OPTIMALLY DEPLOYING AN IMAGE PROCESSING APPLICATION ON THE AMAZON AWS CLOUD

A REPORT
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCES
OF UNIVERSITY OF WISCONSIN MADISON
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCES

Anmol Mohanty
December 2015

Department of Computer Sciences and Engineering
UW Madison

Abstract

Modern image sets and data are getting exponentially more huge with passing time. A specific example would be the Human Connectome Project dataset which runs into few hundreds of GigaBytes. This is leading to a trend where processing on the data locally on a single machine is turning out to be infeasible with regards to the sheer amount of time it takes for the job to finish. This calls for leveraging some of the recent advances in technology which has made it possible to bundle together bunch of off-shelf machines to speed up computation by processing in a massively parallel way. This is commonly referred to as the cloud-computing/big-data paradigm. This report demonstrates a way of migrating compute of an image processing application to the cloud and explores some of the design decisions and tradeoffs which are encountered in the process.

Keywords: Big Data, AWS, Cloud Computing, MGLM, Riemannian Manifolds ,
Image Processing, EC2, EBS

TABLE OF CONTENTS

1. **AWS Services :-**

   Amazon Web Services (AWS), is a collection of remote computing services, also called web services, that make up a cloud-computing platform offered by Amazon.com. These services operate from 11 geographical regions across the world. The most central and well-known of these AWS services arguably include Amazon Elastic Compute Cloud, also known as "EC2", and Amazon Simple Storage Service, also known as "S3". Amazon markets AWS as a service to provide large computing capacity more quickly and more cheaply than a client company building an actual physical server farm.

   The following is a list of AWS technologies that were used in production.

   - **Amazon EC2:-**

     Amazon Elastic Compute Cloud (EC2) forms a central part of Amazon.com's cloud-computing platform, Amazon Web Services (AWS), by allowing users to rent virtual computers on which to run their own computer applications. EC2 encourages scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image to configure a virtual machine, which Amazon calls an "instance", containing any software desired. A user can create, launch, and terminate server-instances as needed, paying by the hour for active servers - hence the term "elastic". EC2 provides users with control over the geographical location of instances that allows for latency optimization and high levels of redundancy.

   - **Amazon EBS:-**

     The Amazon Elastic Block Store (EBS) provides raw block devices that can be attached to Amazon EC2 instances. These block devices can then be used like any raw block device. In a typical use case, this would include formatting the device with a file-system and mounting it. In addition EBS supports a number of advanced storage features, including snapshotting and cloning. EBS volumes can be up to 1TB in size. EBS volumes are built on replicated storage, so that the failure of a single component will not cause data loss.

   - **Elastic IP:-**

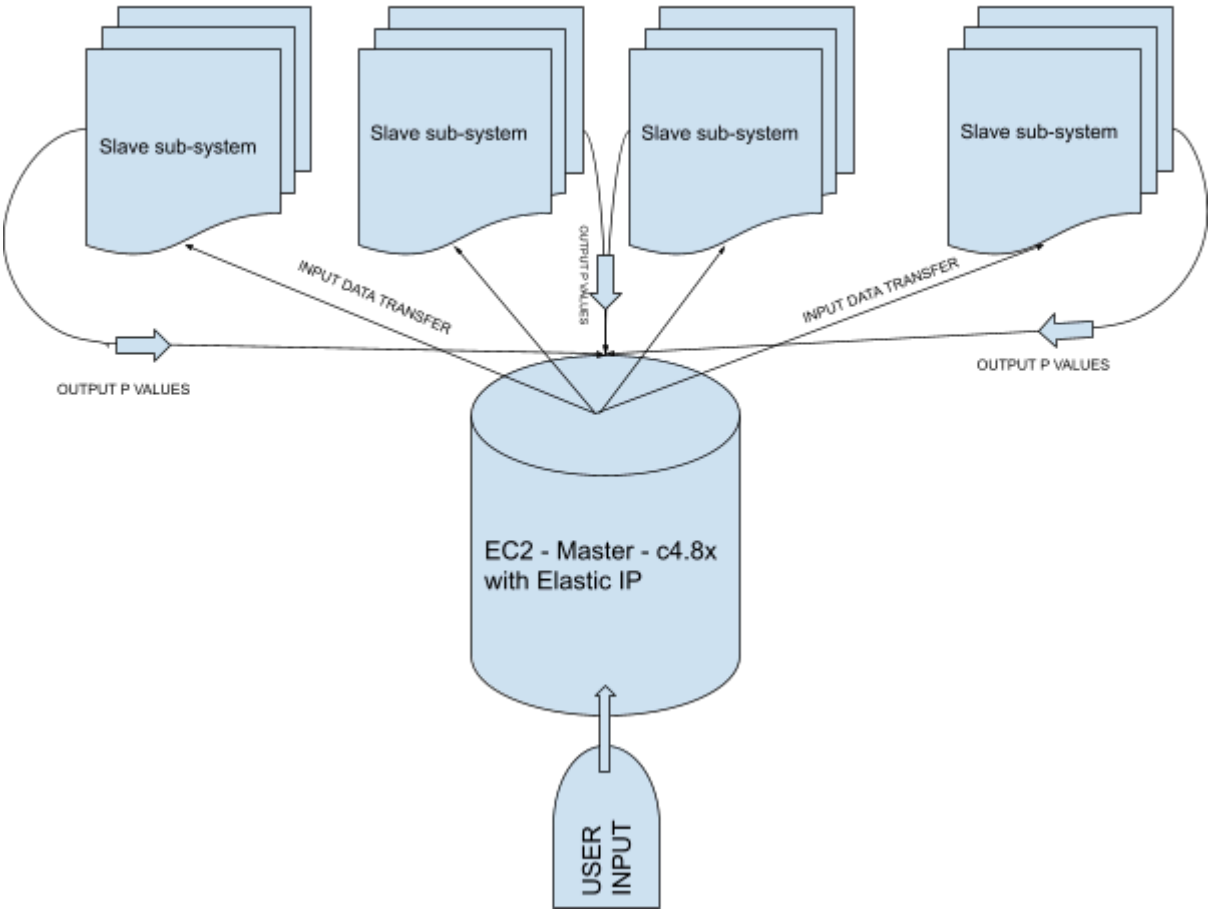
     The elastic IP address feature is similar to static IP address in traditional data centers, with one key difference. A user can programmatically map an elastic IP address to any virtual machine instance without a network administrator's

help and without having to wait for DNS to propagate the new binding. In this sense an Elastic IP Address belongs to the account and not to a virtual machine instance. It exists until it is explicitly removed, and remains associated with the account even while it is associated with no instance.

2. The 'Brain-AWS' system :-

The following figure gives a diagrammatic overview of the system.

Figure 1: Overview of the system



a) Key details :-

These are the components of the system built. These systems are co-located in the us-east-1b region(N. Virginia).

a. Master - This is a c4.8x instance persisted with an Elastic IP and a EBS backed storage of 100GB SSD with 300 IOPS (I/O operations per second).
b. Slaves - There are a configurable number of worker slaves in the system, although due to constraints of AWS regarding the number of instances permitted.

b) The Compute Flow

This is a bird's eye overview of how the system works. Each step has been detailed with the corresponding code snippet used.

A. First data is pre-processed on the master server. This is a MATLAB segment and it generates the intermediate outputs and the .MAT files comprising the image data and the metadata (age, gender, family history) which are used in subsequent steps of the routine.

```
matlab -nosplash -nodesktop -r run step1.m
```

B. The data is produced on the master instance and split into a configurable number of chunks(equal to the number of slaves).

C. The master spawns configurable number of slaves on the AWS cloud

```
aws ec2 run-instances --config
```

D. Then the master initiates connections to the slaves and authenticates the network connectivity to enable the slave to copy data back to the master.

E. The master obtains a list of slave IPs and uses it to distribute work and do any necessary processing.

```
aws ec2 describe-instances > servers.txt
grep "PublicIpAddress" servers.txt > public_ip.txt
#should store list of ips in public_ip.txt ##done
```

```
#interesting nice script to properly format the file
vim -S vim.trim public_ip.txt

#remove the master ip address from this list
sed '/52.5.57.157/d' public_ip.txt > final_public_ip.txt

#get list of instance ids handy for emergency termination if
necessary
aws ec2 describe-instances | grep "InstanceId" > ids.txt

#trim the file to the proper format with vim.trim.instances, same
file works !!magic
vim -S vim.trim ids.txt
```

F. The master copies over the corresponding data partition to the slave.(This is done for all slaves sequentially).

```
#The variable input data folder
scp -q -i ~/data/brain_aws.pem -r $a ec2-user@$line:~/
#The shared data folder common to all
scp -q -i ~/data/brain_aws.pem -r shared ec2-user@$line:~/
scp -q -i ~/data/brain_aws.pem ~/.ssh/config_copyl
ec2-user@$line:~/.ssh/config
#Look into implementing this feature
#scp -q myfile user@host.com:. && echo success!
echo "Completed transfer to Machine #$a @ $line---------->  Success"
```

G. The master launches jobs on the slave by running a remote command by spawning a virtual shell. This is done in parallel (by spawning processes for each slave).

```
#these need to be done in parallel on the slave nodes
ssh -i ~/data/brain_aws.pem ec2-user@$line 'bash -s' < local.sh &
```

H. The slaves operate the commands, on the data copied on them, process and generate the output (the p_value file) which is then copied back to the master.

```
echo "Job starting on $a"
#legr stuff | compute
./shared/legr_dti_parallel $a/. shared/.
echo "Job completed on $a"
#copy back p_value to master
cp p_value.txt $a.txt
echo "Copying back p_value result from Machine #$a to Master"
scp       -q       -i       LaunchEC2installMATLAB1023.pem       $a.txt
ec2-user@52.5.57.157:~/PVALUES/.
```

I. The master verifies data integrity of the incoming files and makes sure all the slaves have copied their data back.

J. Upon verification, the master initiates termination of the slave instances.

```
aws ec2 terminate-instances --instance-ids "$line"
```

K. The master concatenates all the p_values onto a single aggregated file.

```
mv 1.txt 01.txt
mv 2.txt 02.txt
mv 3.txt 03.txt
mv 4.txt 04.txt
mv 5.txt 05.txt
mv 6.txt 06.txt
mv 7.txt 07.txt
mv 8.txt 08.txt
mv 9.txt 09.txt
mkdir FINAL_P_VALUE
cat *.txt >> FINAL_P_VALUE/final_p_value.txt
```

L. Then a mapping is done with the original .MAT file. This ensures that we have a one-one mapping between the brain voxels coordinates and the p_value corresponding to it.

c) MATLAB preprocessing step and core executable:-

The input to the whole system is a bunch of CDT brain scan images preferably of many subjects and the corresponding metadata file. The metadata file contains the age, gender and miscellaneous information like family history, genetic information, disease history etc. There is additional MATLAB method which can take raw input images in the Jacobi Tensor format (3x3 Positive definite but not symmetric) and convert them into CDT if the end user so desires.

The first and foremost step is to aggregate all the image data, meta data and mask information into a MATLAB .MAT file to begin subsequent processing.

Next random permutations are computed for permutation testing to give a confidence metric of the result obtained post which the random permutations is added to the .mat file as well. In current models 20,000 random permutations are used. The next step is

slightly more involved and in this step we split the .mat file into partitions equal to the number of servers we have for processing. The output of this step are 3 .mat files. The Y .mat file which is the output and the X mat file (comprised of inputs and the target column). These two make the core of the model we are trying to fit and find errors in. Additionally a mask is also generated specific to the Y .mat file in the partition output.

These partitions are written out to separate folders and shipped out to the various servers after which p_value computation is done on the partitions independently (as described in previous paragraphs).

In the executable file, the error is computed between the actual values and the predicted values based on model fitting via a distance heuristic (like least square error).

The files and source code may be found at:- my GITHUB

3. Some notable technical aspects of the project.

A. Multithreading :-

Full multi-threading wherever possible has been implemented both to leverage premium AWS resources and speed up computation tremendously. We have observed that the c4.8xlarge, compute intensive instance with 36 vCPUs gives a speedup of greater than 20 which is instrumental in making sure that a very long job, like the ones that are tested finish quickly. In fact with multi-threading enabled, processing 1000 voxels took about 50 minutes to process translating to a speed-up of 40x. In the end making sure the code is thread safe and there are no memory leaks made enabling multithreading worthwhile.

B. Verifying the p_values :-

This was an interesting problem to tackle, albeit trivial. The master makes sure with a long running script that the number of files(partitions) is equal to the number of slaves which indicates a healthy state, it also verifies the p_values = by reading in the files and making sure that each line is a rational number between 0 and 1.

C. Optimizing the compute model :-

In the original model of the code , the model was run twice, one with the target input vector present and one with the vector absent. The error binary files for both the runs would be written out separately, post which they would be read back and the delta error read. This made

D. Networking Challenges:-

One of the challenges in any distributed system is the inter-node communication. Here the system does an interesting thing to be able to inter-communicate without giving up network security. It ingeniously obtains the list of active worker nodes and their IPs and appends them to it's security file to allow ssh into the machines without asking for a prompt (thus enabling automated data transfers) . It then also copies in its security passkey to the slaves enabling them to transfer data to itself securely. This has been explained via code snippets above.

E. Managing the costs:-

A good way to keep costs low is to spin up the compute instances only when you need them and terminate them once the job is done. This would have been possible very easily had there not been MATLAB code portions, since it is not free and licensed. So MATLAB runtime is needed to process the raw data to produce the intermediate data which can be consumed by a c++ based linux executable (which is free and available everywhere). So the solution used is to have a persistent EBS with a running copy of MATLAB on it (In this system, a 100 GB SSD EBS is allocated for this purpose), So when the first instance spins up, the EBS store can be mounted onto the instance and computation can proceed as usual. This allows for no-instance to be persistently running and allows quick boot-up, attach and access to the compute. Another interesting heuristic we came up with is the idea of bounding computation to multiples of hours as AWS charges in hourly increments, so a user would be charged double for a usage of 61m as opposed to 59m. Keeping it just under an hour would be a great way to keep costs under control.

4. Experimental results:-

Instance	Multi-threading	Voxels Processed	Time taken	Cost/hr	Total Time(Hrs)	Overall cost (20k voxels)
c4.8xlarge	No	20	43m	$1.763	717	$1264.00
m3.large	No	20	98m	$0.133	1634	$217.23
t2.small	No	20	122m	$0.026	2033	$52.87
t2.large	No	20	55m	$.1040	917	$105
t2.medium	No	20	100m	$0.052	167	$104
c4.8xlarge	Yes	400	53m	$1.763	20	$80

A very interesting set of results were obtained by running the compute across various instance types.

One of the key takeaways is that it prohibitively expensive to run the compute on a premium machine without exploiting multithreading. Another smart thing to do would be to run the job only using one partition to sample the result and use that to scale up the estimate cost. This would give a fair idea of how much cost will the real job incur without actually spending up so much.

After due experimentation and analyses, it is concluded that the c4.8xlarge(the last row) with proper multithreading support built in is best suited for the project's needs as it finishes the job in just under an hour and leaves ample time for the system bootup and also result copyback and verification to successfully complete.

5. Conclusion:-

The full job comprised of 200,000 voxels completes in under an hour, with 50 slaves and costs about $80 to complete. An equivalent job on condor took 5 hours and $140 (by our estimates) to complete. The massive improvement in speed up and substantial cost benefit stand testament to the amazing infrastructure we have access to today; and software if written well can leverage it fully to produce some fantastic results. This has the additional benefit of being open to the world while condor is only available to academics.