

**Parallel Computing
Achieved using AWS and
Compute Canada Cloud**

Table of Content

1. Abstract
2. Keywords
3. Introduction
 - 3.1 Big Data
 - 3.2 Cloud Computing
4. Potential Benefit
5. Tools and Technologies
 - 5.1 Compute Canada Cloud
 - 5.2 MobaXterm
 - 5.3 WinScp
 - 5.4 Amazon Web Services
6. Feature Description
 - 6.1 Parallel Processing
 - 6.2 Clusters
7. Design and Methodology
8. Implementation
 - 8.1 Data
 - 8.2 Implementation on Compute Canada Cloud
 - 8.3 Implementation on Amazon Web Services
9. Future Research Work
10. Conclusion
11. References

1. Abstract:

Big Data has an important value and immense perception in industry and economy as well as in science and research. It needs to be stored, managed and analyzed for addressing the problems that were difficult to deal with. The processing of this data is next to impossible when doing it in a personal system. To compute Big Data, Cloud computing can be used as it has high-computational power. The efficiency of the model can be increased even more if we process the data parallelly using Cloud Computing.

2. Key Terms:

Big Data, Compute Canada Cloud, Amazon AWS, Clusters, Parallel Processing.

3. Introduction:

With the rapid growth of Cloud Computing and Big Data, the demand for High-Performance Computing is significantly increasing. Compute Canada Cloud makes High Performance Computing available to the researchers, and Amazon Web Services(AWS) has pay-per-use methodology, these resources substantially improves the allocation of complex tasks to the allocated systems thus increasing efficiency and productivity.

3.1 Big Data:

Big data simply represents huge sets of data, both structured and unstructured, that can be further processed to extract information. Huge volumes of data in all kinds of formats over the internet every second.

Businesses store that data to be analyzed later using tools increase the outcome and performance. Big Data can be structured or unstructured that can be further processed for analysis using Data Analysis. Big Data carry hidden patterns and algorithms which are unlocked by using various tools available in the market. These datasets are further analyzed to provide business insights.

3.2 Cloud Computing:

Cloud computing is shared pools of configurable computer system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, like a public utility.

Cloud computing is a technology used to store data and information on a remote server rather than on a physical hard drive. Cloud here signifies the internet which in this case, acts as IaaS (infrastructure as a service). There is no need to run programs on your own servers. Simply just use the internet to access programs.

4. Potential Benefit:

Processing of Big data has become easier with the computation power provided by Cloud providers. Hence, we can increase the efficiency and productivity of the model if we use Parallel Processing. We can divide Big Data into multiple chunks and process them parallelly, with this the time required for computation is reduced, henceforth increasing the efficiency and the resources are used productively, without any wastage.

5. Tools and Technologies:

5.1 Compute Canada Cloud:

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.

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.


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 High-Performance Computing. These clusters are called as Arbutus, Cedar, Graham, Niagara.

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. We will be accessing Graham depending on the geographical location or preference.



compute
canada

calcul
canada

English || Français
Logged in as Mohanavel Inbavel (CCI: daw-830) || [Logout](#)

[My Account](#) | [Resource Applications](#) | [FAQ](#) | [Browse](#)

Compute Canada account for Mohanavel Inbavel (CCI: daw-830, Username: minbavel)

Roles

daw-830-01
Master's Student, Computer Science, Lakehead Un., activated

Resource Allocation Projects

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

Local Accounts
[Person daw-830 linked to user minbavel \(default account\)](#)

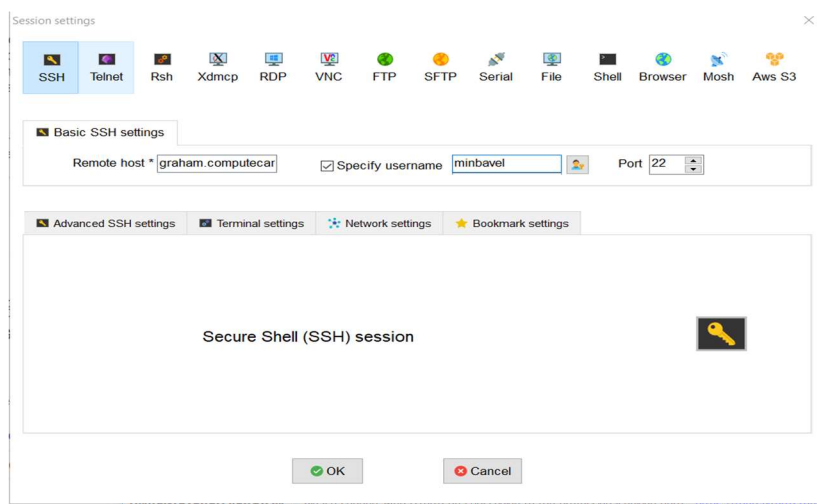
© 2008-2018 Compute Canada || [email webmaster](#)

5.2 MobaXterm:

MobaXterm is one of the toolboxes for remote computing.

It provides all the important **remote network tools** (SSH, X11, RDP, VNC, FTP, MOSH, ...) and **Unix commands** (bash, ls, cat, sed, grep, awk, rsync, ...) to Windows desktop, in a **single portable .exe file** which works out of the box. It also has a built-in SFTP client to transfer files as well as a built-in X11 server to allow you to run graphical programs.

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.

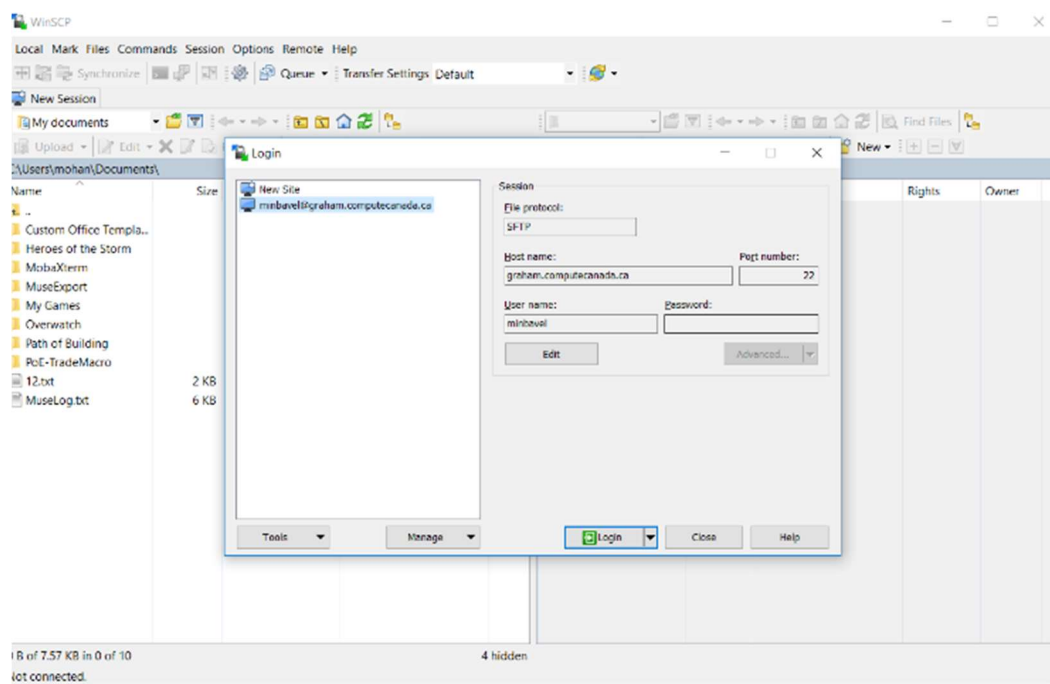


5.3 WinScp:

WinSCP is an open source SFTP and SCP client for Microsoft Windows. The main function of this is to securely transfer files from a local host to a remote computer.

SFTP, which stands for SSH File Transfer Protocol, or Secure File Transfer Protocol, is a separate protocol packaged with SSH that works in a similar way over a secure connection. The advantage is the ability to leverage a secure connection to transfer files and traverse the filesystem on both the local and remote system.

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.



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

5.4 Amazon Web Services:

Amazon Web Services (AWS) is a subsidiary of Amazon launched in 2006 built to handle its online retail operations. AWS was one of the first companies to introduce a pay-as-you-go cloud computing model that scales to provide users with compute, storage or throughput as needed.

Amazon Web Services provides services from dozens of data centers spread across availability zones (AZs) in regions across the world. An AZ represents a location that typically contains multiple physical data centers, while a region is a collection of AZs in geographic proximity connected by low-latency network links. An AWS customer can spin up virtual machines (VMs) and replicate data in different AZs to achieve a highly reliable infrastructure that is resistant to failures of individual servers or an entire data center.

Resources Available:

In AWS, a resource is an entity that you can work with Amazon EC2 instance, an AWS CloudFormation stack, or an Amazon S3 bucket.

Amazon EC2:

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing

requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate them from common failure scenarios.

Step 1: Choose an Amazon Machine Image (AMI) [Cancel and Exit](#)

AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Search:

Quick Start (0) | My AMIs (0) | AWS Marketplace (11) | **Community AMIs (3)**

Operating system

- ☐ Amazon Linux
- ☐ Cent OS
- ☐ Debian
- ☐ Fedora
- ☐ Gentoo
- ☐ openSUSE
- ☐ Other Linux
- ☐ Red Hat

AMI ID	AMI Name	Architecture	Root device type	Virtualization type	ENA Enabled
ami-09b8f2f441fc21b6f	RStudio-1.2.1335_R-3.6.0_CUDA-10.0_cuDNN-7.5.1_ubuntu-18.04-LTS-64bit	64-bit (x86)	ebs	hvm	Yes
ami-0ccb91fcd3d2460f3	RStudio-1.1.456_R-3.5.1_CUDA-9.0_cuDNN-7.2.1_ubuntu-16.04-LTS-64bit	64-bit (x86)	ebs	hvm	Yes
ami-75c17911	RStudio-1.1.383_R-3.4.2_Julia-0.6.0_CUDA-8_cuDNN-6_ubuntu-16.04-LTS-64bit	64-bit (x86)	ebs	hvm	Yes

6. Feature Description:

6.1 Parallel Processing:

With the rapid growth of technology, there are many high-level IT resources available, one of the most significant technology which provides these resources is Cloud.

Processing of Big data has become easier with the computation power provided by Cloud providers. Furthermore, we can increase the efficiency and productivity of the model if we use Parallel Processing. We can divide Big Data into multiple chunks and process them parallelly, with this the time required for computation is reduced, henceforth increasing the efficiency and the resources are used productively, without any wastage.

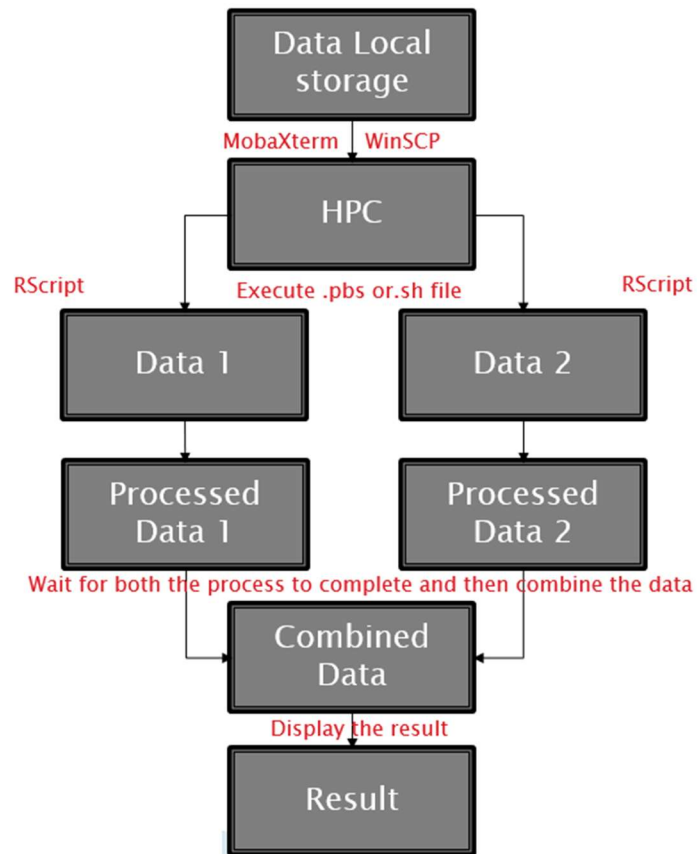
6.2 Clusters:

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 need to create clusters depending on the number of cores and the number of groups the data is split into.

7. Design and Methodology:

- Created 2 separate Cloud account on AWS & Compute Canada Cloud (CCC) to show the performance results between two vendors.
- R script are used to implement the parallel and serial computing in both the Cloud Environment.
- Latency are noted down for both parallel and serial computing to show the variation in the process
- The Basic idea of implementation for both processing is as follows:
 - Split the Data into pieces
 - Execute the pieces in parallel
 - Combine the results back together just by using R language.
- Implemented using RStudio where,
 - The R script (. R file) executed and time is noted down.
- We have used Clusters to split the data into multiple parts and sent those blocks to clusters and processed the data both parallelly and serially.



8. Implementation:

8.1 Data:

File: numbers.csv

Data points: 100000

Input: Random numbers between 0 to 100.

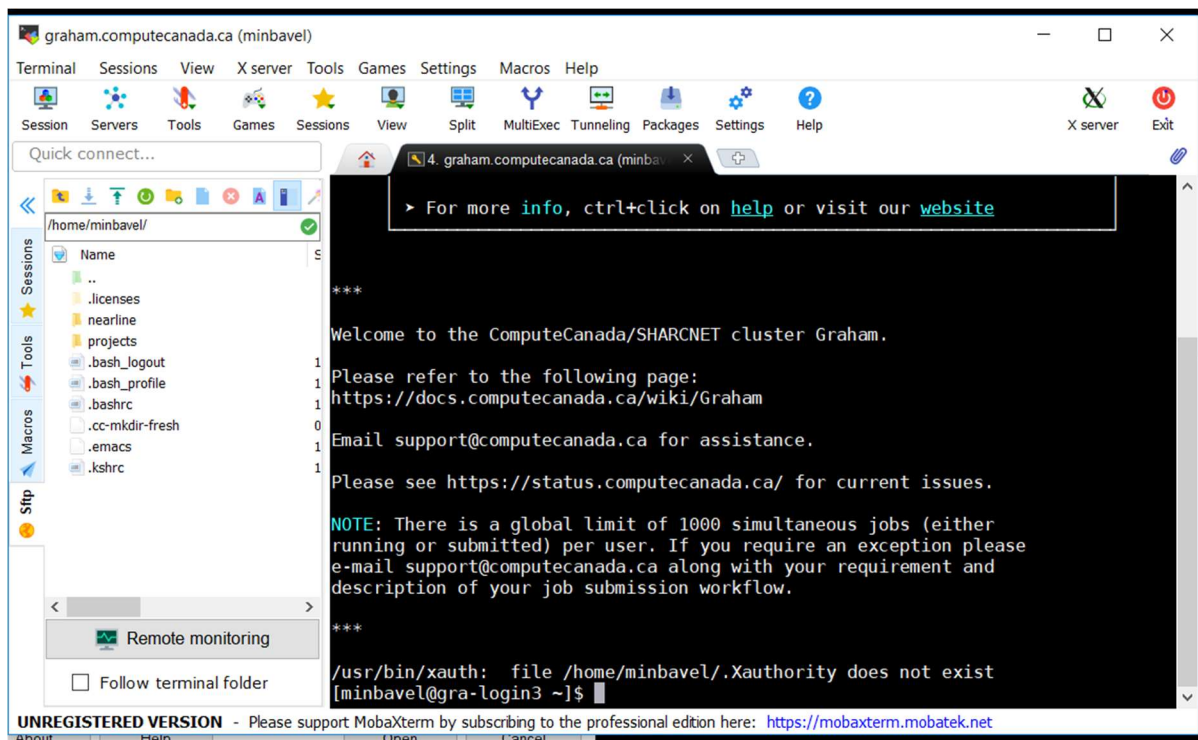
Output: < 51 is "1" and >51 is "0"

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.

8.2 *Implementation on Compute Canada Cloud:*

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



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.

The screenshot shows the MobaXterm application window titled 'graham.computeCanada.ca (minbavel)'. The interface includes a menu bar (Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, Help), a toolbar with icons for Session, Servers, Tools, Games, Sessions, View, Split, MultiExec, Tunneling, Packages, Settings, Help, X server, and Exit. A 'Quick connect...' search bar is at the top left. On the left sidebar, there are sections for Sessions, Tools, and Macros. The main terminal pane shows the following text:

```
> install.packages("dplyr")
Installing package into '/home/minbavel/R/x86_64-pc-linux-gnu-library/3.5'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
Secure CRAN mirrors
```

1: 0-Cloud [https]	2: Algeria [https]
3: Australia (Canberra) [https]	4: Australia (Melbourne 1) [https]
5: Australia (Melbourne 2) [https]	6: Australia (Perth) [https]
7: Austria [https]	8: Belgium (Ghent) [https]
9: Brazil (PR) [https]	10: Brazil (RJ) [https]
11: Brazil (SP 1) [https]	12: Brazil (SP 2) [https]
13: Bulgaria [https]	14: Chile 1 [https]
15: Chile 2 [https]	16: China (Hong Kong) [https]
17: China (Guangzhou) [https]	18: China (Lanzhou) [https]
19: China (Shanghai 1) [https]	20: China (Shanghai 2) [https]
21: Colombia (Cali) [https]	22: Czech Republic [https]
23: Denmark [https]	24: East Asia [https]
25: Ecuador (Cuenca) [https]	26: Ecuador (Quito) [https]
27: Estonia [https]	28: France (Lyon 1) [https]
29: France (Lyon 2) [https]	30: France (Marseille) [https]
31: France (Montpellier) [https]	32: France (Paris 2) [https]
33: Germany (Erlangen) [https]	34: Germany (Göttingen) [https]
35: Germany (Münster) [https]	36: Greece [https]
37: Iceland [https]	38: India [https]

At the bottom of the terminal window, it says: **UNREGISTERED VERSION** - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

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`

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]
[1,] 37.07558
```

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
```

- **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 shard each 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
```

- **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_ cluster_get(cluster, name) 15", fun5)
```

- **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,]
1,] 7.699522
```

Results:

The Time taken for Serial Processing

```
Time differences in secs  
[ ,1]  
[1,] 37.07558  
>
```

The Time taken for Parallel Processing using 5 clusters

```
Time differences in secs  
[ ,1]  
[1,] 7.699522  
>
```

We compared the time elapsed for serial processing and the time elapsed for parallel processing.

With the results, it is evident that parallel processing is better than serial processing for Big Data as the time elapsed for serial processing is 37 seconds and the time elapsed for parallel processing is 7 seconds if we split the data into 5 parts.

The Time taken for Parallel Processing using 10 clusters

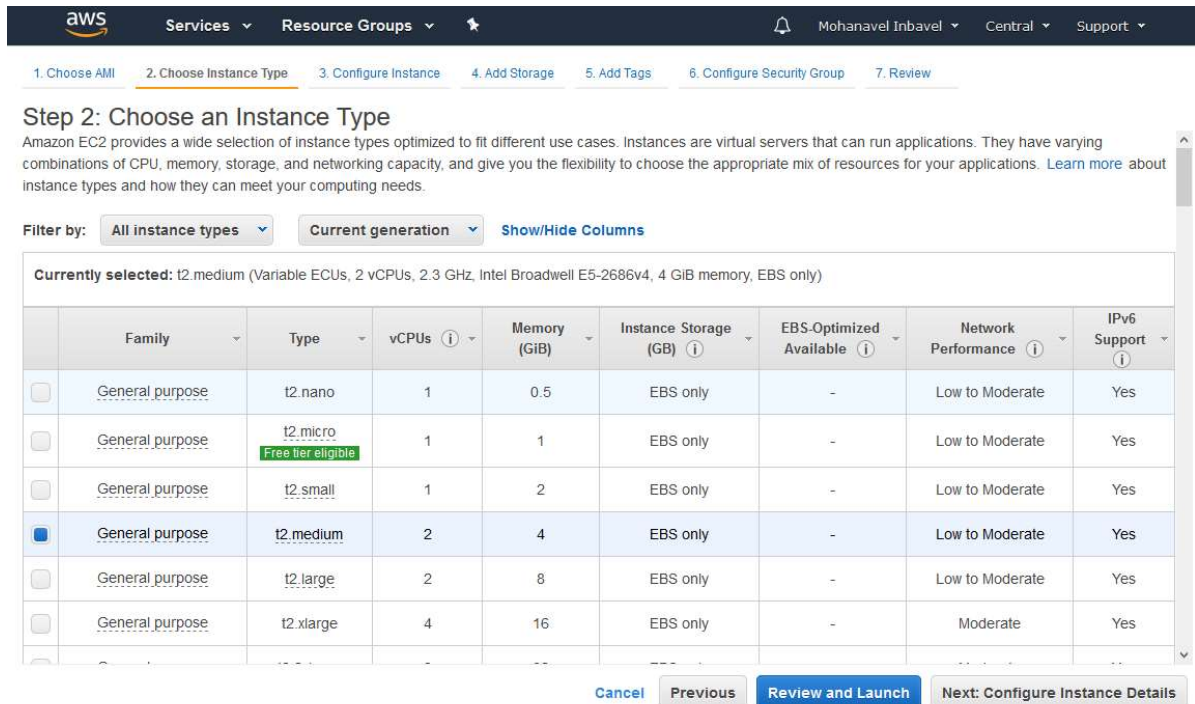
```
Time differences in secs  
[ ,1]  
[1,] 4.665699  
>
```

The processing time will decrease even more if we split the data into more parts, we split our data into 10 parts and executed it, the time elapsed is 4 seconds. We can conclude that parallelizing code can drastically improve speed on multi-core machines.

8.3 Implementation on Amazon Web Services:

Instance Type

When you launch an instance, the *instance type* that you specify determines the hardware of the host computer used for your instance.



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: **All instance types** **Current generation** [Show/Hide Columns](#)

Currently selected: t2.medium (Variable ECUs, 2 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 4 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Configure Instance Details](#)

Each instance type offers different compute, memory, and storage capabilities and are grouped in instance families based on these capabilities. Select an instance type based on the requirements of the application or software that you plan to run on your instance.

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP F	TCP	80	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP F	TCP	8787	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop

[Add Rule](#)

Warning

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

[Cancel](#) [Previous](#) [Review and Launch](#)

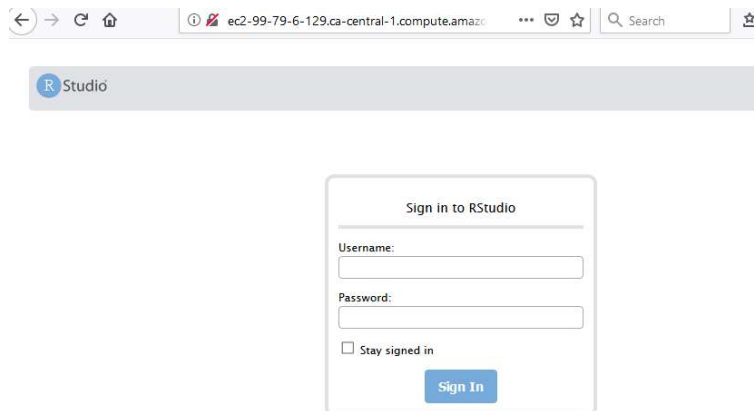
We need to add custom TCP Rule so that the instance will be available in public for access. For HTTP traffic, add an inbound rule on port 80 from the source address 0.0.0.0/0. For HTTPS traffic, add an inbound rule on port 8787 from the source address 0.0.0.0/0. These inbound rules allow traffic from IPv4 addresses. To allow IPv6 traffic, add inbound rules on the same ports from the source address ::/0.

Instance: **i-0b8a13cc8c52b83ef** Public DNS: **ec2-99-79-6-129.ca-central-1.compute.amazonaws.com**

Description		Status Checks		Monitoring		Tags	
Instance ID	i-0b8a13cc8c52b83ef	Public DNS (IPv4)	ec2-99-79-6-129.ca-central-1.compute.amazonaws.com				
Instance state	running	IPv4 Public IP	99.79.6.129				
Instance type	t2.micro	IPv6 IPs	-				
Elastic IPs		Private DNS	ip-172-31-9-148.ca-central-1.compute.internal				
Availability zone	ca-central-1b	Private IPs	172.31.9.148				
Security groups	launch-wizard-5, view inbound rules , view outbound rules	Secondary private IPs					
Scheduled events	No scheduled events	VPC ID	vpc-0d903a4b00c2be754				
AMI ID	RStudio-1.2.1335_R-3.6.0_CUDA-10.0_cuDNN-7.5.1_ubuntu-18.04-LTS-64bit (ami-09b8f2f441fc21b6f)	Subnet ID	subnet-0c45a26149583b1ef				
Platform	-	Network interfaces	eth0				
IAM role	-	Source/dest. check	True				
Key pair name	mohan	T2/T3 Unlimited	Disabled				

The description of the instance which is created is available, it provides all the details of the instance like security groups, public IPs, private IPs etc. these are required for the user for access and modification of the instance.

RStudio Login:



Sign in to RStudio

Username:

Password:

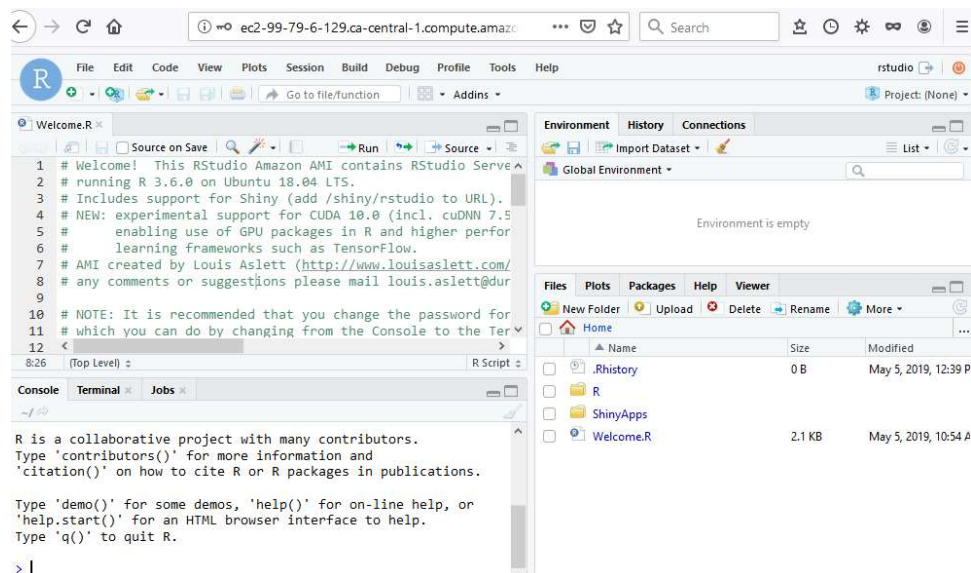
☐ Stay signed in

Sign In

Since we created from an Image that is already available in AWS the username is rstudio and the password the instance ID.

Execution of Code:

Here we execute the R Script.



```
1 # Welcome! This RStudio Amazon AMI contains RStudio Server
2 # running R 3.6.0 on Ubuntu 18.04 LTS.
3 # Includes support for Shiny (add /shiny/rstudio to URL).
4 # NEW: experimental support for CUDA 10.0 (incl. cuDNN 7.5)
5 # enabling use of GPU packages in R and higher performance
6 # learning frameworks such as TensorFlow.
7 # AMI created by Louis Aslett (http://www.louisaslett.com/)
8 # any comments or suggestions please mail louis.aslett@durham.ac.uk
9
10 # NOTE: It is recommended that you change the password for
11 # which you can do by changing from the Console to the Terminal
12 >
```

8:26 (Top Level) R Script

Console

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Environment History Connections

Global Environment

Environment is empty

Files Plots Packages Help Viewer

Name	Size	Modified
.Rhistory	0 B	May 5, 2019, 12:39 P
R		
ShinyApps		
Welcome.R	2.1 KB	May 5, 2019, 10:54 A

Results:

The Time taken for Serial Processing

```
Time differences in mins  
[ ,1]  
[1,] 1.162119
```

The Time taken for Parallel Processing using 2 clusters

```
Time differences in secs  
[ ,1]  
[1,] 33.72222
```

9. Future Research Work:

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.

10. Conclusion:

The results of the project clearly demonstrate 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.