custom Network 10p

Links across regions will typically have high latency, but also possibly high bandwidth. TCP has known limitations to fully utilize the whole bandwidth due to the latency problem. The custom network protocol we will use is Ledbat[1], [2].

1. Benchmark the Ledbat protocol. Gather statistics for single/parallel transfers as well as Ledbat and TCP parallel transfers. Ledbat is supposed to be a scavenging protocol and backoff when it encounters TCP traffic.

2. Modify/Optimize code/parameter setup and provide additional benchmark statistics as well as explanations for the newly determined behaviour.

The code in the git repository is seen as a guideline that you can use. The code implements mostly Ledbat from https://tools.ietf.org/html/rfc6817#ref-uTorrent with one major change from DTL - reducing the congestion window drastically (0.99) when the "offTarget" is negative. The offTarget being negative is an indication of a another stream, possibly another Ledbat stream. If the other stream is Ledbat and the current stream does not backoff drastically, the new-commer will estimate a wrong(higher) base delay on the link. This will lead to what is known as the "late-comer advantage" of Ledbat. By resetting the congestion window of the first stream, we make sure both stream will take the same estimation of the link delay and they will fairly share the link.

Another implementation detail is the use of a simple wheel timer. Kompics is not very good at tracking millions of timeouts, one for each outstanding message, and as such we will group close timeouts into one of the wheel's intervals. Afterwards we will periodically turn the wheel using one kompics periodic timeout.

The implementation is not complete and you will need to write the application code(sending the data) as well as the Kompics serialization of messages. If you find any issue in the implementation, do a git pull request and depending on the significance of the fix, you will receive bonus points(you are still limited to a overall maximum of 40 project points).

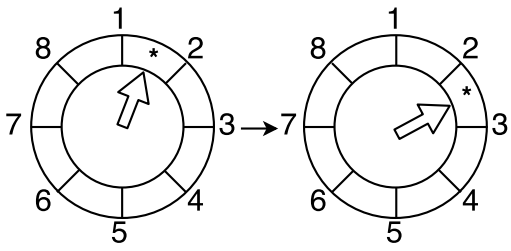

Figure 1: Wheel timer

8 Task 6(Bonus) - Use orchestration 5p

karamel.io is a tool for orchestrating deployment of complex software on either baremetal or GCP/AWS platforms. You can find the git repo here or get the executable here. If you download the sources, you can see here how to build karamel. When you start karamel, the karamel web-ui will open in your default browser. Load this GCP cluster definition from web-ui -> Menu -> LoadClusterDefn. In order to launch your cluster definition you must enable Compute Engine's OAuth 2.0 as described here and download your json GCP private key. You should also have set your local rsa key - karamel will enable this key on the newly booted machines so that you can ssh into them. You configure both your

rsa key and the GCP json key in the launch step.
Sumarized requirements:

1. Generate rsa key on your private machine. The key should not be password protected
2. Enable Compute Engine's OAuth 2.0 and download the json containing your private key

In karamel:

- Load cluster definition
- Configure local rsa key (karamel should pick it up automatically)
- Setup path to the GCP json key.
- Launch installation
- Click on the status of your cluster setup to see the status of the installation.

The loaded cluster definition launches 3 vms with different services on the machines. Once karamel says all tasks are done, using the provided public ip you can ssh, copy and run your jar on the machine with the private ip:10.132.0.2. You can also run this application from outside the GCP environment, but you might need to setup port forwarding rules for Hops port:8020.

You can find helper code and a very simple example here. The simple example creates a file in the Hops(HDFS) cluster you just deployed.

You can use this HDFS shell client to check the results of your java program. You will find the client under `/srv/hops/hadoop/bin/hdfs`.

You can generate and then copy the example jar on the GCP vm:

```
#cd to your vt18-id2210-cloud directory
mvn clean install
#change the <publicIp> with the Karamel value for
#machine with private ip 10.132.0.2
scp hops/target/hops-1.0-SNAPSHOT-shaded.jar ubuntu@<publicIp>
```

Copy, run the jar and then use the hdfs shell client to verify that the file is there:

```
#23.251.143.57 is the public ip for this run
scp hops/target/hops-1.0-SNAPSHOT-shaded.jar ubuntu@23.251.143.57:
ssh ubuntu@23.251.143.57
java -jar hops-1.0-SNAPSHOT-shaded.jar
/srv/hops/hadoop/bin/hdfs dfs -ls /test
```

GCP will reuse the public ip between your runs if they are close enough in time. In this case you might get "Host key verification failed" since a different vm will be running behind this ip now. In order to fix it you can remove the entry for your public ip from `~/.ssh/known_hosts`.

The help code provides wrapper operations for:

- create empty file

- read from file
- write to file (write is asynchronous, buffered - returning doesn't necessarily mean your data is in the cluster)
- flush (synchronous - thus slow) - forces the client to flush the buffered data to the cluster
- file length

Task requirements:

1. Deploy a distributed file system - HopsFS (variant of HDFS) on the cloud provider.
2. Write simple code to benchmark the read/write operations on the deployed HopsFS. Read HDFS documentation for the DFS.

9 Task 7(Bonus) - Dela integration 5p

Dela is a streaming application written using the java Kompics framework. Dela allows pluggable storage using the DurableStorageProvider. In this task you should wrap your external storage access code in a java Kompics component capable to deal with the DStorageWrite and DStorageRead events. If the external storage client allows asynchronous writes, I expect you to provide support for asynchronous writes and the DStorageWrite.Response to be issued only when you are sure that the data is in the cloud storage and not when it might still be in your local client's buffers. If you are using only the synchronous write mode of the GCP client, clearly show the APIs you have used and the fact that these APIs do not support asynchronous writes. Keep in mind that when libraries implement both synchronous and asynchronous writes there will be a significant performance difference between the two.

The Dela storage systems uses two abstraction to work with cloud storage: endpoints and resources. A resource identifies a file in the storage and might include a URI, a path, an identifier, depending what the storage picked by you uses to identify files. An endpoint is an identifier for the storage cloud container and should contain the data you need to identify and communicate with that storage endpoint. Add in the MyStreamEndpoint and the MyStreamResource the extra information you need to be able to identify/interact with files and the storage backend.

The main classes you will have to fill in are:

- MyStorageComp
- MyStreamResource
- MyStreamEndpoint

The application behaviour you should expect is for a new MyStorageComp kompics component to be created for any new file being accessed. If your storage supports only append type of writes, be sure to deal with this and order the writes such that they can be performed as append only operations.

The help code can be found here:https://gits-15.sys.kth.se/id2210/vt18-id2210-cloud/tree/master/dela

Task requirements:

- Provide the code to interact with the chosen external storage.
- Benchmark the kompics component to make sure that your wrapper code does not impact on the previously benchmarked external storage through blocking writes or other issues.
- Integrate fully with Dela and test (more to be announced here, might change the template code to fully work with the dela code, but the code you write will not need to change for this).

10 Code repository

The help code for tasks 5,6,7 can be found in the git repo: https://gits-15.sys.kth.se/id2210/vt18-id2210-cloud.

11 Report

The report should be provided as pdf. I suggest writing it in latex(sharelatex). Each task should have around 1/2p-1p reporting at least the steps/methodology/investigated resources. Submit the pdf report separately in Canvas. If you used git for your code, add a link in the report to the git repo. Adding this link does not replace the zipped code submitted on canvas. Your report should also include, where relevant, description on how to execute your code.

In canvas, you should submit:

1. Report submitted as pdf
2. Cleaned code (no target/logs) submitted as a zipped archive.

12 Notes

The base tasks 1,2,3,4 total 30p out of the project maximum 40p. Tasks 5,6,7 are marked as bonus, since you should choose a subset of them to provide you with 10p.

References

[1] Riccardo Reale, Roberto Roverso, Sameh El-Ansary, and Seif Haridi. Dtl: dynamic transport library for peer-to-peer applications. In *International Conference on Distributed Computing and Networking*, pages 428–442. Springer, 2012.

[2] Sea Shalunov, Greg Hazel, Janardhan Iyengar, and Mirja Kuehlewind. Low extra delay background transport (ledbat). Technical report, 2012.