

# Big Data and Compute Canada Cloud

Mohanavel Inbavel  
MSc (Computer Science),  
Lakehead University,  
Thunder Bay, Ontario, P7B5E1  
Email: minbavel@lakeheadu.ca

Dr. Jinan Fiaidhi  
PgD, PhD, MBCS, ISP, IEEE Senior  
Member, PEng  
Department of Computer Science,  
Lakehead University,  
Thunder Bay, Ontario, P7B5E1  
Email: jinan.fiaidhi@lakeheadu.ca

**Abstract**—Big Data is changing the way Research is conducted, especially in fields where computational methods were not traditionally used. So, processing of huge amount of data can be next to impossible when doing it in a personal system. To compute these data, Traditional HPC clusters in compute Canada cloud can be used where the user has control over the software stack and can configure it with typical Big Data Stacks. The question we begin to explore is how we can increase the efficiency and get desired output by reducing the processing time using HPC's where our input will be a Big Data stack.

**Keywords**—Big Data, Compute Canada Cloud, High Performance Computing (HPC), Parallel Processing, Clusters

## I. INTRODUCTION

**Big Data** is nothing but extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions. If the size of the data is huge and contains a lot of information and requires a lot of storage capacity. The big data is the combination of volume i.e. the data can be obtained from various sources like business transactions, social media etc. so large volumes of data is created, velocity, the data is being collected from time to time and the data streams in high speed and variety, the data comes in different formats like structured and unstructured.

The structured data have a schema like numbers, dates etc., it can be easily processed and organized into the database. and whereas the unstructured data doesn't have a schema and is hard to analyze as it cannot be interpreted by classic data models. The examples of this kind of data are web pages, textual data etc.

To process these huge amounts of data we cannot do it in our normal systems. We need a high-performance computer to solve these kinds of problems. This is where Compute Canada Cloud is used.

The **Compute Canada Cloud** service responds to researchers who have a need for greater configurability, availability, durability or portability than may be available on non-cloud systems or clusters. By using virtual machines (VMs) you configure through your account, you can customize the computing environment to meet your unique needs. Carry out both data and computationally intensive work easily. Then share your work with your other devices, your team or other collaborators. Use the Compute Canada to build portals and platforms, or to handle data that you're scraping from the web.

The compute Canada cloud resource is an accumulation of hardware that supports virtualization, it can be considered as infrastructure as a service. The researchers who need considerable availability, configurability, durability or portability can seek the infrastructure through Compute

Canada cloud. The user can access the host and install any software with administrative privileges and run the needed applications. The applications can be CPU intensive or GPU intensive or memory intensive. The advantage of this resource is that the user has complete control over the software stack.

## II. IMPLEMENTATION

### A. Singularity

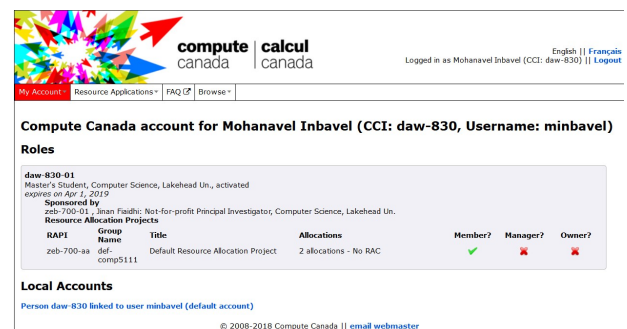
The newer national HPC systems in Compute Canada Cloud now has Singularity Installed. Singularity enables users to have full control of their environment. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data. This means that you don't have to ask your cluster admin to install anything for you - you can put it in a Singularity container and run.

### B. Resources Available in Compute Canada

The These Resources are new resources and were deployed only after 2016. Compute Canada contains of four clusters where you could access their HPC. These clusters are called as Arbutus, Cedar, Graham, Niagara.

### C. Accessing These Resources

Compute Canada Cloud provides these resources to researches or a student from any university in Canada. To get these resources a sponsor or Principal Investigator needs to request for the permission and add the other members to the group. Everyone else is given a Default Resource Allocation where they could access any of the latest HPC's. We will be accessing Graham depending on the geographical location or preference.



The screenshot displays the 'Compute Canada account for Mohanavel Inbavel (CCI: daw-830, Username: minbavel)'. It includes a header with the Compute Canada logo and navigation links. The main content area shows account details, roles, and local accounts. The roles section lists the user's role as 'Master's Student' with a table of allocations. The local accounts section shows the user's default account.

| RAPI       | Group Name   | Title                               | Allocations            | Member? | Manager? | Owner? |
|------------|--------------|-------------------------------------|------------------------|---------|----------|--------|
| zeb-700-aa | def-comp5111 | Default Resource Allocation Project | 2 allocations - No RAC | ✓       | ✗        | ✗      |

Fig:1 Compute Canada Account

To connect to the HPC we are using MobaXTerm a toolbox for remote computing in windows. It provides all the important remote network tools like SSH (Secure Shell),

X11 (Graphical processing) and so on. It also helps the users to run the Unix commands like bash, ls, cat, cd etc.

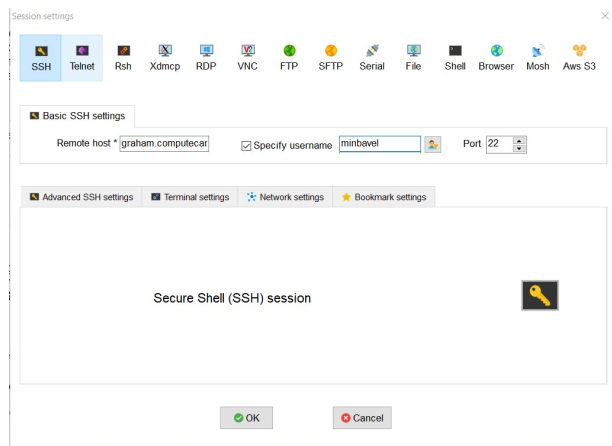


Fig:2 MobaXTerm SSH Login

Secure Shell (SSH) is a widely-used standard to connect to remote servers in a secure way. The entire SSH connection is encrypted - especially the login credentials (username and password). SSH is the normal way for Compute Canada users to connect in order to execute commands, submit jobs, follow the progress of these jobs and in some cases, transfer files. To create an SSH session in MobaXTerm click on create a new SSH session and in that Remote host enter which host you are connecting, in our case graham.computecanada.ca and for username enter your Compute Canada username and port 22 is nothing but SSH connection and click create. A new window pops and enter your compute Canada password and now you are connected to the Graham HPC.

#### D. Transferring Files

Files can be transferred using a Secure File Transfer Protocol (SFTP). We are using WinSCP to do it. Any SFTP program can be used to transfer the files into the HPC. Enter graham.computecanada.ca as Host name, port 22 for SSH and username, password to connect to the remote system.

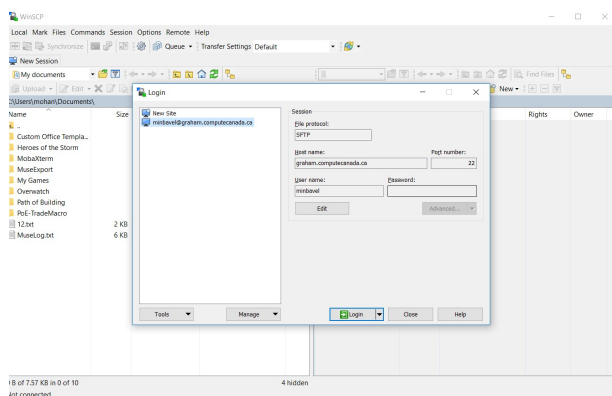


Fig:3 WinSCP Login

Files can be transferred by clicking and dragging. They can also be easily modified and deleted.

#### E. Running RScript in HPC

R programming language is mainly used to deal with Big Data problems. So, to run these RScripts in Graham HPC, we need to first install R in it.

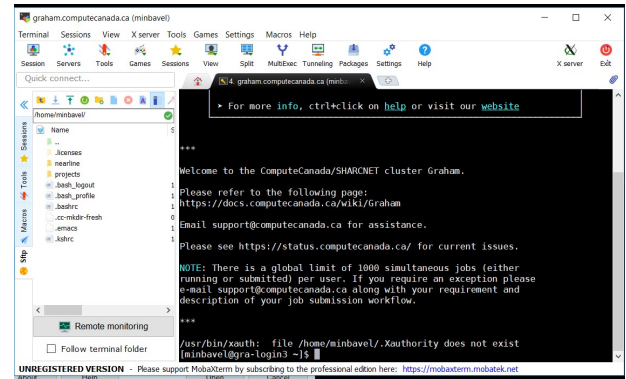


Fig:4 Accessing the HPC

To install R, they already have those packages available in the HPC and we just need to install them. Use commands like module spider r to check the available R packages and module load r/3.5.0 to load R in the system. Use module load gcc/5.4.0 to compile these R files in their system. To install different libraries in R use commands like install.packages("package name"). The packages are installed from CRAN mirror. CRAN is short for comprehensive r archive network which R packages are maintained and being downloaded from. CRAN mirror is the referral websites towards CRAN so that network traffic are better maintained and thus download speed is faster for a mirror near the location.. To run the R Script use RScript Filename.R .

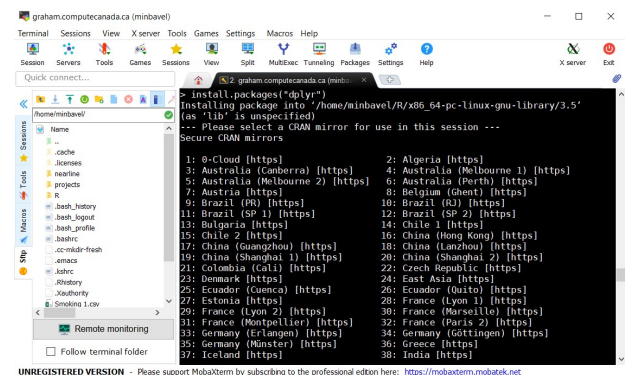


Fig:5 Downloading from CRAN mirror

### III. DATASET

The data used for this project is a .csv (Comma-separated values) file. This file contains one lakh datapoints of random.

|    | A  | B | C | D |
|----|----|---|---|---|
| 1  | 56 |   |   |   |
| 2  | 21 |   |   |   |
| 3  | 6  |   |   |   |
| 4  | 84 |   |   |   |
| 5  | 92 |   |   |   |
| 6  | 1  |   |   |   |
| 7  | 4  |   |   |   |
| 8  | 67 |   |   |   |
| 9  | 24 |   |   |   |
| 10 | 52 |   |   |   |

Fig:6 Sample Dataset

The sample data set contains numbers between 0 and 100. This data set is created using floor () function with the help of This data is the input and the output after processing the data should be 1 if the number is less than 51 and 0 if the number is greater than 50. We added this Data set to the HPC using WinSCP.

|    | A  | B | C | D |
|----|----|---|---|---|
| 1  | 56 | 0 |   |   |
| 2  | 21 | 1 |   |   |
| 3  | 6  | 1 |   |   |
| 4  | 84 | 0 |   |   |
| 5  | 92 | 0 |   |   |
| 6  | 1  | 1 |   |   |
| 7  | 4  | 1 |   |   |
| 8  | 67 | 0 |   |   |
| 9  | 24 | 1 |   |   |
| 10 | 52 | 0 |   |   |

Fig:7 Processed Dataset

#### IV. METHODOLOGY

The main objective of our project is to implement parallel processing in HPC so that the time taken to process a huge amount of data should be significantly less when compared to processing the same data serially. This can be done in two methods

##### A. Method 1:

Singularity is used to create multiple containers and depending on the number of containers the data can be split into different parts and processed parallelly in each container. This can be done by,

- Building a container from singularity hub where the user can choose the type of container he wants and what operating system it runs. It creates a virtual environment for him to work in.
- Create a job submission script (.sh file) to submit jobs in a shell file. It tells the system on how to split

the data and send them to containers and then process them.

- To submit these jobs sbatch is used. (sbatch - Submit a batch script to Slurm. Slurm is an open source cluster management and job scheduling system for large and small Linux clusters)
- By default, the output is placed in a file named "slurm-", suffixed with the job ID number and ".out"

##### a) Limitations:

The main drawback of this method is you need an RAC (Resource Allocation Competition) account where depending on the users project and their resource required compute canada will provide the account. RAC account is needed to submit any jobs in their system and thus data cannot be split into different parts and processed parallelly. But all the basic features can be done in singularity.

##### B. Method 2:

Due to limitations in Method 1 we are implementing the same parallel processing in HPC using clusters. Clusters act as a work environment on a core. Clusters have a separate node set to perform a task, which is controlled by a software or a scheduler. The constituents of the clusters are generally connected to each other, with each node performing its own task. Normally every node is dependent on the same hardware and the operating system. Clusters are generally used to enhance the performance and availability over that of a single computer while being cost-efficient. We have also gone through this paper [3] "**High Performance Computing in R**" by **Rahim Charania**. The basic idea is to

- Split the problem into pieces
- Execute the pieces in parallel
- Combine the results back together just by using R language.

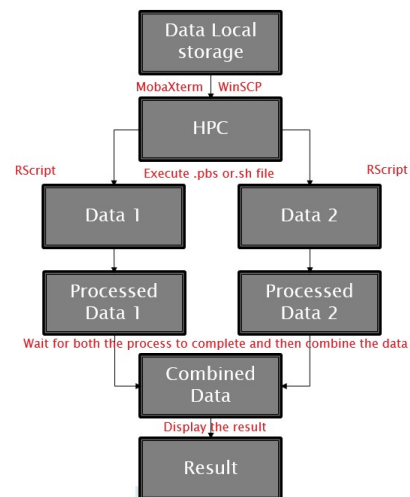


Fig:8 Flow Chart

#### a) Serial Processing:

Serial processing is the act of attending to and processing one item at a time. The computation time took 37.07 seconds for us process the data set serially.

```

> old<-Sys.time()
> num1_processed_in_series <- num1 %>%
+   mutate(
+     total = map(.x = num1$X1,
+                 ~ fun5(X1 = .x,
+                       return_format = "tibble")
+   )
+ )
> new<- Sys.time()-old
> t(new)
Time differences in secs
[1,] 37.07558

```

Fig:9 Processing Time for Serial Processing

#### b) Parallel Processing:

The steps involved in parallel processing are:

- **Detecting cores:** Core is an independent processing unit in a processor. We must detect the cores in the processor before splitting the data. Cores can be detected by using 'detectcores ()' function.
- **Splitting data into groups:** We must split the data into multiple groups depending on our choice or the number of cores. We have used a sequential vector of 1:nto split the data into groups, n can be number of cores or a number of choices. In our project we divided the data into 5 groups.

```

[ reached getOption("max.print") -- omitted 50001 rows ]
> detectCores()
[1] 32
>
> group <- rep(1:5, length.out = nrow(num1))
> num1 <- bind_cols(tibble(group), num1)
> num1
# A tibble: 100,000 x 2
  group  X1
  <int> <int>
1     1   20
2     2   36
3     3   94
4     4   75
5     5   64
6     1   78
7     2   98
8     3   33
9     4   78
10    5   51
# ... with 99,990 more rows

```

Fig:10 Splitting Data into Groups

- **Creating clusters:** We need to create clusters depending on the number of cores and the number of groups the data is split into. Clusters act as a work environment on a core. We have created 5 clusters in our project. The data is processed in clusters parallelly.
- **Partition by group:** Partitioning is sending a part of initial data frame into the cluster. Party\_df is the name of the partitioned data frame. We are using partition () function to split the data by group and sending each group to a different cluster. The data is stored in shards, the data is split into 5 shardseach shard has 20,000 data elements.

```

Source: party_df [100,000 x 2]
Groups: group
Shards: 5 [20,000--20,000 rows]

```

```

# S3: party_df
  group  X1
  <int> <int>
1     2   36
2     2   98
3     2   96
4     2   42
5     2   13
6     2   27
7     2   46
8     2   65
9     2   47
10    2   50

```

Fig:11 Partition by Group

- **Cluster Setup:** We need to setup clusters by adding libraries and defining cluster functions. We need to load each library into every clusters before using them for processing. We added cluster libraries by using Cluster\_library("library name ") function. In order to check if the libraries are added into the clusters, we use cluster\_eval() function.

```

by_group %>%
# Assign libraries
cluster_library("tidyverse") %>%
cluster_library("stringr") %>%
cluster_library("lubridate") %>%
cluster_library("quantmod") %>%
# Assign values (use this to load functions o
cluster_eval(cluster_name) h5", fun5)

```

Fig:12 Cluster Libraries Included

- **Execution of code:** After dividing the data into groups and sending them to clusters we run the code parallelly. We used mutate() function, which replicates the function we created and sends it into different clusters, map() function which maps each element to the function and collect() function which collects the data and stores in a tidy data frame. The time elapsed is recorded

```

old<-Sys.time()
num1_processed_in_parallel <- by_group %>% # Use by_group party_df
  mutate(
    total = map(.x = X1, ~fun5(X1 = .x,
                              return_format = "tibble")
  )
  ) %>%
  collect() %>%
  as_tibble()
new<- Sys.time()-old
t(new)
Time differences in secs
[1,] 7.699522

```

Fig:13 Processing Time for Parallel Processing

## V. RESULT

The time taken to process the data parallelly in an HPC using clusters is 7.69 seconds. This processing time taken is when using five clusters. When we use ten clusters to do the same process the processing time again gets divided into half.

```
Time differences in secs
[1,1]
[1,] 4.665699
```

Fig:14 Processing Time for 10 clusters.

## VI. CONCLUSION

The above Figures displays clearly that the time taken to process the data parallelly is significantly less than the time taken to process the data serially. Even if the time difference is less in these cases, when we are processing billions of data where the system runs for days or even months, reducing the processing time by half itself creates a huge difference. Thus, processing big data files parallelly can save a huge amount of time for researchers or big investors working in clouds or HPC's.

## VII. FUTURE RESEARCH

In big data, the datasets will contain multiple attributes, if we can group the data based on the attributes and use clusters to process them, the performance and efficiency will significantly increase. As each core works as a unique processor, we can program nodes with different functions so that we can get outputs of the respective functions parallelly.

## ACKNOWLEDGMENT

We would like to thank **Dr. Jinan Fiaidhi** for research ideas and provided us the access to the Compute Canada Cloud services which helped us in accessing HPC's and doing testing on it. She helped us by reviewing our work and guided us to make necessary changes and improve our work.

## REFERENCES

- [1] <https://www.computecanada.ca/research-portal/national-services/compute-canada-cloud/>
- [2] [https://docs.computecanada.ca/wiki/Getting\\_Started](https://docs.computecanada.ca/wiki/Getting_Started)
- [3] <https://quantdev.ssri.psu.edu/tutorials/high-performance-computing-r>
- [4] <https://www.business-science.io/code-tools/2016/12/18/multidplyr.html>
- [5] <https://www.computecanada.ca/research-portal/account-management/apply-for-an-account/>
- [6] <https://www.ace-net.ca/wp-content/uploads/2016/05/Handling-Big-Data-on-the-Cloud.pdf>
- [7] <https://www.computecanada.ca/research-portal/accessing-resources/available-resources/>
- [8] <https://docs.computecanada.ca/wiki/Singularity>
- [9] [https://docs.computecanada.ca/wiki/Available\\_software](https://docs.computecanada.ca/wiki/Available_software)
- [10] <https://docs.computecanada.ca/wiki/R>