Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons

Wong Zi Xiang

SESSION 2017/2018

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
FEBRUARY 2018

Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons

BY

Wong Zi Xiang

SESSION 2017/2018

THIS PROJECT REPORT IS PREPARED FOR

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
IN PARTIAL FULFILLMENT
FOR
BACHELOR OF COMPUTER SCIENCE (HONS)
WITH SPECIALIZATION IN
DATA SCIENCE
FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY

FEBRUARY 2018

Declaration

I hereby declare that the work in this thesis have been done by myself and no portion of the work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Name: *Wong Zi Xiang*
Student ID: 1142701102
Faculty of Computing & Informatics
Multimedia University

Acknowledgements

First and foremost, I have to thank my supervisor Mr. Wan Ruslan Yusoff, who has guided and supported me throughout this project. Without his assistance and guidance, it would be impossible to accomplish this project. He is one of the most kind, helpful and passionate lecturers I have met in my university life. I am very blessed to have him for his support and guidance over the course of this project. I am honored and grateful to have him as my supervisor for this project. Every meeting would take an average of a few hours and my friends are surprised because their meetings with supervisor are only less than an hour. But I am grateful to have spent so much time discussing and listening to my supervisor because through the lengthful meetings, I have a clear direction of what I should do next. The life experiences and stories he shared to me are also very educative and insightful.

The most important lesson I have learned from my supervisor is "start small, but right". Before this, I am always greedy and try to do all things at once, which in turn caused me to be counterproductive.

I would also like to express my gratitude to all my friends and family for their support and encouragement. I am also grateful to my partner who stayed by my side and supported me when I am stressed out. Last but not least, I would like to thanks my previous schools for providing me a chance to achieve what I had today.

Contents

Declaration iii

Acknowledgements iv

Contents v

List of Figures xi

List of Tables xiv

Listings xv

Abbreviations and Acronyms xvi

Abstract xvii

1 Introduction 1
- 1.1 Introduction . . . 1
  - 1.1.1 Project Brief Description . . . 2
  - 1.1.2 Project Objectives . . . 5
  - 1.1.3 Project Motivations . . . 5
- 1.2 Project Scope . . . 7
  - 1.2.1 Phase 1 Scope of Work . . . 7
  - 1.2.2 Project Deliverables for Phase 1 . . . 7
  - 1.2.3 Phase 2 Scope of Work . . . 7
  - 1.2.4 Project Deliverables for Phase 2 . . . 8

2 Literature Review 9
- 2.1 Concurrent Computing . . . 9
- 2.2 Parallel Computing . . . 11
- 2.3 Distributed Computing . . . 12

2.4 Concurrent versus Parallel Computing 14
2.5 Processes - PID and PPID 16
2.6 Multithreading 18
2.7 Multiprocessing 19
2.8 Problems in Concurrent and Parallel Computing 20
2.8.1 Race Condition 21
2.8.2 Deadlock 22
2.8.3 Livelock 22
2.8.4 Starvation 23
2.9 Concurrency Control 25
2.9.1 Mutual Exclusion 25
2.9.2 Semaphore 27
2.9.3 Barrier 27
2.9.4 Blocking Algorithm 28
2.9.5 Non-Blocking Algorithm 29
2.9.6 Condition Variable 29
2.9.7 Monitor 30
2.10 SSHFS 31
2.11 Magnetic Hard Disk Drive's Limitation on Parallel I/O 32
2.12 Big Data 34
2.13 R-language 36
2.13.1 RStudio 37
2.14 Python 38
2.14.1 PyDev 39
2.15 Parallel Programming in R and Python 39
2.15.1 R in Data Science 40
2.15.2 Python in Data Science 40
2.15.3 Parallel and HPC with R 41
2.15.4 Combining Python and R 42
2.15.5 Cross Language Tools 42
2.15.6 FlashR 43
2.16 Semi-structured Data 44
2.17 XML Parser 44
2.17.1 Simple API for XML (SAX) 44
2.17.2 Document Object Model (DOM) 45
2.18 MongoDB 46
2.19 Performance Analysis and Monitoring 46
2.19.1 LTTng 47
2.19.2 GNU Debugger 47
2.19.3 Common Trace Format 47

2.19.4 Trace Compass 48
2.20 Chapter Conclusion 48

3 Project Design Proposal **49**
3.1 Phase 1 49
3.1.1 Introduction 49
3.1.2 Context Diagram 50
3.1.3 Flowchart 51
3.1.4 Project data 52
3.1.4.1 stackoverflow 52
3.1.4.2 softwareengineering 52
3.1.4.3 datascience 52
3.1.5 Project Implementation Plan 53
3.1.5.1 Data Accounting 53
3.1.5.2 Data conversion and uploading to MongoDB . . . 54
3.1.5.3 Decision on program objective 54
3.1.5.4 Serial Processing on Project Data 54
3.1.6 Parallel Processing on Project Data 55
3.2 Phase 2 56
3.2.1 Introduction 56
3.2.2 Flowcharts 57
3.2.2.1 DOM and SAX Parse PostLinks.XML 57
3.2.2.2 Local SAX Parse PostHistory.XML and Posts.XML Sequentially 58
3.2.2.3 Local SAX Parse PostHistory.XML and Posts.XML Concurrently 59
3.2.2.4 Local SAX Parse PostHistory.XML and Posts.XML in Parallel 60
3.2.2.5 Remote SAX Parse PostHistory.XML and Posts.XML Sequentially 61
3.2.2.6 Remote SAX Parse PostHistory.XML and Posts.XML Concurrently 62
3.2.2.7 Keyword Search Using SAX Parser 63
3.2.3 Project Implementation Plan 64
3.2.3.1 DOM and SAX Parse PostLinks.XML 64
3.2.3.2 Local SAX Parse PostHistory.XML and Posts.XML Sequentially 65
3.2.3.3 Local SAX Parse PostHistory.XML and Posts.XML Concurrently 66
3.2.3.4 Local SAX Parse PostHistory.XML and Posts.XML in Parallel 66

3.2.3.5 Remote SAX Parse PostHistory.XML and Posts.XML Sequentially . . . 67
3.2.3.6 Remote SAX Parse PostHistory.XML and Posts.XML Concurrently . . . 68
3.2.3.7 Keyword Search Using SAX Parser . . . 69

4 Implementation Methodology 70
4.1 Software Engineering Methods . . . 70
4.1.1 Prototyping Model . . . 71
4.1.2 Incremental Model . . . 72
4.1.3 Methodology for this Project . . . 73
4.2 Project Infrastructures . . . 74
4.2.1 Hardware Specifications . . . 74
4.2.2 List of Software Resources . . . 74
4.2.3 Other Project Resources . . . 75
4.2.3.1 R - Installed Packages . . . 75
4.2.3.2 Python - Installed Packages . . . 76
4.2.4 Infrastructure Setup and Installation . . . 76
4.2.4.1 R . . . 77
4.2.4.2 RStudio . . . 77
4.2.4.3 Python . . . 77
4.2.4.4 PyDev . . . 77

5 Implementation Plan 79
5.1 Project Task Identification . . . 79
5.1.1 Identification and Mitigation Measures of Critical Tasks . . . 79
5.1.2 Project Tasks for Phase 1 . . . 82
5.1.3 Project Tasks for Phase 2 . . . 83
5.1.4 Gantt Chart for Implementation Schedule . . . 84
5.1.5 Milestone Deliverables . . . 85
5.2 Planned Execution Activities . . . 85

6 Results and Findings 86
6.1 Project Data Preparation . . . 87
6.2 Program Codes . . . 89
6.3 Phase 1 . . . 91
6.3.1 Data accounting and Performance Assessment . . . 91
6.3.2 Serial Processing on Project Data . . . 91
6.3.3 Parallel Processing on Project Data . . . 92
6.3.4 Execution Time Comparison . . . 93
6.4 Phase 2 . . . 93

6.4.1 Comparison between DOM and SAX Parse 93
6.4.2 Keyword Search Engine 96
6.4.3 Wrong Method in Parallel Processing Project Data in Phase 1 96
6.4.4 Performance Comparison of Parallel, Concurrent and Sequential SAX Parse 98
6.4.5 Performance Comparison of Local and Remote SAX Parse 100
6.4.6 Performance Comparison of SAX Parse on R and Python . 102
6.4.7 Discussion on Results and Findings 103

7 Discussions 106
7.1 Achievements 106
7.2 Problems Encountered & Overcoming Them 107
7.2.1 New Programming Paradigm 107
7.2.2 Unfamiliar Data Format 108
7.2.3 Circumventing MongoDB 108
7.2.4 Trace Compass 109
7.2.5 Corrupted Project Data on Clusterrocks Server 110

8 Conclusions 111
8.1 Conclusions 111
8.2 Lessons Learned 114
8.3 Recommendations for Future Work 115

Appendices 121

A Data Structure of Project Data 121

B Source Codes 124
B.1 xml_accounting.py 124
B.2 xml-to-json-mongodb.py 125
B.3 votetypeid-serial.py 126
B.4 votetypeid-parallel.py 127
B.5 dom-parse-PostLinks.py 129
B.6 sax-parse-PostLinks.py 130
B.7 python-sax-parse.py 130
B.8 r-sax-parse.r 132
B.9 keyword-search.py 133
B.10 sequential-concurrent-parallel-sax-plot.py 137
B.11 local-vs-remote-sax-plot.py 139
B.12 r-vs-python-sax-plot.py 141

C R Installation **144**

D RStudio Installation **147**

E PyDev Installation **149**

F Technical Specifications **154**
F.1 Personal Computer 154
F.2 Clusterrocks Server 159
F.3 1 TB HGST HTS541010A9E680 HDD 166
F.4 500 GB ATA ST500LT012-1DG14 HDD 167

G Mapping Clusterrocks File System to Local Machine **168**

H Keyword Search Engine **170**

I System Monitor during Remote SAX Parse **172**

J Meeting Logs **173**

List of Figures

2.1 Multi-tasking on a single CPU machine. 10
2.2 Parallel Computing Diagram 12
2.3 Distributed System Diagram 13
2.4 Parallel Computing and Concurrent Computing 15
2.5 Race Condition 21
2.6 Deadlock 22
2.7 Dining Philosopher's problem 24
2.8 Mutual Exclusion 26
2.9 Barrier's overview 27
2.10 Blocking algorithm diagram 28
2.11 Non-blocking algorithm diagram 29
2.12 Structure of monitor 30
2.13 SSHFS VPN 31
2.14 HDD Components 32
2.15 Platter Components 33
2.16 Screenshot of R-language 36
2.17 Screenshot of RStudio 37
2.18 Screenshot of Python 38
2.19 Screenshot of PyDev 39
2.20 SAX Parser Overview 44
2.21 DOM Parser Overview 45
2.22 Screenshot of MongoDB Shell 46
2.23 Screenshot of Trace Compass 48

3.1 Context Diagram 50
3.2 Parallel Program Flowchart 51
3.3 Data accounting 53
3.4 DOM & SAX Parse PostLinks.XML Flowchart 57
3.5 Local Sequential SAX Parse PostHistory.XML & Posts.XML Flowchart 58
3.6 Local Concurrent SAX Parse PostHistory.XML & Posts.XML Flowchart 59
3.7 Local Parallel SAX Parse PostHistory.XML & Posts.XML Flowchart 60

3.8 Remote Sequential SAX Parse PostHistory.XML & Posts.XML Flowchart 61
3.9 Remote Concurrent SAX Parse PostHistory.XML & Posts.XML Flowchart 62
3.10 Keyword Search SAX Flowchart 63

4.1 Prototyping Model 72
4.2 Incremental Model 73

5.1 Gantt Chart Schedule 84

6.1 Serial Processing Results 92
6.2 Parallel Processing Results 92
6.3 System Monitor during DOM Parse 93
6.4 DOM Parse Results 94
6.5 System Monitor during SAX Parse 95
6.6 SAX Parse Results 95
6.7 Performance Comparison of Parallel, Concurrent & Sequential SAX Parse 98
6.8 Performance Comparison of Local & Remote SAX Parse 100
6.9 Performance Comparison of R & Python SAX Parse 102

C.1 R terminal 146

D.1 RStudio Install 1 147
D.2 RStudio Install 2 148
D.3 RStudio Install 3 148

E.1 PyDev Install 1 149
E.2 PyDev Install 2 150
E.3 PyDev Install 3 151
E.4 PyDev Install 4 152
E.5 PyDev Install 5 152
E.6 PyDev Install 6 153

F.1 1 TB HDD Technical Specifications 166
F.2 1 TB HDD Benchmark 166
F.3 500 GB HDD Technical Specifications 167
F.4 500 GB HDD Benchmark 167

G.1 Cisco VPN 168

H.1 Keyword Search Engine 1 170
H.2 Keyword Search Engine 2 171

H.3 Keyword Search Engine 3 171

I.1 System Monitor Remote SAX 172

List of Tables

2.1 Example of a list of running processes 17

6.1 StackOverflow data dump 87
6.2 SoftwareEngineering data dump 88
6.3 DataScience data dump 88
6.4 Program source codes list 89
6.5 Performance Comparison 93

Listing

6.1 Data accounting and performance assessment on PostHistory.xml 91
A.1 Data structure of Project Data 121
B.1 Source code for XML data accounting 124
B.2 Source code for XML-JSON conversion and upload to MongoDB . 125
B.3 Source code for serial data processing 126
B.4 Source code for parallel data processing 127
B.5 Source code for DOM parse PostLinks.XML 129
B.6 Source code for SAX parse PostLinks.XML 130
B.7 Source code for Python SAX parse program 130
B.8 Source code for R SAX parse program 132
B.9 Source code for keyword search program 133
B.10 Source code for plotting performance comparison of parallel, concurrent and sequential SAX parse line graph 137
B.11 Source code for plotting performance comparison of local and remote SAX parse line graph 139
B.12 Source code for plotting performance comparison of R and Python SAX parse line graph 141
C.1 R Installation Steps 144
F.1 Personal computer technical specifications 154
F.2 Clusterrocks server technical specifications 159
G.1 Instructions to map clusterrocks directory to local machine 169

Abbreviations and Acronyms

API	Application Programming Interface
BSON	Binary JavaScript Object Notation
CTF	Common Trace Format
CSV	Comma Separated Value
DOM	Document Object Model
GDB	GNU Debugger
HDD	Hard Disk Drive
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
LTTng	Linux Trace Toolkit next generation
SAX	Simple API for XML
SDLC	Software Development Life Cycle
SSHFS	Secure Shell File System
SFTP	SSH File Transfer Protocol
XML	Extensible Markup Language

Abstract

The main goal of this project is to improve the computation times in data processing for the R and Python programming languages by utilizing parallel programming techniques. The goals include implementing and leveraging various parallel computing strategies. The performance between the two languages are then compared.

In this project, we processed three large XML datasets, comprising [***datascience*** *(70 MB)*], [***softwareengineering****(1.3 GB)*] and [***stackoverflow*** *(185 GB)*] data. We used R and Python program scripts that implement parallel and distributed programming strategies for computation speed ups.

We implemented SAX (Simple API for XML) processing instead of DOM (Document Object Mode) on the humongous XML files in this project because DOM processing requires memory several times the size of XML files. Thus, it is impossible to use DOM processing as the machine used in this project only have 8 GB memory. SAX processing requires almost no memory to process large XML files as it does not load them into memory.

Processing time is very important in data analytics. It would be very unproductive to be waiting hours, days or even months for the results. Not to mention time-sensitive analysis. So, we must achieve a reasonable duration for completion of data processing. In this project, we found that parallel and distributed processing improved the processing time of large XML files significantly.

We executed multi-threading SAX processing on an exactly same XML file multiple times and saw increase in data processing speed. This experiment is wrongly setup because when we did multi-threaded SAX processing on real dataset (several different XMLs), we saw a decrease in data processing speed. This is because the machine in this project only had single hard disk drive, thus the performance is limited by the single read head as the disk need to spin back and forth to serve multiple read on multiple XML files. In this project, we proved that data processing speed is improved when we did parallel SAX processing on multiple HDD.

We analysed large XML files in this project, but depending on the types of data and its source format, the method to use in Big Data analysis must be approached

differently on a case by case basis. We found the methods of processing in this project for our XML can be improved by adding more HDD, parallel process data on separate machines and combine the results upon completion and increasing the buffer size of SAX parser.

It is important to note that, as computer science students, our primary focus in this project is on computing techniques and time speed ups. Even though we will be processing big data, the value of the data and its output are of secondary importance.

Chapter 1

Introduction

1.1 Introduction

In layman's term, parallel computing can be defined as *distributing workloads* to *multiple workers* for them to work on *simultaneously*[1], thus increasing its overall productivity. The purpose of distributed computing is also the same.

Both parallel and distributed computing originated from the core idea of "Divide and Conquer". Parallel computing is more about dividing tasks into multiple threads or processes to have them processed simultaneously and returning their results when completed. Distributed computing is more about dividing and executing separate tasks on multiple machines and then merging their results.

The boundary between these two concepts is blurry, especially for Big Data. It is hard to draw a line between the parallel and distributed concepts because Big Data nowadays are usually processed both in parallel, and at the same time in a distributed network of computers, where each computer, with its own individual memories, may have multiple-processors or multiple-cores.

In order to conduct data processing, there must be suitable software tools. Two programming languages, R and Python, will be used in this project to execute parallel processing of data. Both languages are suitable for data analytics. Both also come with a lot of libraries and packages that are well-suited to deal with data-related processing. We expect that speed performance is better in Python because memory access in R is slower. Also in R, every operation carries more overhead than Python. While both language are hundreds time slower than compiled language like C/C++, it would be a very slow development and usually data scientist need to build prototype model rapidly which is extremely difficult in C/C++.

In this project, we will implement parallel and distributed programming in R and Python for big data processing and analysis.

1.1.1 Project Brief Description

For this project, we will conduct simulated parallel and distributed programs to speed up processing times on our selected big data, three large XML datasets, comprising:
(a) datascience (70 MB),
(b) softwareengineering (1.3 GB),
(c) stackoverflow (185 GB).
Even though our programs will be realistic, we consider them as simulated because the value of the program outputs is our secondary goal. Our primary goal is to capture and compare running program performances.

To process XML data, we have a choice between DOM or SAX processing strategies. SAX is an event-driven online algorithm for parsing XML documents, where we process (by parsing or reading) XML data on the fly, without loading the entire XML data into memory. On the other hand, DOM requires the entire XML data be loaded into memory before processing. We will be using the SAX processing strategy for this project. Details on SAX and DOM processing will be covered in Chapter 2.17.

Both R and Python programming languages have features that support parallel and distributed processing. Both R and Python programming languages have packages to process XML files. In addition, Python codes can be embedded in Rscripts, and vice versa, Rscripts can be embedded in Python codes. In this project, we will write and execute Rscript programs using the RStudio IDE and Python programs using the Eclipse-PyDev IDE.

Our XML data files will be converted into JSON data files and then stored in MongoDB. Both R and Python programming languages have packages to process data in MongoDB. The document-oriented MongoDB can be implemented as a standalone or as a distributed database. Data also can be extracted and converted to suitable format for data processing in R and Python. Details on techniques for storing data in MongoDB will be covered in Chapter 3.1.5.2.

Debugging and tracing are very different activities. Debugging is about finding errors (bugs) and fixing the errors so that the program will run correctly and successfully. The focus in debugging is to trace a running program to find, fix errors and get it correct. Whereas, the focus on pure tracing is on capturing and recording the running performances (usually execution times) of a correctly

running program. It is futile and useless to conduct performance tracing for an incorrectly running program. The program may run but it is not giving the correct results or it is not performing or displaying behaviors according to its design. If the program is not running at all or crashed out, then it is about debugging not tracing.

When we debug, while the program is running, we "go inside the program and intervene to trace steps, view intermediate variable values, pause and step through the codes, and so on". Whereas, in performance tracing, we "stay out, only place minimum intervention using tracepoints into the codes", and allow the program to run until completion. In performance tracing, we usually have the running program write to outside files like log files, with date time stamps of key identified events, write variable values in logs, and so on. In this project, we focus on performance tracing using binary CTF (Common Trace Format) files, instead ot text ASCII-based log files. A CTF file is faster to write, it is binary, can capture a lot of events, and can get high resolution time traces, down to nanoseconds event tracing, and so on. CTF can also capture parallel programs, hardware CPU activities, and so on. Python and R can produce CTF files. In this project, we will use software tools like LTTng (Linux Trace Toolkit next generation), babeltrace, DDD tool, GDB (GNU Debugger), GUI GDB debugger and Trace Compass.

In this project, we will finally compare performances of serial programs with parallel programs written using R language and Python. The comparison programs shall produce the same outputs because it is not correct to compare apples against oranges. We will also provide some graphical outputs to show interesting results from our large datasets. We will discuss the results and make

recommendations based on our objectives and findings in this project.

1.1.2 Project Objectives

The objectives of this project are:

1. To learn and understand about the R and the Python programming language concepts and their parallel processing features.
2. To explore different techniques on data processing, parallel and distributed programming for big data.
3. To conduct performance comparisons between R and Python language parallel processing implementations for big data.
4. To implement the handling and conversion of big data into the MongoDB, a document-oriented, JSON-style distributed database.
5. To process combinations of data on multiple repositories in a network that represents a real time, parallel and distributed processing environment.

1.1.3 Project Motivations

I am very interested in parallel computing after one subject in my bachelor degree named Computer Architecture and Organization. I am fascinated by the works and achievements of the computer scientist throughout the century. I always wondered how did their brilliant mind come up with such sophisticated structures and theories. It is amazing how something from bit-level can be compiled into such wonderful system. Everything interacts perfectly, and the computers (mobile phone included) we are always using right now have

integrated into our life so much that it cannot be separated from our everyday life.

One day, I noticed that our processors have their clock rates stalled at the middle of 3 to 4 Gigahertz for a decade. I am curious and did some research and found out about what is Moore's Law and how it is dying. I am once again amazed by the computer scientists after this. When they faced bottleneck in trying to keep up with Moore's Law because of physical constraint in single core processor, they instead changed their approach to use multi-core processors. Nowadays, multi-core processors is the norm in computers' industry. Even our personal device like mobile phone have up to eight cores.

Later in my degree second year, I chose my specialization to be Data Science. From there, I learned about how parallelism is everywhere. I have learned that Big Data is always being processed in parallel by using framework like Apache Hadoop or otherwise it is an impossible feat. Many complex problems previously thought impossible was solved by utilizing parallel computing's ultra high computational speed.

When I saw this project on the final year project list, I knew I must choose it. I realized that parallel computing is the way to go. It will never go wrong to have parallel programming skill in the future. I knew this project will be challenging and interesting.

When I started this project, I realized I am lacking knowledge in this field. The school never taught us about parallel computing other than Flynn's taxonomy and multiprocessing in operating system. Although I have vague idea on how does parallel computing works, I have no practical experience with it. I have never tried to code my program for parallel execution. The learning curve is

also steeper than my previous work because I am not familiar with the tools needed for this project on top of needing to apply parallelism. However, the more challenging it is, the more motivated I am to overcome it.

1.2 Project Scope

1.2.1 Phase 1 Scope of Work

1. Understand and describe about parallel computing.
2. Learn how to perform data analytics on R and Python.
3. Learn how to program in parallel.
4. Implement parallel computing for data processing.

1.2.2 Project Deliverables for Phase 1

1. Running parallel processing and serial processing on test datasets.
2. Performance comparison between serial processing and processing as a proof of concept that data processing time is indeed shortened.

1.2.3 Phase 2 Scope of Work

1. Convert datasets into suitable format to be stored in MongoDB.
2. Implement parallel and distributed processing on real datasets.
3. Compare performance between R and Python on parallel and distributed data processing.

1.2.4 Project Deliverables for Phase 2

1. Data processing techniques on real time, parallel and distributed processing environment.
2. Performance comparison between R and Python on parallel and distributed data processing.
3. Generate a good statistical report from the processed data.
4. A report based on this project.

Chapter 2

Literature Review

2.1 Concurrent Computing

Concurrent computing techniques can be seen in operating systems that support multi-tasking, multi-threading and CPU time-sharing systems.

As an example, for the simple general purpose computer that has only a single CPU core, true parallel computing is not possible. We can run multi-tasks on the single CPU computer by making it a CPU-time-sharing system. Each task is allocated a short time-slice of the CPU. If a task is not finished within the allocated time-slice, the task is paused (pre-empted to stop) and the CPU moves to service the next task. The CPU repeats the process and provide every task a portion of service time. It will sequentially service all the tasks and when all scheduled tasks have been serviced, comes back to the first task. The cycle repeats.

Because the time-slice is very small (in milliseconds) and the CPU clock cycle is very high (in GigaHertz), this CPU time-sharing system gives an illusion that the many tasks are running at the same time, which in reality is not true because there is only one CPU. A single CPU can only process one task at any one time. The diagram below shows an example of multi-tasking, that is, running 3 tasks by sharing a single CPU-core computer.

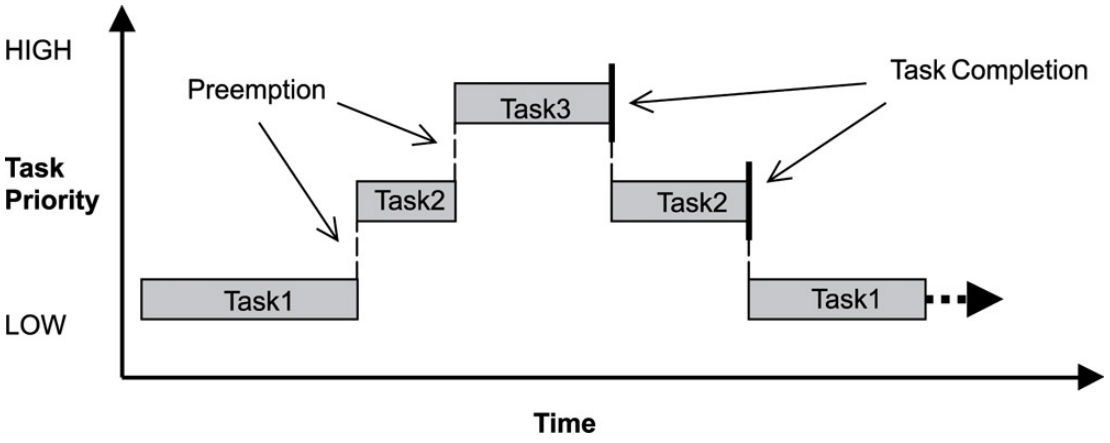


FIGURE 2.1: Three tasks running on time-sharing on a single CPU machine.

Concurrent computing is about scheduling many tasks to run, whether it is run by CPU time-sharing or each task truly run in parallel against each other is a separate matter. Parallel computing is discussed in the next section.

Concurrent computing can be computations where some tasks run sequentially in CPU time-sharing mode *(cases where the number of tasks are more than the number of CPU cores)* or run truly in parallel in an overlapping time period *(cases where the number of tasks are less than or exactly equal to the number of CPU cores).*

2.2 Parallel Computing

Parallel computing is a type of computation in which multiple computations are executed *simultaneously* in an overlapping time frame instead of *sequentially*.

Parallel computing means that execution of processes or calculations are carried out simultaneously[1]. This means that multiple tasks are being executed at the same time.

Parallel computing was mainly implemented in High Performance Computing (HPC) systems. Parallel computing becomes popular later with the development of multi-core and multi-processor computer systems. Because there is more than one CPU core, we can have true parallel computing, meaning, many tasks can truly run at the same time. The maximum number of tasks that can really run in parallel at any one time is equal to the number of physical CPU cores available.

The diagram below shows an overview of true parallel computing. The diagram shows an example for a system with N-CPUs. All of the N-tasks (problems) on the left will be processed in parallel (simultaneously) by the N-CPUs. Each CPU will be executing a single task.

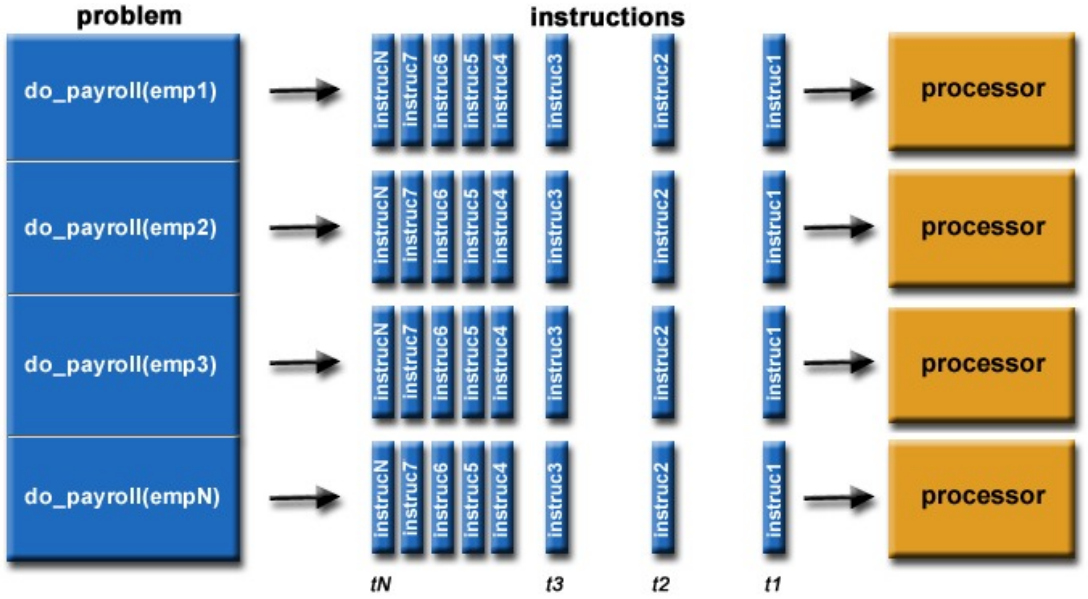


FIGURE 2.2: A Parallel Computing Diagram

2.3 Distributed Computing

Distributed computing is a type of computation where networked computers coordinate and communicate with each other by passing messages. A distributed computing system comprises multiple machines, but run as a single system. The machines in a distributed computing system can be nearby physically and connected by local area network (LAN), or they can be spread across large geographic areas and connected by wide area network (WAN). The purpose of distributed computing is to make these machines work as a single system.

There are many advantages provided by distributing computing, the main advantages are:

Scalability

The system can be scaled easily by adding more machines.

Redundancy

The system is reliable as other machines can takeover in case a machine failed.

Parallel and concurrent computing is not to be confused with distributed computing. Concurrent computing becomes parallel computing when processes or threads execute on several different processors. Thus, parallel computing can be said to be a subset of concurrent computing.

When the CPUs used to handle processess or threads are on the same machine, it is referred as *parallel computing*; When the CPUs reside on different machines, which may be spread out geographically, it is referred as *distributed computing*.

Therefore, distributed computing is a subset of parallel computing, which is a subset of concurrent computing.

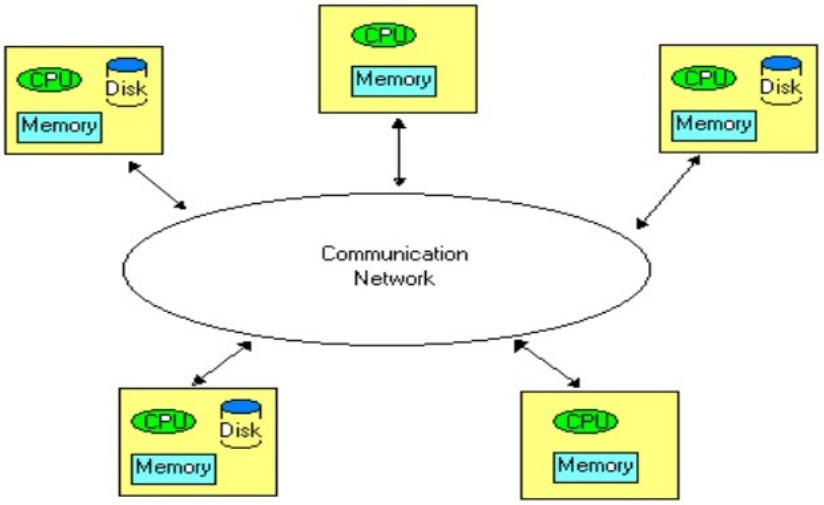


FIGURE 2.3: A Distribtued System Diagram

2.4 Concurrent versus Parallel Computing

Parallel computing and concurrent computing are frequently confused with each other. Both concepts are related but at the same time distinct.

This example will hopefully clear the confusion. Consider an example of a quad core machine (4-cores), and 6 tasks to run. We can schedule these 6 tasks to be run concurrently. But what really happens is, we only run 4 tasks in parallel during the first cycle. The balance of 2 tasks will be run in parallel during the second cycle. In the second cycle, we have 2 cores idle.

From the above example, we see that concurrent computing can involve parallel computing. However, parallel computing is not necessarily concurrent computing.

We have also seen that concurrent computing can involve CPU time-sharing (multi-tasking) computing.

The diagram below shows exactly the difference between parallel computing and CPU time-sharing computing. On the left, each CPU is running one task each, at the same time. This is true parallel computing.

On the right, three tasks are sharing one CPU. The CPU time-sharing computing on the right is considered concurrent computing. It gives the illusion to users that the three tasks are running at the same time, but in real fact, is running sequentially on just one CPU.

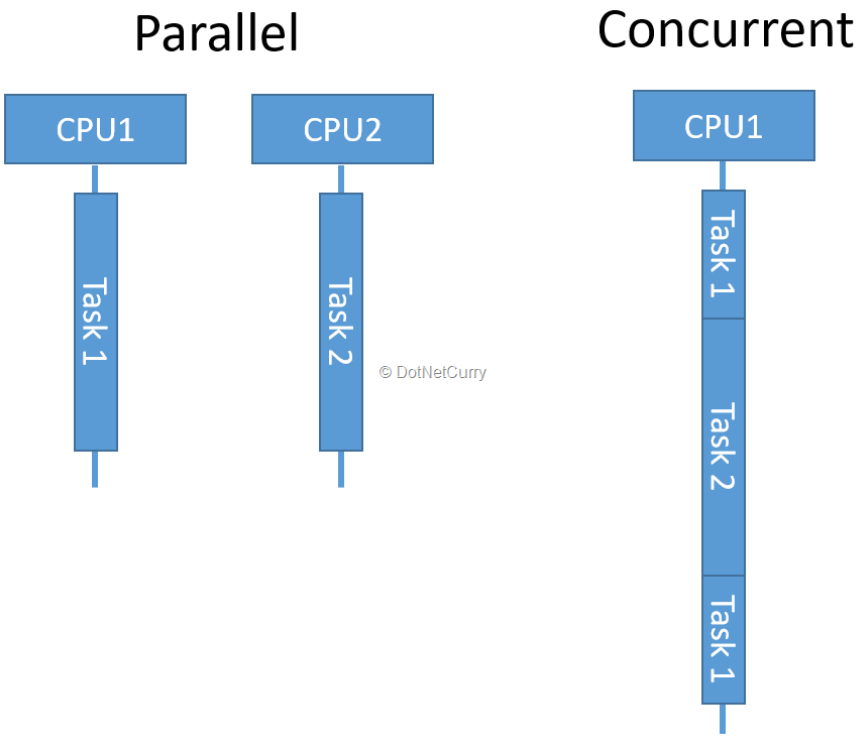


FIGURE 2.4: Parallel Computing and Concurrent Computing.

2.5 Processes - PID and PPID

We will discuss PID (Process ID) and PPID (Parent Process ID) because these two concepts are directly related to parallel and concurrent programming.

In the Linux operating system (as in other operating systems), a process is the smallest executable unit that can be executed by the CPU. Many processes run on a computer, and each process is given a identification. For example, when we run "ps -ef" on a linux terminal, we are listing the current list of running processes on the computer. An example is shown in the table below.

The following explains the column entries in the table of running processes.

1. Row = Row number for reference. Each Row is a specific running process.
2. UID = The effective user ID of the process's owner.
3. PID = Process ID number for the specific process.
4. PPID = Parent Process ID number for the specific process.
5. C = The processor utilization for scheduling.
6. STIME = The starting time of the process (hours, minutes, seconds).
7. TTY = The terminal from which the process (or its parent) was started. A question mark indicates there is no controlling terminal.
8. TIME = Total amount of CPU time used by the process since it began.
9. CMD = The command that generated the process.

We read the relationships between UID, PID and PPID because these concepts play a direct role in parallel and concurrent programming.

Table 2.1: Example of a list of running processes

Selected List of Running Processes								
Row	**UID**	**PID**	**PPID**	**C**	**STIME**	**TTY**	**TIME**	**CMD**
1	root	1	0	0	Aug22	?	00:00:03	/sbin/init splash
2	root	1592	1	0	Aug22	?	00:00:03	/usr/sbin/apache2 -k start
3	data	12021	1592	0	07:35	?	00:00:18	/usr/sbin/apache2 -k start
4	data	12022	1592	0	07:35	?	00:00:18	/usr/sbin/apache2 -k start
5	root	1271	1	0	Aug22	?	00:00:00	/usr/sbin/gdm3
6	root	1900	1271	0	00:11	?	00:00:00	gdm-session-worker [pam/gdm-password]
7	xiang	1917	1900	0	00:11	tty2	00:00:00	/usr/lib/gdm3/gdm-x-session –run-script /usr/lib/gnome-flashback/gnome-flashback-metacity
8	xiang	1931	1917	0	00:11	tty2	00:00:00	/sbin/upstart –user
9	xiang	17062	1931	1	14:39	?	00:06:46	texstudio /Documents/FYP-WRY/FYP1-LEWIS-ReportTemplate/main.tex

Example Interpretation No 1: The parent process ID for Row 9 (running TeXstudio) is Row 8 (upstart by user). Then, the parent process ID for Row 8 is Row 7 (the Metacity theme). Then, the parent process ID for Row 7 is Row 6 (GUI login session). Then, the parent process ID for Row 6 is Row 5 (GUI Gnome Daemon Manager). Then, the parent process ID for Row 5 is Row 1 (/sbin/init splash display). Everything stops at Row 1 which is the parent of all processes in the computer. Notice that Row 1 does not have a parent process ID, because it is the first starting process when the computer boots up. Row 1 conceptually, is similar to Prophet Adam PBUH who was created and have no parents.

Example Interpretation No 2: Notice that both Row 4 and Row 3 are the same process (starting Apache web server) that shares a common parent process ID, that is, Row 2 (also starting Apache Web server). The parent process ID for Row 2 is none other than Row 1 (/sbin/init splash screen display), the parent of all processes on the computer.

From both interpretations, we can see that a process can create another process, or can create a few other processes. The entire structure is like a very large "graph tree" starting with Row 1 (/sbin/init splash) as its root.

2.6 Multithreading

Multithreading is a concept where a single process creates a few other light weight processes called threads. The parent process is the master thread, and its spawns (creates) one or more child threads. All the threads, including the master thread, run in parallel (subject to the number of available CPU cores explained in the previous sections). This is the connection of processes and threads in parallel computing.

In multithreading, when one child thread completes its assigned task, it joins the master thread. All other child threads continue running independently until completion and later join the master thread (master process). However, if the master thread dies (for some reason) while the child threads are still running, then the whole group of threads gets pulled-in and dies together. This is not the case for processes in multiprocessing.

Multithreading is a shared memory paradigm, which is generally a **shared-everything** environment. This means all memory space, variables

values, settings, etc, in the multithreading environment is shared by the master thread and all its child threads. Every thread has read and write access to this environment. Because they share everything there is no need to pass messages among the threads including with the master thread. This is not the case for processes in multiprocessing.

Our Example Interpretation No 2 is actually the case of multithreading. If we kill Row 4 with a child thread with PID 12022, then there is no problem, because Row 3 (another child thread) and Row 2 (the parent thread) will continue running unaffected. However, if we kill Row 2 (the parent thread), then Row 3 and Row 4 will get pulled-in and die together.

2.7 Multiprocessing

Multiprocessing is a concept where a single process creates a few other processes. The parent process is the master process and its spawns (creates) one or more child processes. All the processes, including the master process, run in parallel (of course, subject to the number of available CPU cores explained in the previous sections). This is the connection of processes in parallel computing.

In multiprocessing, when one child process completes its assigned task, it can just stop or join the master process, depending on code design. All other child processes continue running independently until completion. If the master process dies (for some reason), the running child processes are not affected. Every child process in multiprocessing is independent and will continue running until its own completion. This is not the case for threads in multithreading.

Multiprocessing is a **shared-nothing** environment. This means every child process including the master process have their own separate and independent environments, that contain all memory spaces, variables values, settings, etc,. Because processes share nothing, they cannot directly access each others environment. In order to communicate among the child processes, including with the master process, there is the need to pass messages among the processes. One common method is using MPI (Message Passing Interface). This is not the case for threads in multithreading.

2.8 Problems in Concurrent and Parallel Computing

Although parallel and concurrent computing will increase the performance of a machine, but problem arise when things become more complex and processes are sharing the same resources.

The problems are:

1. Race condition
2. Deadlock
3. Livelock
4. Starvation

These problems will be discussed in subsections below.

2.8.1 Race Condition

A *race condition* occurs when 2 instructions from different threads access the same memory location, at least one of these accesses is a write and the threads are not using any exclusive locks to control their accesses to that memory. Race condition are hard to identify as no error will occur but instead the results is not correct or accurate.

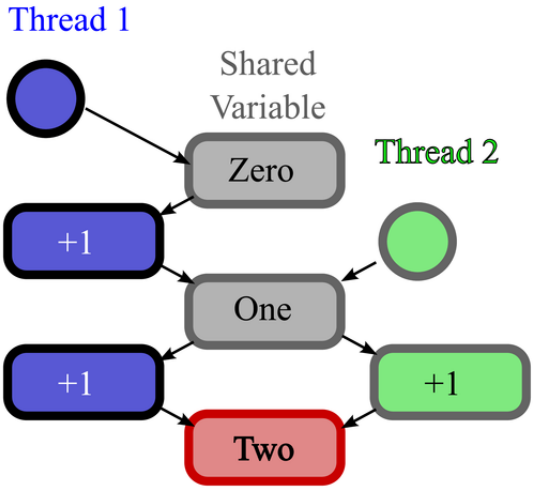


FIGURE 2.5: Race Condition

As we can see from Figure 2.4, *Thread 1* read from the shared variable and add one to it which is $0 + 1 = 1$. *Thread 1* then write 1 to shared variable. *Thread 2* read the shared variable this time, which is 1. At the same time, *Thread 1* also read from shared variable which is also 1. Both threads also increment their value and update the shared variable simultaneously. This cause the shared variable to be 2 but when the intended result should be 3.

2.8.2 Deadlock

A *deadlock* occurs a chain of processes where each of them are holding a locked resource and a trying to acquire a locked resource held by the next element in the processes chain. In a real life scenario, it would be two cars going in the opposite direction a single lane road, both waiting each other to give way but there are cars behind both cars, thus forming a *deadlock*.

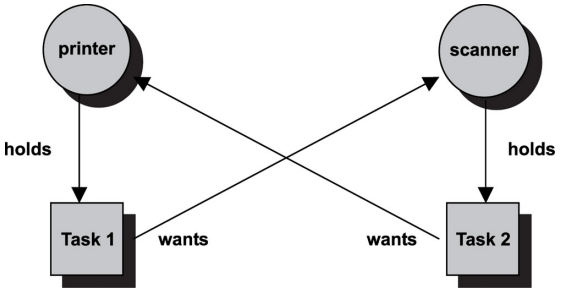


FIGURE 2.6: Deadlock

From Figure 2.5, we can see that Task 1 is using Printer and Task 2 is using scanner. Both of them will not release their resource until their task is finished. In order to finish their task, they require enough resource but both tasks are waiting for each other to finish their tasks and release the resources. Thus, indefinite waiting occurs. Such condition is called *deadlock*.

2.8.3 Livelock

A livelock is almost identical to deadlock, apart from the processes' states in livelock are changing constantly with respect to one another, none progressing. A good example of livelock would be two person coming from opposite direction politely give way to each other but awkwardly give way at the same side.

Livelock usually occurs when algorithm detects deadlock and tries to rollback or recover from it, but more than one process takes action and repeatedly trigger the rollback algorithm. Thus causing state changing but with no progress.

2.8.4 Starvation

Starvation occurs when a process is indefinitely denied required resources to finish its execution.

It happens when:

1. Processes pass resources to other processes without control, starvation may occurs because overall resource requirement of the system is not considered.
2. Processes with lower priority have to wait indefinitely because there are constant stream of processes with higher priority.
3. Random selection is used to determine resource allocation. Arbitrary allocation of resources might cause some processes to wait for a very long time.
4. Not enough resources for the system. Processes have to wait infinitely long as not enough resources are in the system.

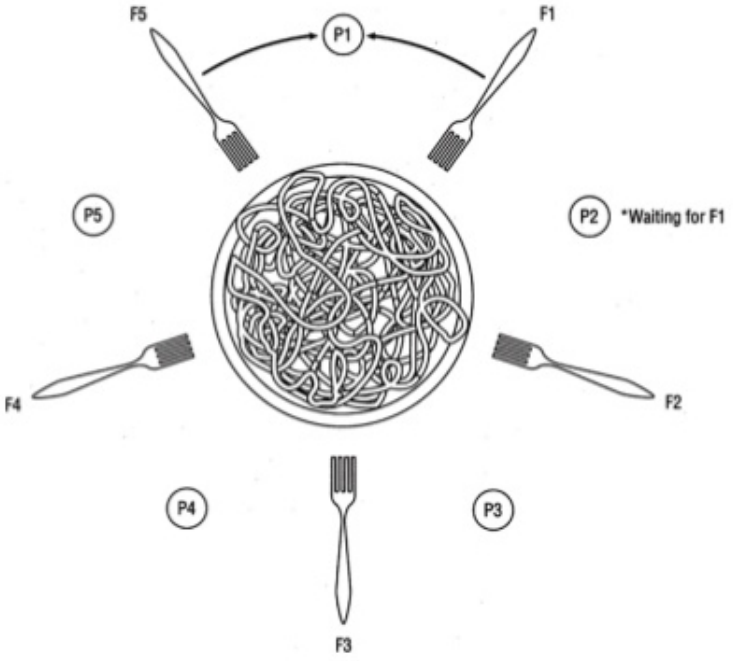


FIGURE 2.7: Dining Philosopher's problem

Each philosophers must have both left and right forks to begin eating, they do not interact with each other, only one fork may be picked up at one time and are only released when finished eating. Unless the resources (forks) are allocated fairly, some philosophers may starve.

2.9 Concurrency Control

Concurrency control are protocols to ensure that concurrent operations yield correct results and getting those results as soon as possible. If concurrency control is not implemented, problems such as deadlock, livelock, race condition and starvation will occurs in concurrent and parallel operations.

Below are lists of concurrency control:

1. Mutual Exclusion
2. Semaphore
3. Barrier
4. Blocking and Non-Blocking Algorithm
5. Condition Variable
6. Monitor

2.9.1 Mutual Exclusion

Mutual exclusion, or mutex for short, is a type of concurrency control intended to prevent race condition. Mutex addresses the issue on how can a system manage processes' access to shared resource. It is implemented based on the concept of ***critical section***. Critical section is a section where shared resources are protected and cannot be executed by more than one process.

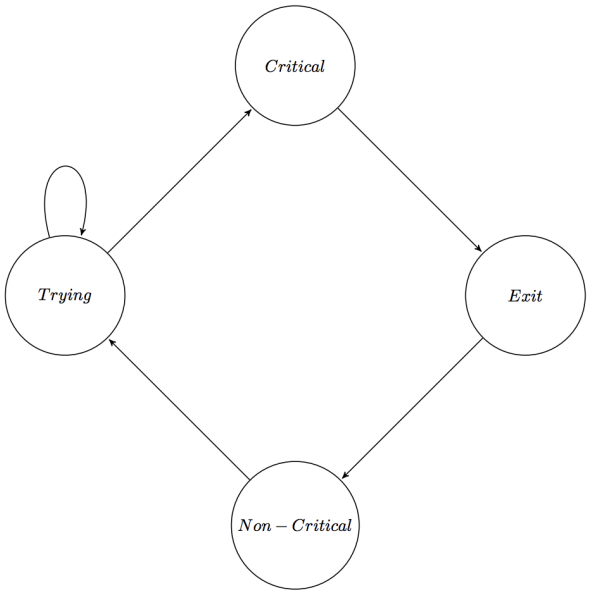


FIGURE 2.8: Mutual Exclusion

From Figure 2.7, we can see four states:

Non-Critical Section. Operation is outside the critical section; the process is not using or requesting the shared resource.

Trying. The process attempts to enter critical section.

Critical Section. Processes in this section are allowed to access the shared resource.

Exit. The process leaves the critical section and makes the shared resource available to other processes.

2.9.2 Semaphore

A semaphore is a variable or abstract data type function as access control of shared resources in a concurrent system.

There are two types of semaphore:

Counting Semaphore. Records how many resources are available and wakes up sleeping process when the required resources are free.

Binary Semaphore. Used to implement lock which is designed to enforce a mutual exclusion.

2.9.3 Barrier

A barrier is a synchronization model in parallel computing. It forces processes to stop at designated point and cannot continue executing until all other processes reach this barrier.

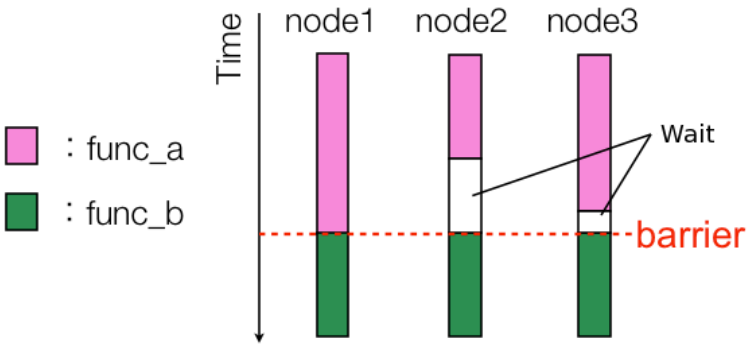


FIGURE 2.9: Barrier's overview

From Figure 2.8, we can see that node1 cannot proceed from func_a to func_b as barrier has blocked it from proceeding until node2 and node3 has reached the barrier.

2.9.4 Blocking Algorithm

An algorithm is called blocking algorithm if failure or suspension of any process will cause failure or suspension of another process.

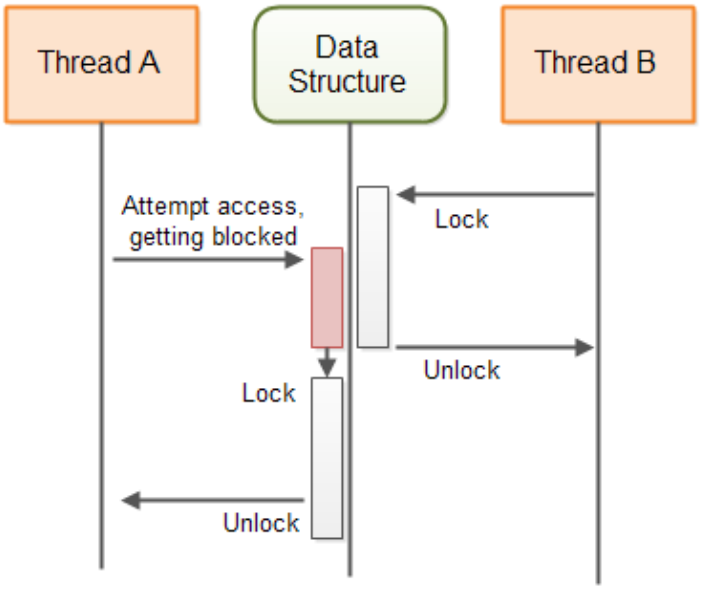


FIGURE 2.10: Blocking algorithm diagram

As we can see in Figure 2.9, when Thread B is using the shared resource, Thread A was blocked when it attempted to access the shared resource until Thread B finished its execution.

2.9.5 Non-Blocking Algorithm

An algorithm is called non-blocking algorithm if failure or suspension of any process will not cause failure or suspension of another process.

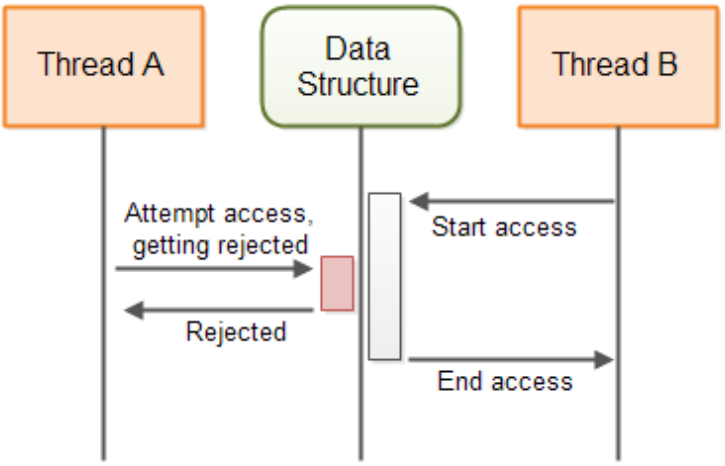


FIGURE 2.11: Non-blocking algorithm diagram

From Figure 2.10, we can see that when Thread B is accessing the shared resource, Thread A simply got denied access to the shared resource instead of getting blocked.

2.9.6 Condition Variable

The functionality of condition variable is to allow processes to communicate and acquire the condition of a shared resource. Thus processes have to wait until they are signaled by other processes that some condition are met.

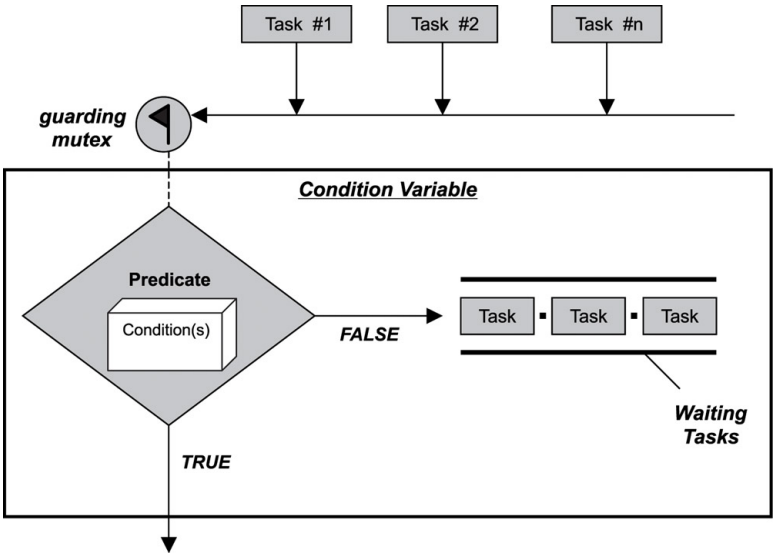


FIGURE 2.12: Structure of monitor

From Figure 2.11, processes evaluate the predicate, if the evaluation is true, it continues execution, else it waits for other tasks to create the desired condition.

2.9.7 Monitor

Monitor allows processes to have mutex ability and wait for required condition to come true. It also has mechanism to allow processes to signal other processes that their condition is met.

As we can see from Figure 2.11, tasks first lock the mutex and evaluate the condition variable, if condition not met, the task have to wait until other task signal to it that the resource is in free condition.

2.10 SSHFS

If we were to perform distributed processing on multiple files on multiple machines, a remote file system is required. SSHFS (Secure Shell File System) is a file system client that enables mounting and interaction with files and directories resided on remote server or workstation via SSH File Transfer Protocol (SFTP), a network protocol which provides file transfer, management and access function over data stream.

SFTP provides secure remote file system and file transfer, but it does not have the capability to mount remote file system locally. In order to mount file system locally, we need to use SSHFS so that remote file systems may be treated as other local volumes (e.g hard drives, DVDs, flash drives).

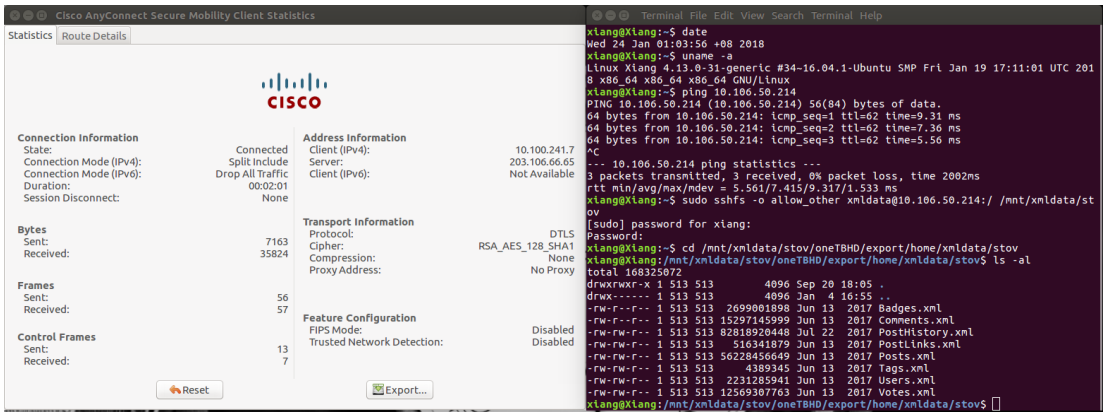


FIGURE 2.13: Mounting remote directory through VPN

2.11 Magnetic Hard Disk Drive's Limitation on Parallel I/O

Magnetic hard disk drive is comprised of a set of spinning platters where data are stored, a spindle that spins the platters, read/write heads, a set of disk arms that move the heads in an arc shape along the radius of the disk, and an actuator that pivots the disk arms. Hard disk drives utilize the read/write heads to access data stored on the spinning platters.

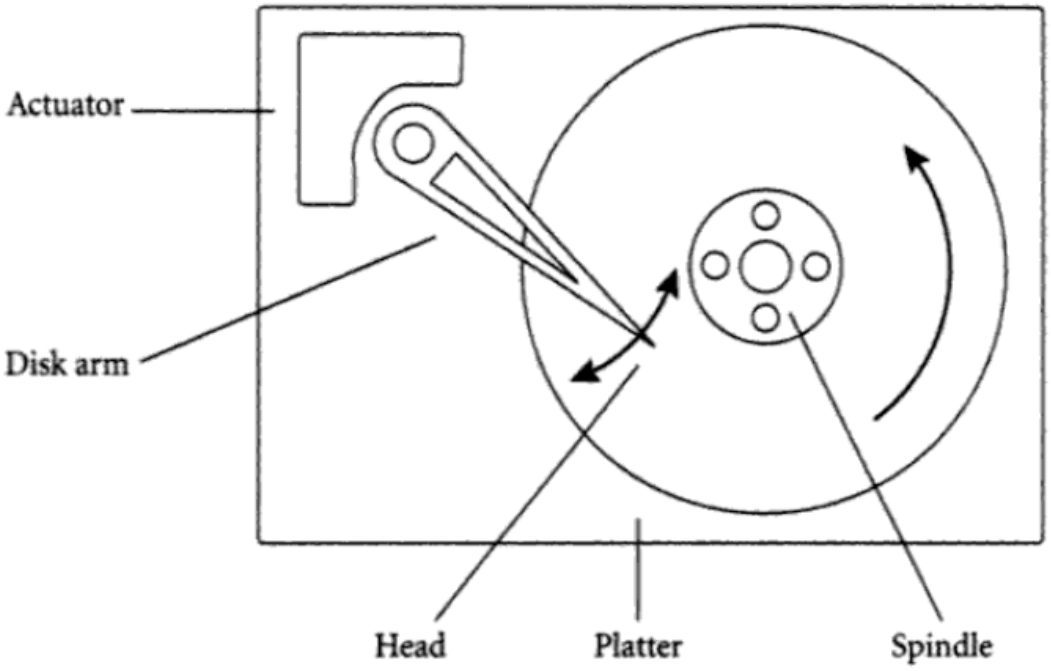


FIGURE 2.14: Major components of magnetic hard disk drive

In order to have a better understanding on why parallel I/O performance is limited on a single magnetic hard disk drive, let us take a look at *Figure 2.15* below:

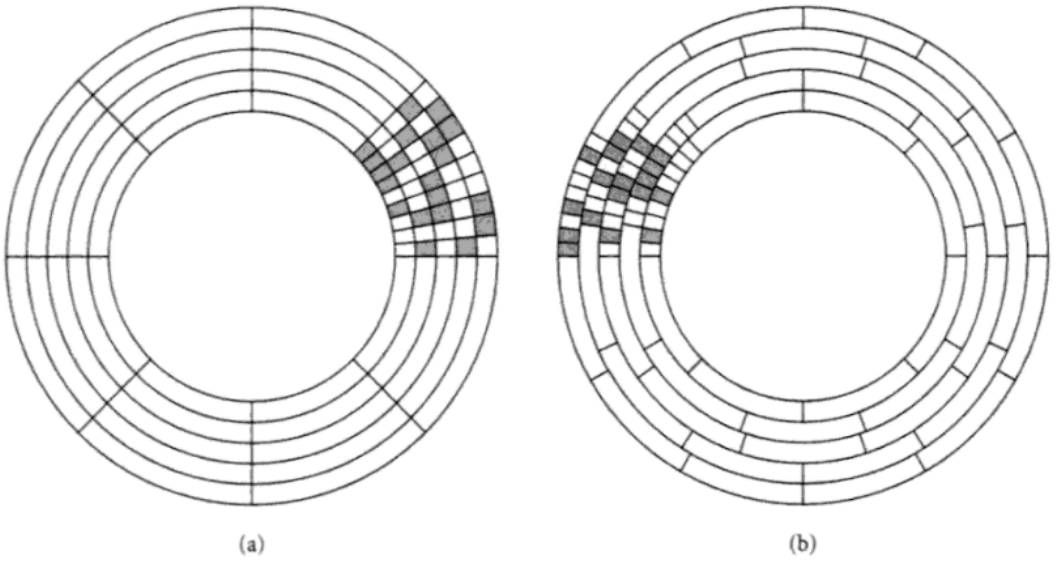


FIGURE 2.15: Structure of a magnetic hard disk drive's platter

Magnetic hard disk drive's platter comprise several layers of concentric rings which we call *tracks*. Each track are made up of several sectors which contain a fixed number of bits. Disk sectors are sometimes called blocks. But both should not be confused to be the same. Blocks refer to fixed-size unit of data that an operating system handles in I/O operation, whereas sectors refer to the physical region of a magnetic hard disk's platter's track.

There are two designs of sectors on tracks as shown in Figure 2.15. Figure 2.15(a) shows a design where all tracks contain the same amount of sectors which affect the length of the sectors (outer tracks has longer sectors than inner tracks), which in turn caused data to be stored less densely on the outer tracks, thus wasting space. Figure 2.15(a) shows a more efficient design where all sectors in all tracks are of the same length (outer track has more sectors than inner tracks) by increasing the number of sectors on the outer tracks.

If we were to read multiple files on a single magnetic hard disk drive (parallel read), it is expected there will be no significant improvement on performance (i.e double read speed). This is because the read head will jump back and forth as the files are stored on different part of the platter and it just become *concurrent* read and thus causing overhead by seeking multiple files on different sector on the platter, yielding *no real benefit* from doing parallel read on single hard disk drive.

2.12 Big Data

Big Data can be defined as data that is too enormous to be analyzed using conventional data analytics[2].

Big Data characteristics encompasses 3Vs:

1. **Volume.** Volume can be 'big' but the word 'big' is subjective. But the volume we are talking about here is so large that everyone will agree that it is 'big'. For example, Facebook have stored 250 billion photos of in 2015[3]. Assuming each photo is 100 KB, Facebook need to store 30 PB of photos per day. Imagine thirteen zeroes with a 3 in front, incredibly large.

2. **Velocity.** It means the speed of data generated. In Volume, we have talked about how Facebook have to store 250 billion photos, which is humongous number. Now imagine, each day they have to store additional 350 million photos, which is 35 TB per day.

3. **Variety.** It means the diversity of data format. For example, a sensor data, photos, videos, documents and much more. The data format of examples provided are incredibly varied.

2.13 R-language

R-language is an open source programming language and software environment specialized for statistical computing and graphics[4]. R is the most popular language for data analytic because it was created with data and statistics in mind. It also has the largest library dedicated for data analytic. It made data analysis such as data visualization, data manipulation, and machine learning to be more comfortable because of its large number of packages.

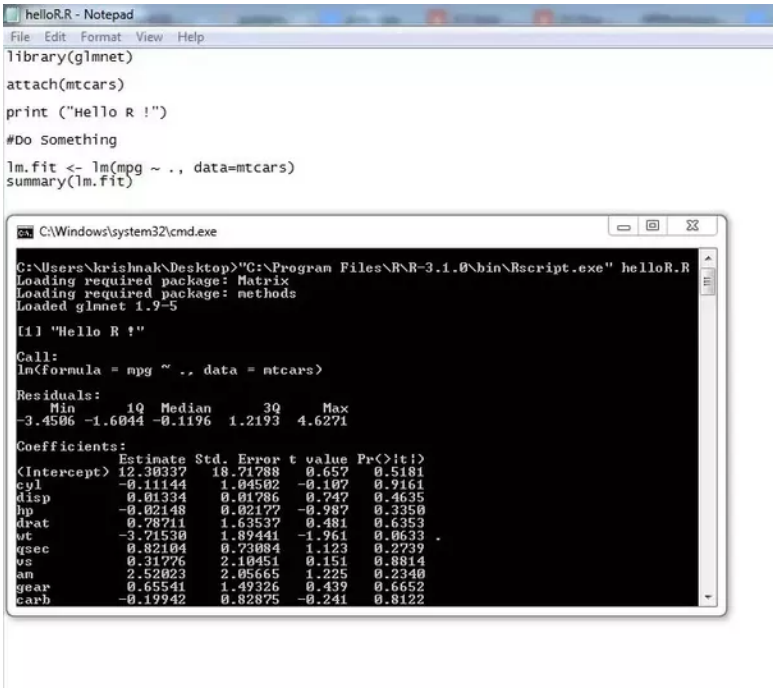


FIGURE 2.16: Screenshot of R-language

2.13.1 RStudio

RStudio is an integrated development environment (IDE) for R-language. It features a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management[5]. RStudio make it more comfortable and easier to work with R-language.

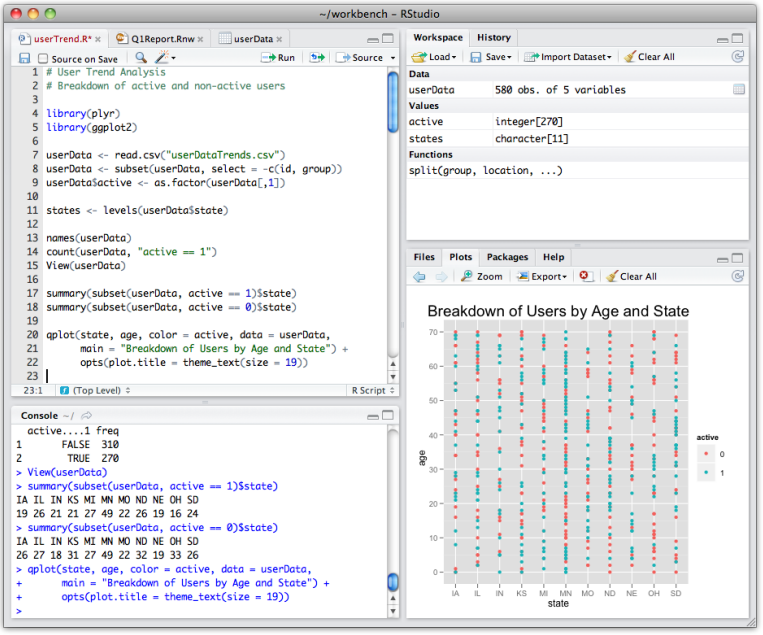


FIGURE 2.17: Screenshot of RStudio

2.14 Python

Python is an open source programming language used to perform general purpose programming. It features expansive libraries and very clean syntax. Python ranked number two in terms of world wide popularity for programming languages[6]. The reason why it is so popular is because nowadays, the field of software and applications very competitive. Thus, softwares and applications need to be updated and released rapidly. A high-level programming language such as Python is the primary choice for such task because the development time using Python is significantly faster, when compared to low-level language such as C/C++.

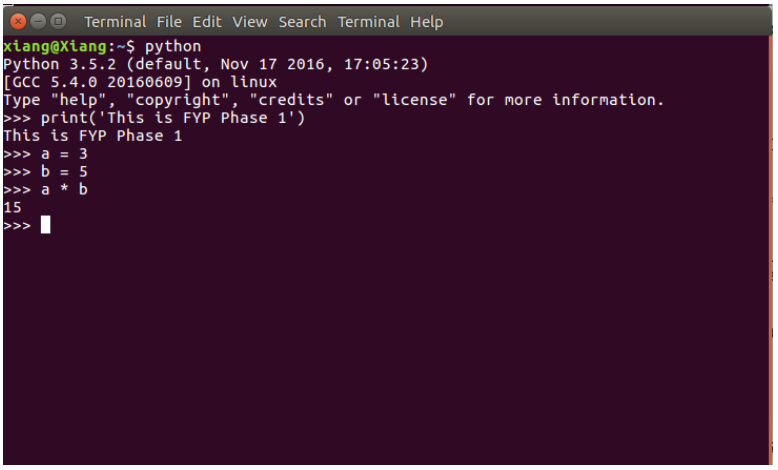


FIGURE 2.18: Screenshot of Python

2.14.1 PyDev

PyDev is a plugin that enables Eclipse to be used as IDE for Python. It features an interactive console, automatic code completion, debugging, refactoring, type hinting and many others[7]. PyDev make it more comfortable and easier to work with Python.

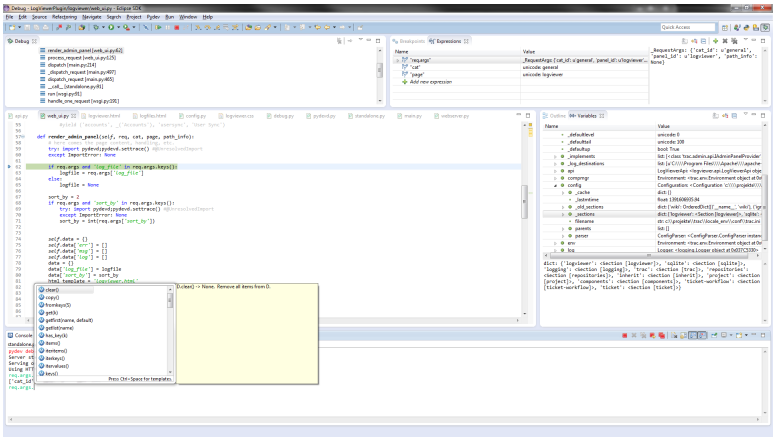


FIGURE 2.19: Screenshot of PyDev

2.15 Parallel Programming in R and Python

Parallelism is the same as all other ideas, they are born when there is a problem to be solved. For parallelism, it is born because time is precious. Data science involves a very demanding processing requirement and computational power is usually the bottleneck to humongous data. This is where parallel processing shines.

The top two most popular programming language that is used for data analytics is R and Python. Of course the most popular data analytics language would have the capability to implement parallel programming.

2.15.1 R in Data Science

R is the lingua franca of statistics[8]. It is developed by statisticians, for statisticians. It is easier for people without programming skills to use. It has become increasingly popular as its commercial utilities is known to more people because of its capabilities and it is open source.

For data analytics, it has hundreds of cutting-edge packages dedicated for statistics. Complex formulas can be used easily in R too. All kinds of statistical models and tests are available in R and can be used easily.

A picture says more than a thousand words. Data visualization is very important to aid in understanding the data. Pictures can show more than raw numbers alone. It is easy to plot pretty graphs and charts that in R using its packages.

2.15.2 Python in Data Science

Python is one of the most popular open source programming languages in the world. It has a good support for general purpose coding so it is very flexible. It is also the most popular programming language used for data analysis. It is usually used when data analysis needed to be integrated with web applications or if statistics code are required in production database.

Python fares quite well in data analytics. There are a lot of libraries dedicated to perform data analysis in Python. Its strength lies in ability to build products integrated with data analytics. It also has a rich data community offering features and toolkits.

Python is commonly used because it is a programming language that can be found in almost any organizations. Data scientists usually do not work alone, so it is preferred to use language that is more likely to found among the teams. It also has a large community of experts where you can ask for help when you face problems in data analysis.

2.15.3 Parallel and HPC with R

Two primary drivers for the growing interest in parallel and HPC (High Performance Computing) in R is large datasets and high computational requirement[9]. Both drivers is addressed via parallel computing.

Datasets can often be divided into smaller parts that can be analyzed in parallel. But the one part of the data must not depend on other part of data. Similarly, simulations for statistical models and tests that are independent of other simulations can be run in parallel. Thus, both drivers can be seen as embarrassingly parallel problems that are suitable for parallel processing.

There are a lot of packages in R that provide parallel computing related function such as explicit parallelism, implicit parallelism, grid computing and hadoop.

2.15.4 Combining Python and R

While R is better than Python at pure data analytics, Python is better than R in building practical data products. It is frustrating to do general purpose programming in a math language such as R. So the idea of leverage on the best of both worlds is born.

It is not uncommon for data scientist or engineers use both programming languages together. Early data analysis are done in R, then switching to Python when it is time to deliver data products. Python outperform R in areas of web scraping and database connections while R outperform Python in areas of statistical analysis options and interactive graphics/dashboard[10]. Although both languages are catching up to each other on their respective weakness, they still have not outperform each other on their respective weakness yet. Because of this issue, both language are commonly seen working together in a data science work flow.

2.15.5 Cross Language Tools

R is popular in some fields, Python is popular in other fields. A diverse team is usually polyglot. Important packages are often only available in one language thus a data science workflow often needs to use multiple languages. We have to know that different languages is optimized for different things. Python is a general purpose language while R is optimized for statistics.

Although most tools are only available in one language, there are also tools that are already cross-language such as XGBoost. XGBoost is a modeling tool available in both R and Python. It is written in native extensions

(C/C++).Since Python and R are both written in C, they can both use XGBoost through wrappers. The advantages of tools written in this way is fast and many languages are written in C. But it is difficult to write in low-level language and takes more code to do the same thing than in high-level language[11].

2.15.6 FlashR

R has became one of the most popular data analytic tools over the year. But R framework is relatively slow because it is a high-level language. A low-level language such as C/C++ is more efficient but they have lower productivity. In order to speed up an R implementation, algorithms are usually implemented in low-level language such as C or FORTRAN and then use a wrapper for R. Thus, from this idea FlashR is invented.

FlashR is a matrix-oriented R programming framework that provides implicit parallelization and out-of-core execution for large data. It re-implements many frequently used R functions in the base and stats packages so that it is similar to R to reduce learning curve. Also, it provides generalized matrix operations to implement more computations efficiently[12]. It is believed that FlashR can lowers the expertise for writing parallel and scalable implementations of machine learning algorithms significantly and provides new opportunities for large-scale machine learning in R.

2.16 Semi-structured Data

Semi-structured data (e.g., XML, JSON and HTML) is a type of structured data which does not comply with the formal structure of data models related to relational databases. But, it still has its own set of rules to abide by, albeit less strict and more flexible than the rules that structured data conform to. For example, a semi-structured data contains tags or other markers to segregate elements and enforce hierarchies of records and fields within the data.

2.17 XML Parser

2.17.1 Simple API for XML (SAX)

Simple API for XML (SAX) is an event-driven online algorithm for parsing XML files[13]. A SAX parser will only report the parsing event when its event handler is triggered. Thus, the memory required for SAX parser is very low. It also processes XML file faster because it does not need to load the whole XML file into memory.

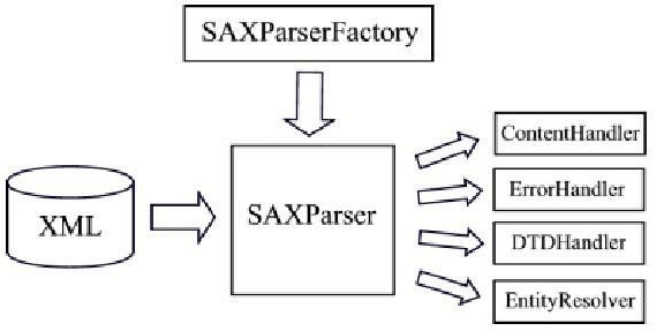


FIGURE 2.20: Simple API for XML (SAX) parser overview

2.17.2 Document Object Model (DOM)

Document Object Model (DOM) is an API that treats XML file as a tree structure where each node represent a part of the XML file[14]. Because DOM has to build a tree structure of the entire XML file in memory, it is generally very slow and time consuming. But the advantage is that once loaded, any part of the file can be accessed in any order. It is impossible to use DOM parser on enormously large file size as it requires the entire file to be loaded into memory.

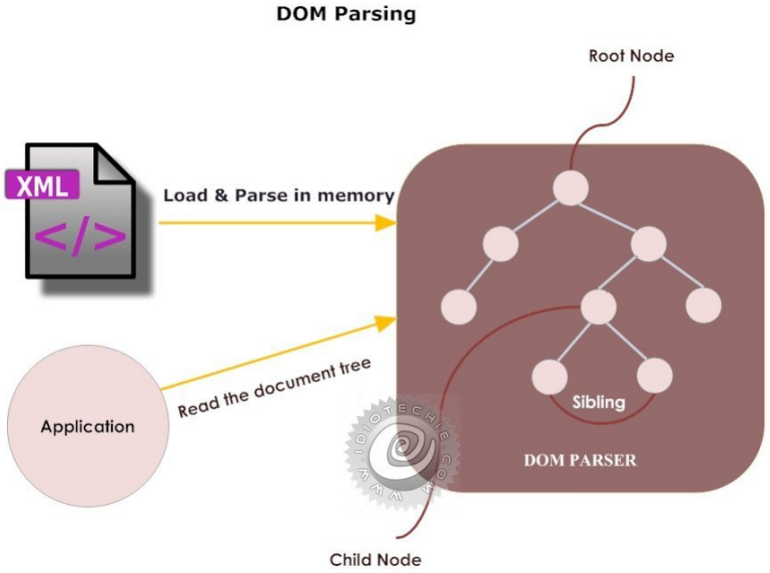


FIGURE 2.21: Document Object Model (DOM) parser overview

2.18 MongoDB

MongoDB is an open source document-oriented database. It is classified as NoSQL database as it stores file in flexible, JSON-like documents, namely BSON. It features ad hoc queries, real time aggregation and indexing for data analyzing and data access. Its core is a distributed database, so it is built in with functions such as horizontal scaling (by sharding), high availability (with replication) and geographic distribution[15].

FIGURE 2.22: Screenshot of MongoDB Shell

2.19 Performance Analysis and Monitoring

In order to analyze and monitor performance of parallel processes, they need to be traced. To trace the processes, runtime information need to be recorded without stopping the processes. Tools are needed to perform the tracing. There are two tools used for performance analysis and monitoring in this project : LTTng and GDB.

2.19.1 LTTng

LTTng is an open source tracing framework for Linux designed for minimal performance impact and debugging bugs that are extremely challenging without tracing[16]. It interacts to trace the Linux kernel and user applications, and to control tracing. The tracing output of LTTng is CTF.

2.19.2 GNU Debugger

The purpose of GDB is like any other debugger, to look into a program and see what happened when it crashed or output is unintended[17]. GDB can be used to debug program written in C, C++, Fortran and Modula-2. A debugger is needed because it can do what a conventional debugging method, using print function, cannot do. It can change variables' value at run-time, pause the program temporarily and much more.

2.19.3 Common Trace Format

CTF is a binary trace format designed to write quickly without sacrificing great flexibility[18]. It is extremely quick in writing output because it is in binary format. It allows traces to be natively generated by any C/C++ application or system. We will use tracing software that can read CTF file because both R and Python is written in C. In another word, R and Python can generate CTF file.

2.19.4 Trace Compass

Trace Compass is an GUI for viewing and analyzing any type of logs and traces, including LTTng's. It provides functionalities such as graphs, metrics and more to obtain meaningful insights from traces, in a way that is more human readable and informative than simply text dumps[19]. Trace Compass can only read and convert, CTF but not write CTF.

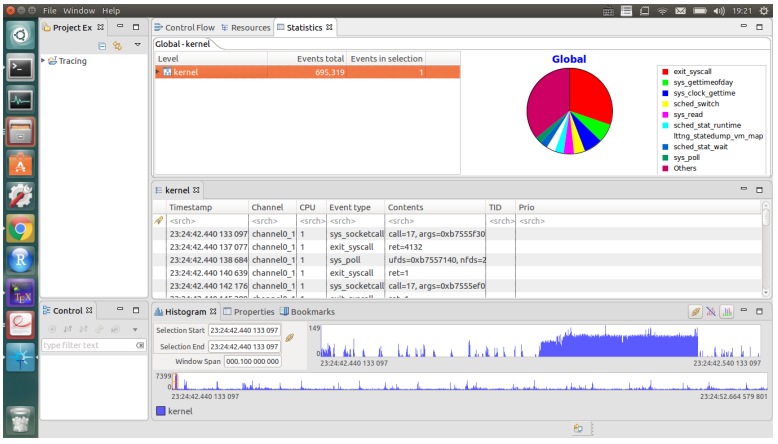


FIGURE 2.23: Screenshot of Trace Compass

2.20 Chapter Conclusion

Findings in literature review shows that computation time certainly can be improved by utilizing parallel computing. I am very confident that the performance tracing of parallel computing on R and Python will show significant improvement compared to serial computing.

Chapter 3

Project Design Proposal

3.1 Phase 1

3.1.1 Introduction

The main goal of Phase 1 is to provide a prototype that parallelize data processing as a proof-of-concept. The requirements for implementation are as below:

1. Parallel programming will be implemented for data processing as a proof-of-concept that parallel processing is capable of speeding up the data processing time.
2. The parallel program will be built on a chosen programming language for the data processing as a prototype.
3. Performance of the program will be gauged based on its execution time.

4. To verify that there is indeed a speed up in data processing time, we will compare the execution time of the same dataset processed in serial and parallel.

3.1.2 Context Diagram

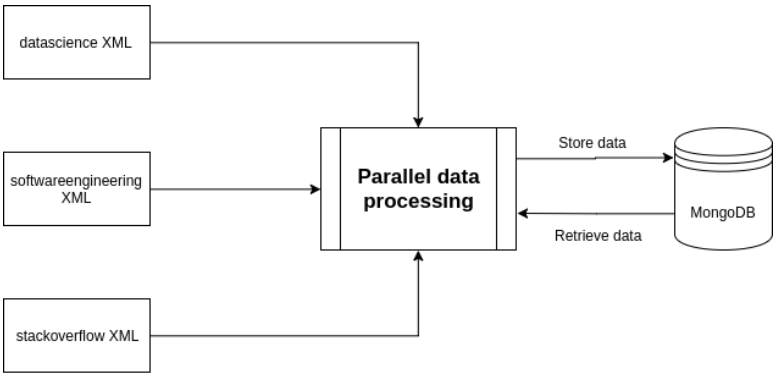


FIGURE 3.1: Context Diagram

The context diagram contains four inputs and one output. The input sources are our three project data - *datascience*, *softwareengineering*, *stackoverflow* and another input is data stored in MongoDB. The data processing is done by our program written in Python.

3.1.3 Flowchart

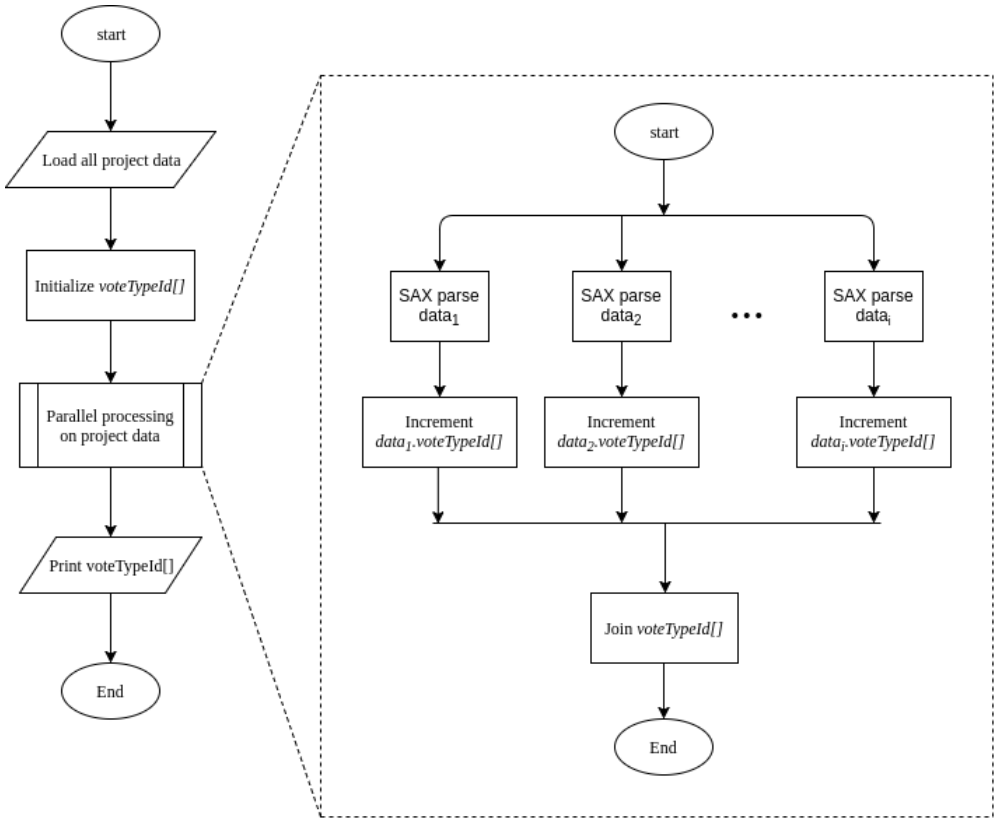


FIGURE 3.2: Parallel Program Flowchart

We will SAX parse project data multiple times in parallel to get its *VoteTypeId*. Refer to Appendix A under votes.xml, there are a total of 13 types of *VoteTypeId* ranging from 1 to 13. We first initialize the voteTypeId to array of thirteen zeroes. We then increment the respective index of the array depending on the value parsed. The total number of each 13 *VoteTypeId* will then be displayed.

3.1.4 Project data

For our project, we are using XML formatted data dumps of Stack Exchange websites. Stack Exchange is a network of Q&A websites on topics in varied fields, each site covering a specific topic[20]. We have chosen data dumps of three sites for our projects which are *datascience*, *softwareengineering* and *stackoverflow*. The data structures of the all three data dumps are the same. Detailed data structures are provided in Appendix A.

3.1.4.1 stackoverflow

Stack Overflow is a Q&A site for professional and enthusiast programmers. The data dumps of this site is the largest out of all three sites. The total file size of the data dumps amounted to 185.1 GB. This data dump is chosen because programming relates to what I have studied in this course.

3.1.4.2 softwareengineering

Software Engineering Stack Exchange is a Q&A site for professionals, academics, and students working within the systems development life cycle. The total file size of the data dumps is 1.3 GB. This data dump is chosen because software engineering relates to what I have studied in this course.

3.1.4.3 datascience

Data Science Stack Exchange is a Q&A site for Data Science professionals, Machine Learning specialists, and those interested in learning more about the

field. The total file size of the data dumps is 70.5 MB. This data dump is chosen because data science is the major I am pursuing now.

3.1.5 Project Implementation Plan

For proof-of-concept purpose, we will use only Python in Phase 1 of this project. The project will be implemented using both Python and R in Phase 2.

3.1.5.1 Data Accounting

Before converting the data into JSON file, we must do data accounting to make sure that after converting the data to JSON, the total number of rows of each attributes tally with the raw XML. We can also get information such as how many rows are without certain attributes.

```
Checking the file......
Total rows in file : 41912
Rows with 1 attribute: 0
Rows with 2 attribute: 0
Rows with 3 attribute: 0
Rows with 4 attribute: 37604
Rows with 5 attribute: 4308
```

FIGURE 3.3: Data accounting

As we can see from Figure 3.1, the total rows of raw XML file is 41912, where number of rows with 4 attributes is 37604 and 5 attributes is 4308. These informations can be used to for data validation later on. The source code for data accounting can be found in Appendix B.1.

3.1.5.2 Data conversion and uploading to MongoDB

Because MongoDB only support JSON/BSON formatted data, so we have to do conversion from XML to JSON using Python. Source code for data conversion can be found in Appendix B.2.

3.1.5.3 Decision on program objective

In order to make a comparison between parallel and serial program, we need to have a common objective. It is pointless to do comparisons if the programs are doing different things. It is like comparing apples with oranges.

In this project, we will be extracting the information on count of each 'VoteTypeId' in 'Votes.xml' of *stackoverflow* data (12.6 GB) multiply by four - 50.4 GB. This is to utilize four cores on my machine to run four SAX parse function in parallel. For detailed description of the data structure, refer to Appendix A.

3.1.5.4 Serial Processing on Project Data

1. Referring to flowchart in Figure 3.2, we first load the project data.
2. Initialize *voteTypeId* to array of thirteen zeroes (Number of types of 'VoteTypeId').
3. SAX parse the three project data in for loop with target element tag 'row'.
4. Extract attribute 'VoteTypeId' of element 'row'

5. Increment the respective index of array if the corresponding parsed 'VoteTypeId' match the index.

6. Increment the respective index of array if the corresponding parsed data match the index.

7. Display the count of each *VoteTypeId*.

For source code, refer to Appendix B.3.

3.1.6 Parallel Processing on Project Data

1. Referring to flowchart in Figure 3.2, we first load the project data.

2. Initialize *voteTypeId* to array of thirteen zeroes (Number of types of 'VoteTypeId').

3. SAX parse the three project data in parallel with target element tag 'row'.

4. Extract attribute 'VoteTypeId' of element 'row'

5. Increment the respective index of array if the corresponding parsed 'VoteTypeId' match the index.

6. Join and return the results.

7. Display the count of each *VoteTypeId*.

For source code, refer to Appendix B.4.

3.2 Phase 2

3.2.1 Introduction

In Phase 2, we have developed a vast understanding of parallel and distributed processing as well as XML parsing. The required implementation are listed below:

1. A program that DOM parse huge XMLs will be implemented as a proof-of-concept that DOM could not be used to parse huge XMLs.
2. A program that SAX parse huge XMLs will be implemented as a proof-of-concept that SAX is the way to parse huge XMLs.
3. SAX parse huge XMLs sequentially and in parallel on local machine's single HDD to verify a single HDD's limitation on parallel data processing.
4. SAX parse huge XMLs in parallel on separate HDD to verify that multiple HDD will indeed increase data processing performance.
5. SAX parse huge XMLs on remote machine to find out the performance of distributed processing.
6. A keyword search engine to demonstrate how SAX can be utilized.
7. Compare SAX parsing performance between Python and R.
8. Performance of the program will be gauged based on its execution time.

We have chosen the two largest and most insightful XML files from stackoverflow data which are PostHistory.XML (95 GB) and Posts.XML (56 GB) for proof of concept purpose.

3.2.2 Flowcharts

3.2.2.1 DOM and SAX Parse PostLinks.XML

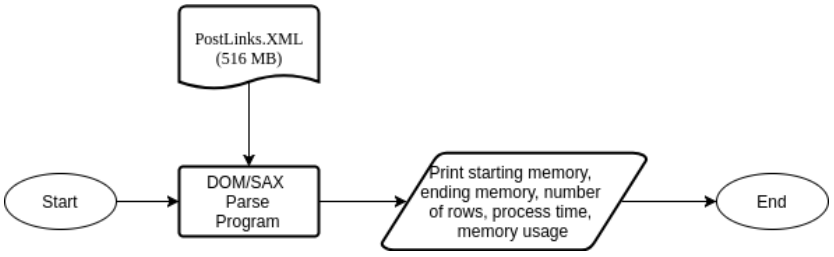


FIGURE 3.4: DOM and SAX Parse PostLinks.XML Flowchart

In this program, we will DOM and SAX parse PostLinks.XML (516 MB) and record laptop's memory before and after parse, number of rows, processing time and its memory consumption in Megabytes.

3.2.2.2 Local SAX Parse PostHistory.XML and Posts.XML Sequentially

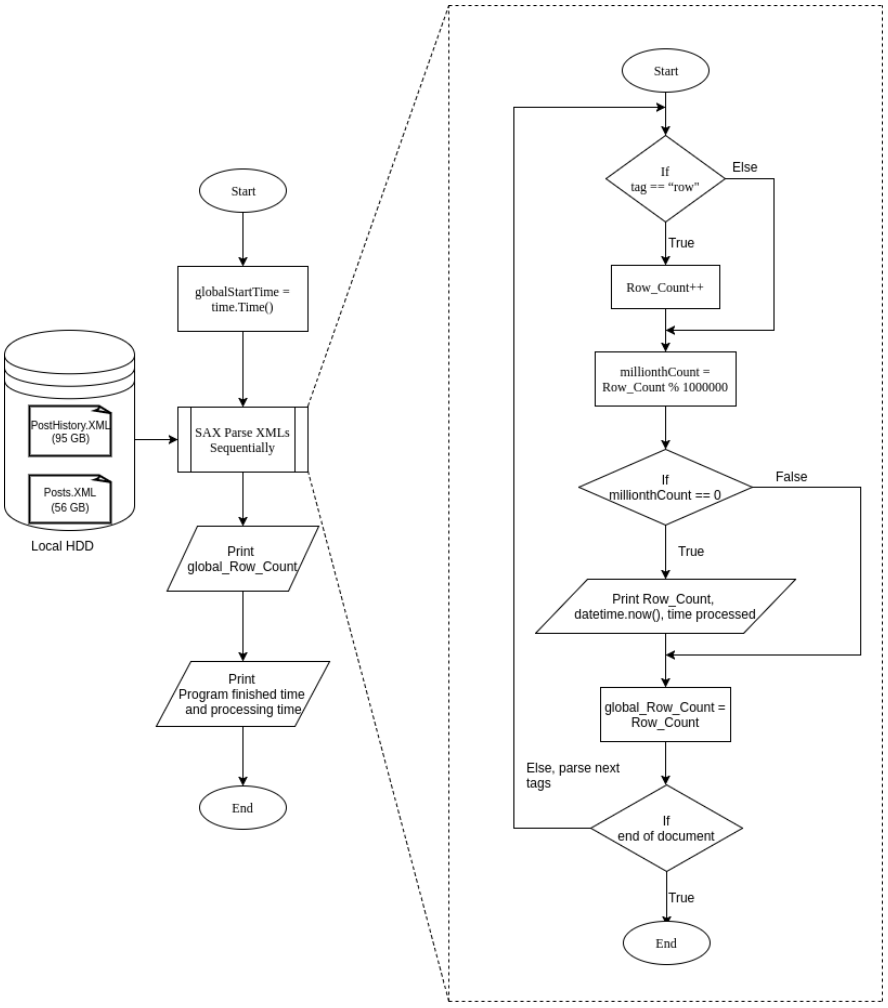


FIGURE 3.5: SAX parse PostHistory.XML and Posts.XML sequentially on local HDD flowchart

In this program, we will SAX parse PostHistory.XML (95 GB) and Posts.XML (56 GB) sequentially (one after another) on a local machine's HDD. The output of this program would be processing time in seconds at the interval of one million rows. The result will be saved in respective CSV files.

3.2.2.3 Local SAX Parse PostHistory.XML and Posts.XML Concurrently

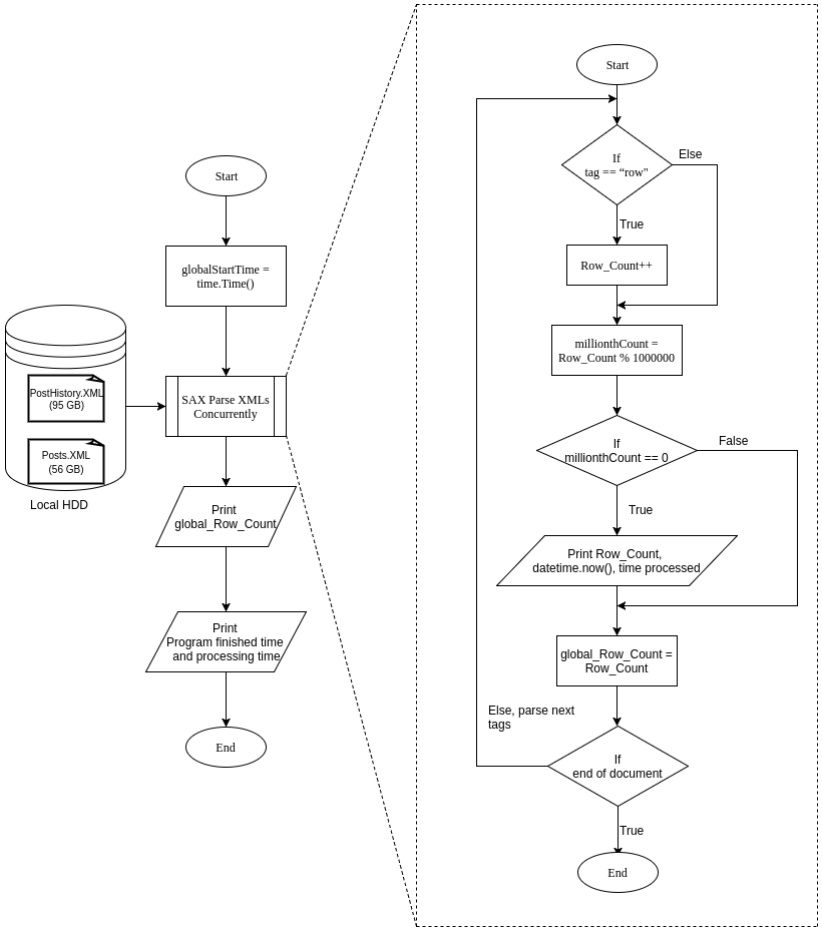


FIGURE 3.6: SAX Parse PostHistory.XML & Posts.XML concurrently on local HDD flowchart

In this program, we will SAX parse both PostHistory.XML and Posts.XML on a local machine's HDD concurrently. The output of this program would be processing time in seconds at the interval of one million rows for respective XMLs. The result will be saved in respective CSV files. Referring to Chapter 2.11, we use the word *concurrent* instead of parallel because it is impossible to read multiple files in parallel on a single HDD.

3.2.2.4 Local SAX Parse PostHistory.XML and Posts.XML in Parallel

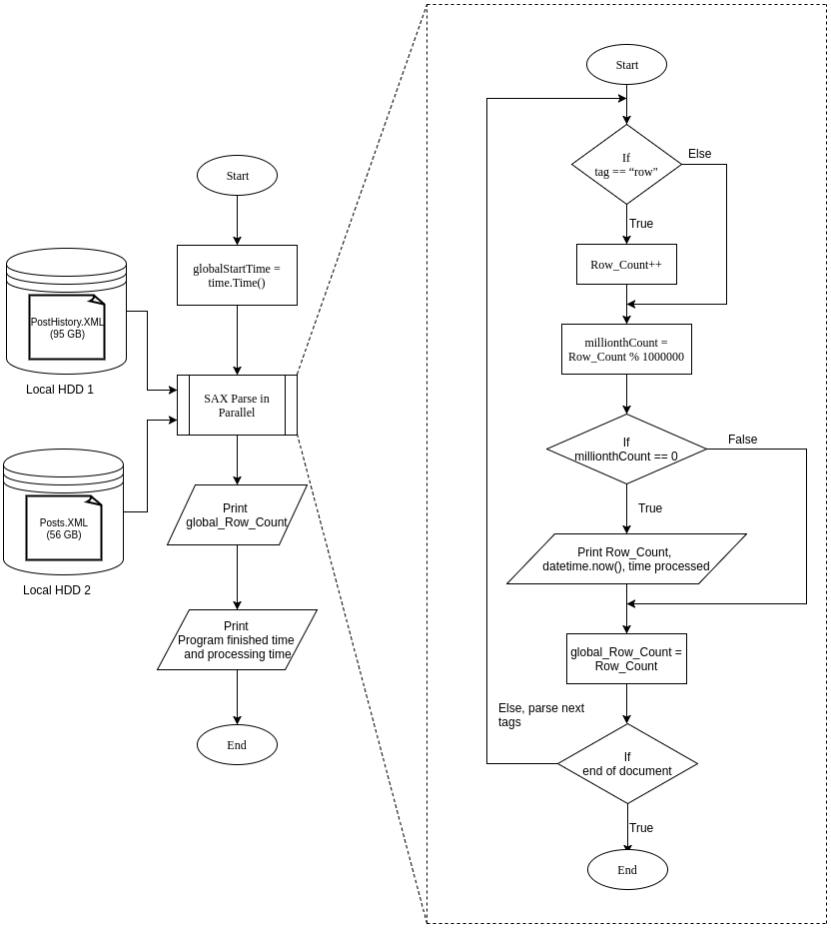


FIGURE 3.7: SAX Parse PostHistory.XML & Posts.XML in parallel on separate local HDD flowchart

In this program, we will SAX parse both PostHistory.XML and Posts.XML on two separate local HDDs in parallel (at the same time). The output of this program would be processing time in seconds at the interval of one million rows for respective XMLs. The result will be saved in respective CSV files. Referring to Chapter 2.11, the objective of this program is to provide an solution for multiple files to be parsed in parallel.

3.2.2.5 Remote SAX Parse PostHistory.XML and Posts.XML Sequentially

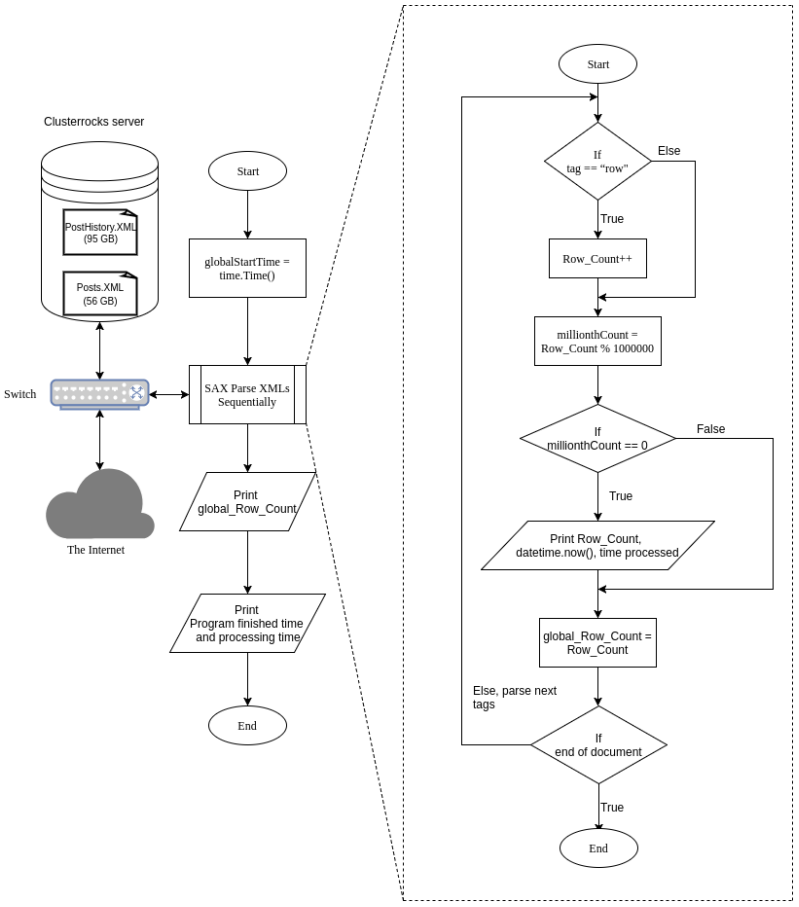


FIGURE 3.8: SAX Parse PostHistory.XML and Posts.XMl sequentially on remote server's HDD flowchart

In this program, we will SAX parse PostHistory.XML and Posts.XML on a remote server's HDD. The output of this program would be processing time in seconds at the interval of one million rows. The result will be saved in respective CSV files.

The purpose of this program is to find out the difference in performance between SAX parsing local XMLs and remote XMLs.

3.2.2.6 Remote SAX Parse PostHistory.XML and Posts.XML Concurrently

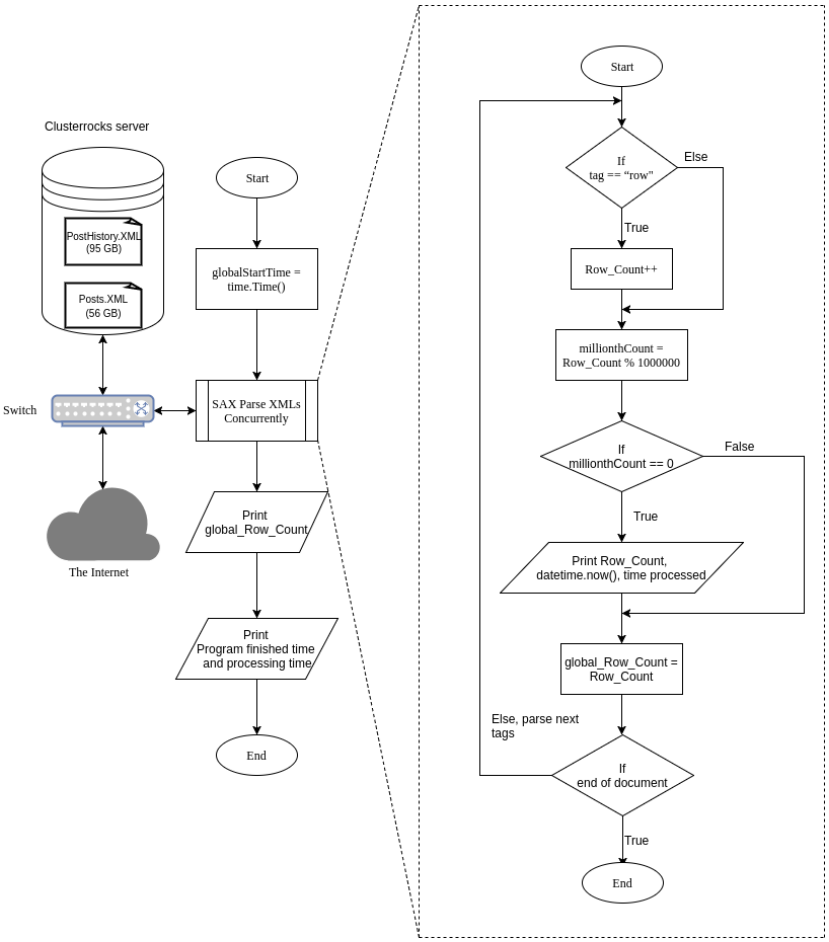


FIGURE 3.9: SAX Parse PostHistory.XML & Posts.XML concurrently on remote server's HDD flowchart

In this program, we will SAX parse both PostHistory.XML and Posts.XML on a remote machine's HDD concurrently. The output of this program would be processing time in seconds at the interval of one million rows for respective XMLs. The result will be saved in respective CSV file. Refer to Chapter 2.11 to understand why the XMLs are parsed concurrently instead of in parallel.

3.2.2.7 Keyword Search Using SAX Parser

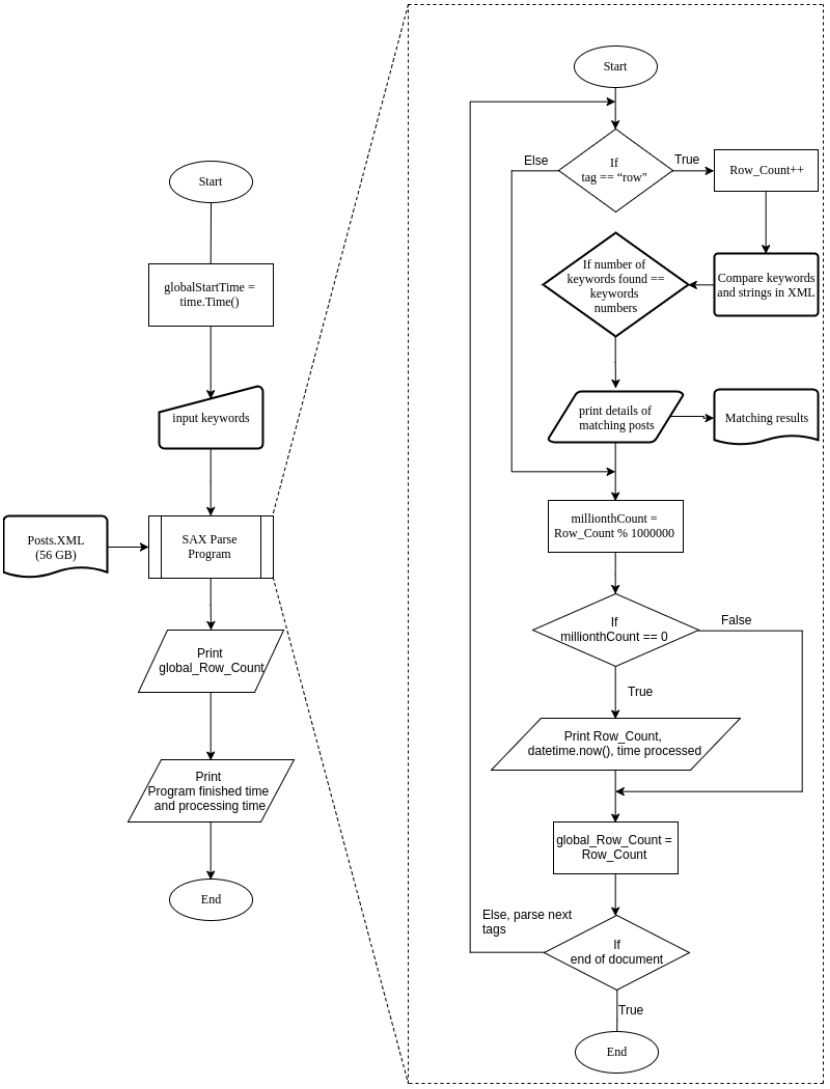


FIGURE 3.10: Keyword Search using SAX flowchart

A parser without an useful feature or function is meaningless. Thus, we had written a search engine that utilizes SAX parser to do keyword search. In this program, we will SAX parse Posts.XML, a document where the details of posts from stackoverflow site are stored. Users are required to key in keywords that they wanted to search, then the program will compare the keywords and the

posts one-by-one in a streamline fashion. When number of matching keywords found in the post is the same as number of keywords user provided, the posts are displayed and saved to a text file at the same time.

Row accounting is also implemented to ensure that the XML is parsed thoroughly until the end of the file. After the parsing is finished, number of results found, number of rows parsed and processing time is displayed and saved to a text file.

3.2.3 Project Implementation Plan

3.2.3.1 DOM and SAX Parse PostLinks.XML

When using DOM to parse a XML file, we would experience around 5 10 times memory bloating when compared to the original XML file. The reason for DOM's memory bloat is because it requires 5 pointers for each node solely for representing the tree structures of XML, there are still many other pointers used for attributes and etc. [21]

When using SAX to parse a XML file, we would not experience memory bloating like we do when using DOM. SAX parser is an event-driven parser, so it read files in a stream and does not keep a record of the XML's elements and attributes in a tree structure like DOM.

So, in order to prove that DOM parse could not parse huge XMLs:

1. Referring to flowchart in Figure 3.4.
2. DOM parse PostLinks.XML (516 MB).

3. Print the starting memory, ending memory, number of rows, process time and memory usage.

4. Count the number of rows in the XML to prove the file is correct.

5. Capture the memory usage using System Monitor and "free" UNIX command.

6. Repeat the steps above using SAX parse.

Refer to Appendix B.5 for DOM parse program source code and Appendix B.6 for SAX parse program source code.

3.2.3.2 Local SAX Parse PostHistory.XML and Posts.XML Sequentially

1. Referring to flowchart in Figure 3.5, we first start a timer to measure process time.

2. SAX parse PostHistory.XML (95 GB) on local hard disk drive.

3. During SAX parse, print the process time and number of row parsed for every million row parsed.

4. When finished SAX parse, print the total process time and total number of row parsed.

5. Save the outputs to a CSV file.

6. After completing steps above, repeat them by SAX parsing Posts.XML (56 GB).

For source code, refer to Appendix B.7 for Python version and Appendix B.8 for R version.

3.2.3.3 Local SAX Parse PostHistory.XML and Posts.XML Concurrently

1. Referring to flowchart in Figure 3.6, we first start a timer to measure process time.
2. SAX parse Posts.XML and Posts.XML simultaneously on same local HDD.
3. During SAX parse, print the process time and number of row parsed for every million row parsed.
4. When finished SAX parse, print the total process time and total number of row parsed.
5. Save the outputs to a CSV file.

For source code, refer to Appendix B.7 for Python version and Appendix B.8 for R version.

3.2.3.4 Local SAX Parse PostHistory.XML and Posts.XML in Parallel

In order to prove when trying to read multiple files in parallel, multiple HDD is required to increase read speed:

1. Referring to flowchart in Figure 3.7.

2. Mount extra HDD on local machine.
3. Place PostHistory.XML and Posts.XML in separate HDD.
4. SAX parse PostHistory.XML and Posts.XML simultaneously on separate HDD.
5. During SAX parse, print the process time and number of row parsed for every million row parsed for both XML.
6. When finished SAX parse, print the total process time and total number of row parsed for XML.
7. Save the outputs of both XMLs to their respective CSV files.

For source code, refer to Appendix B.7 for Python version and Appendix B.8 for R version.

3.2.3.5 Remote SAX Parse PostHistory.XML and Posts.XML Sequentially

1. Create a local directory for mounting remote server's file system.
2. Mount the remote server's file system on local directory created in Step 1.
3. Referring to flowchart in Figure 3.8, we first start a timer to measure process time.
4. SAX parse PostHistory.XML on mounted file system.
5. During SAX parse, print the process time and number of row parsed for every million row parsed.

6. When finished SAX parse, print the total process time and total number of row parsed.

7. Save the outputs to a CSV file.

8. After completing steps above, repeat steps 3-7 by SAX parsing Posts.XML.

For guideline on using SSHFS to mount remote file system, refer to Appendix G.

For source code, refer to Appendix B.7 for Python version and Appendix B.8 for R version.

3.2.3.6 Remote SAX Parse PostHistory.XML and Posts.XML Concurrently

1. Create a local directory for mounting remote server's file system.

2. Mount the remote server's file system on local directory created in Step 1.

3. Referring to flowchart in Figure 3.9, we first start a timer to measure process time.

4. SAX parse PostHistory.XML and Posts.XML simultaneously on mounted file system.

5. During SAX parse, print the process time and number of row parsed for every million row parsed.

6. When finished SAX parse, print the total process time and total number of row parsed.

7. Save the outputs to a CSV file.

8. After completing steps above, repeat steps 3-7 by SAX parsing Posts.XML.

For guideline on using SSHFS to mount remote file system, refer to Appendix G.

For source code, refer to Appendix B.7 for Python version and Appendix B.8 for R version.

3.2.3.7 Keyword Search Using SAX Parser

This program is implemented to showcase one of the useful function of SAX parser:

1. Referring to flowchart in Figure 3.10, we first start a timer to measure process time.

2. Take in keywords from user input.

3. SAX parse Posts.XML.

4. Compare keywords with posts extracted from Posts.XML.

5. If a post contains all the matching keywords, display it.

6. Save the details of the post to text file.

7. When finished SAX parse, print the number of results found, total process time and total number of row parsed.

For source code, refer to Appendix B.9.

Chapter 4

Implementation Methodology

4.1 Software Engineering Methods

The definition of Software Engineering according to IEEE[22] is:

1. *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*
2. *The study for approaches as in the first definition.*

The core process framework[23] for software construes five framework activities:

- **Communication.** Communication between customers and software engineers is critical to understand the customers' requirements before starting a project.
- **Planning.** Planning comprises technical and management practices that assist software engineers to create a road map towards the project goal.

- **Modeling.** Software modeling represents the design, overview and requirements of the project which are important to gain a better understanding of the software to be built.
- **Construction.** Comprises a set of coding and testing activities which lead to end result of deliverable and operating software to customers.
- **Deployment.** Cycle of delivery, support and feedback is performed once the software is deployed to customers.

Several types of SDLC framework models exist. Two of the SDLC framework model methods, prototyping and incremental are used in this project.

4.1.1 Prototyping Model

Software evolves gradually over time. Often times, we only know the general objectives but cannot the detailed requirements are fuzzy. In such cases, prototyping model assists us to understand better what should be built.

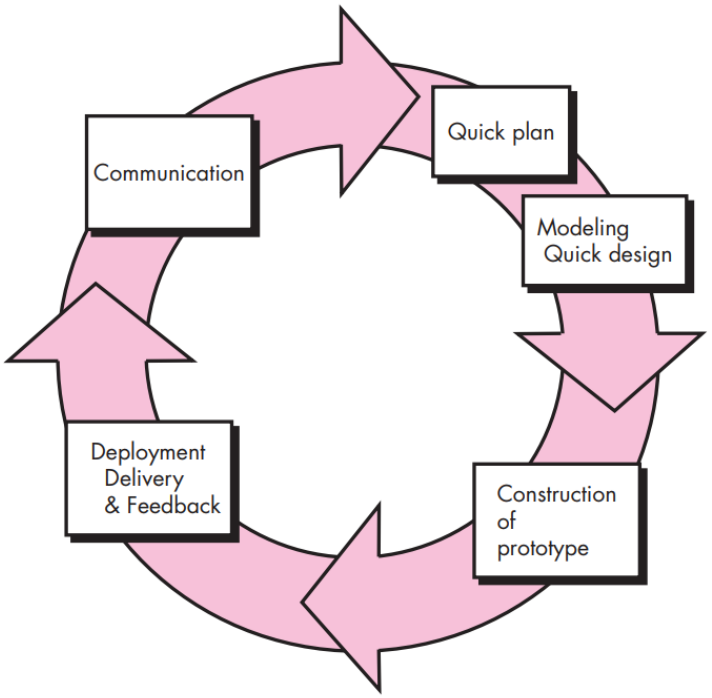


FIGURE 4.1: Prototyping Model

The quick design results in construction of prototype. The prototype is then assessed by end users who then provide feedbacks which further refine requirements. This process is iterated over and over to fine tune the prototype to satisfy end users and stakeholders.

4.1.2 Incremental Model

In software development, there are many cases that the major requirements are clear but the scope is too large to be implemented linearly. In such cases, incremental model assists us to generate working softwares with core function early and build on the details later on by dividing cycles to smaller and manageable module.

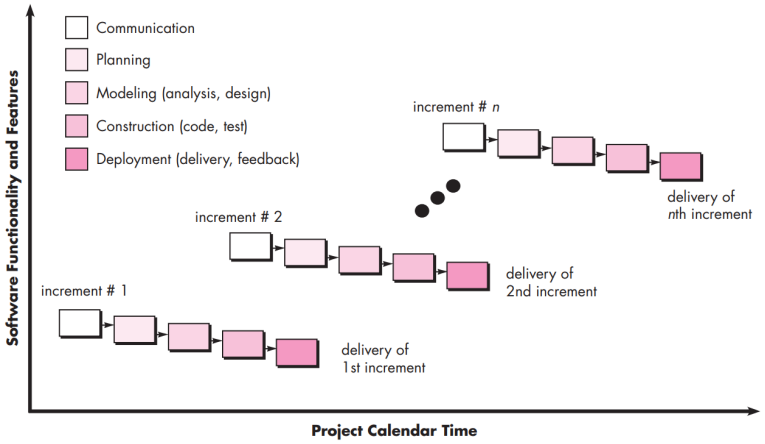


FIGURE 4.2: Incremental Model

The first increment is usually the core product without many supplementary features. The next incrementation will address modification needed for the core product based on the evaluation by end users on the core product.

4.1.3 Methodology for this Project

In this project, we will be developing a parallel program for data processing. To achieve this, features and functions will be added while it is in development. We will be using Incremental Prototyping model in this project, which is a combination of Prototyping Model and Incremental Model introduced in Section 4.1.1 and 4.1.2.

4.2 Project Infrastructures

4.2.1 Hardware Specifications

64-bit Personal Computer. A Personal Computer (PC) running on Linux Ubuntu 16.04 LTS, powered by 64-bit Intel Core^{TM} i7-4500U 1.80GHz Dual-core with hyper-threading processor, 8GB RAM, 1TB HDD and a DVD-Drive. Refer to Appendix F.1 for more detailed information.

Clusterrocks server. A server running on Rocks 6.2 Sidewinder (CentOS 6.6), powered by quad-core Intel Xeon Processor E5440 (12M Cache, 2.83 GHz, 1333 MHz FSB), 6GB RAM, two 250 GB HDD, 1TB HDD. Refer to Appendix F.2 for more detailed information.

1 TB HGST HTS541010A9E680 HDD. Refer to Appendix F.3 for technical specifications and benchmark.

500 GB ATA ST500LT012-1DG14 HDD. Refer to Appendix F.4 for technical specifications and benchmark.

4.2.2 List of Software Resources

1. **Linux Ubuntu 16.04 LTS.** This operating system is chosen because of its flexibility, constantly updated features and huge libraries for developers. Most of all, the OS itself is open source and so are most software applications built on the OS.

2. **R version 3.4.1 - "Single Candle".** R is an open source programming language for statistical computing supported by R Foundation for Statistical Computing. It is used for data processing in this project.

3. **RStudio Version 1.0.153.** RStudio is an IDE for R language. We will be using this IDE throughout the project.

4. **Python version 3.5.2.** Generally, we will use Python 3 for data processing in this project.

5. **Python version 2.7.12.** Python 2 is used in case there is a package we need to use but it is not compatible with Python 3.

6. **PyDev.** An IDE for python.

7. **MongoDB version v3.4.6.** A NoSQL document-oriented database which we will use to store and retrieve our JSON file converted from XML file in this project.

4.2.3 Other Project Resources

4.2.3.1 R - Installed Packages

Below are list of required packages in R for this project:

1. **XML.** This package is used to manipulate and read XML.

2. **jsonlite.** This package is used to work with JSON in R. The main function we will be using in this package is to convert R object to JSON file.

3. **mongolite**. This provides R the capabilities to work with MongoDB, our database for this project.
4. **foreach.** Enable loop to execute in parallel.
5. **parallel.** Support for parallel computation in R.
6. **doParallel.** It acts as an interface between *foreach* and *parallel* package.

All the package listed here can be installed using the command *install.packages('package-name')* in R.

4.2.3.2 Python - Installed Packages

Below are the list of required packages in Python for this project:

1. **xmltodict.** A package that can parse XML input into dictionary.
2. **json.** A package for encoding and decoding JSON.
3. **pymongo.** Python driver for MongoDB.
4. **lxml.** A package for processing XML and HTML.
5. **multiprocessing.** Allow parallel execution of processes on multiple cores.

All the package listed here can be installed using the command *pip install 'package-name'* in terminal.

4.2.4 Infrastructure Setup and Installation

The required resources are listed in Section 4.2.1 and Section 4.2.2.

4.2.4.1 R

1. Add R to Ubuntu Keyring.
2. Add R repository.
3. Install R-base.

The full installation steps for R is found in Appendix C.

4.2.4.2 RStudio

1. Ensure that R is installed.
2. Download RStudio installer from official website.
3. Run the installer.

The full installation steps for RStudio is found in Appendix D.

4.2.4.3 Python

Ubuntu 16.04 comes with both Python 2 and 3 pre-installed. So no need to install Python, only upgrade by using *sudo apt-get install python* and *sudo apt-get install python3* command.

4.2.4.4 PyDev

1. Download installer from Eclipse official website.
2. Install and run Eclipse.

3. Install PyDev from Eclipse Marketplace.

4. Setup Python interpreter.

The full installation steps for PyDev is found in Appendix E.

Chapter 5

Implementation Plan

5.1 Project Task Identification

5.1.1 Identification and Mitigation Measures of Critical Tasks

In order to achieve this project, critical tasks have to be identified before the project begins.

The list of critical tasks in are described below:

1. **Acquire a suitable computer hardware.** Because our project focus is parallel processing on Big Data, our computer's processor must be at least of multi-core and memory sufficient for data processing.

2. **Choosing a suitable operating system.** There are many types of operating systems and each of them are designed differently. So, we have

to choose an operating system that is most optimized for parallel processing.

3. **Acquire a suitable datasets.** We will not be doing domain analysis in this project. So, a dataset that is easily understood and of large size is required. We have chosen the Stack Exchange data dump (XML) for this project.

4. **Selection of programming language for parallel data processing.** To manipulate and read data in parallel, a programming language is needed. We have chosen R and Python, which is top two most trending programming language for data analytics with capabilities to process data in parallel.

5. **Selection of database.** There are many types of databases, but not all is suitable for our data format (XML), especially SQL database. We have chosen MongoDB which is a NoSQL database because XML file can be converted easily to JSON and vice versa.

6. **Setup of database.** Database have to be setup to enable distributed data processing.

7. **Coding of program for serial data processing.** We must create a serial data processing program for comparison between serial and parallel processing. To mitigate and reduce the risk, this task has to be completed as soon as possible.

8. **Coding of prototype program for parallel data processing.** We must create a prototype parallel data processing program as a

proof-of-concept. To mitigate and reduce the risk, this task has to be completed as soon as possible.

9. **Execution time comparison for parallel and serial program.** We must compare the execution time between the protoype parallel program and the serial program as a proof-of-concept. Task 7 and Task 8 have to be completed before this task.

10. **Coding of serial processing programs on raw project data in Python and R.** We must create a serial processing program on raw project data in Python and R to be compared against each other. To mitigate and reduce the risk, this task has to be completed as soon as possible.

11. **Coding of parallel processing program on raw project data in Python and R.** We must create a parallel processing program on raw project data in Python and R to be compared against each other. To mitigate and reduce the risk, this task has to be completed as soon as possible.

12. **Coding of serial processing program on database data in Python and R.** We must create a serial processing program on database data in Python and R to be compared against each other. To mitigate and reduce the risk, this task has to be completed as soon as possible.

13. **Coding of parallel processing program on database data in Python and R.** We must create a parallel processing program on database data in Python and R to be compared against each other. To mitigate and reduce the risk, this task has to be completed as soon as possible.

14. **Trace Compass analyses results on Task 10 to Task 13.**
    Performance analysis must be performed on both program in each task to compare their performance. To mitigate and reduce the risk, this task has to be completed as soon as possible.

15. **Statistical report of project data.** An accurate and insightful statistic report must be generated based on the data we have processed. Various data analytics techniques has to be used to ensure a quality statistical report.

5.1.2 Project Tasks for Phase 1

1. Acquire suitable computer hardware.
2. Choosing suitable operating system.
3. Select programming languages for parallel data processing.
4. Select a suitable database.
5. Setup database.
6. Coding program for serial data processing.
7. Coding a prototype program for parallel data processing.
8. Compare execution time between prototype parallel program and serial program.

5.1.3 Project Tasks for Phase 2

1. Coding serial processing programs on raw project data in Python and R.
2. Coding of parallel processing program on raw project data in Python and R.
3. Coding of serial processing program on database data in Python and R.
4. Coding of parallel processing program on database data in Python and R.
5. Compare performance between both program on all tasks above using Trace Compass.

5.1.4 Gantt Chart for Implementation Schedule

	Task Name	Start	Finish	Duration
	⊟ FYP Phase 1	13/7/2017	19/9/2017	69.0 d.
	⊟ **Project initialization**	13/7/2017	17/7/2017	5.0 d.
	Project Objectives	13/7/2017	16/7/2017	4.0 d.
	Project Scope	13/7/2017	16/7/2017	4.0 d.
	Project Deliverables	16/7/2017	17/7/2017	2.0 d.
	⊟ **Literature Review**	18/7/2017	13/8/2017	27.0 d.
	Concurrent Computing & Parallel Computing	18/7/2017	31/7/2017	14.0 d.
	Big Data	1/8/2017	1/8/2017	1.0 d.
	R & Python	2/8/2017	8/8/2017	7.0 d.
	XML Parser	9/8/2017	10/8/2017	2.0 d.
	Performance Analysis & Monitoring	11/8/2017	13/8/2017	3.0 d.
	⊟ **Project Design & Planning**	14/8/2017	23/8/2017	10.0 d.
	Define Design Models	14/8/2017	17/8/2017	4.0 d.
	Define Project Infrastructure	18/8/2017	21/8/2017	4.0 d.
	Define Milestone Deliverables	22/8/2017	22/8/2017	1.0 d.
	Define Execution Tasks	23/8/2017	23/8/2017	1.0 d.
	⊟ **Project Task Execution**	24/8/2017	13/9/2017	21.0 d.
CR	Acquire suitable computer hardware	24/8/2017	24/8/2017	1.0 d.
CR	Choosing suitable operating system	25/8/2017	27/8/2017	3.0 d.
CR	Acquire suitable dataset	28/8/2017	31/8/2017	4.0 d.
CR	Select programming language	1/9/2017	3/9/2017	3.0 d.
CR	Select database	4/9/2017	4/9/2017	1.0 d.
CR	Setup database	5/9/2017	6/9/2017	2.0 d.
CR	Coding serial data processing program	7/9/2017	13/9/2017	7.0 d.
CR	Coding parallel data processing program prototype	7/9/2017	13/9/2017	7.0 d.
CR	Execution time comparison between serial and parallel prototype program	13/9/2017	13/9/2017	1.0 d.
	Acquire Results and Findings	14/9/2017	14/9/2017	1.0 d.
	Finalize Interim Report	15/9/2017	19/9/2017	5.0 d.
	⊟ FYP Phase 2	20/11/2017	9/2/2018	82.0 d.
	Review Interim Report	20/11/2017	24/11/2017	5.0 d.
	Literature Review	25/11/2017	16/12/2017	22.0 d.
	Project Design & Planning	17/12/2017	19/12/2017	3.0 d.
	⊟ **Project Task Execution**	20/12/2017	2/2/2018	45.0 d.
CR	Coding serial processing programs on raw project data in Python and R	20/12/2017	26/12/2017	7.0 d.
CR	Coding parallel processing program on raw project data in Python and R	27/12/2017	9/1/2018	14.0 d.
CR	Coding of serial processing program on database data in Python and R	10/1/2018	16/1/2018	7.0 d.
CR	Coding of parallel processing program on database data in Python and R	17/1/2018	30/1/2018	14.0 d.
CR	Trace Compass analyses results on Task 33 to Task 36	31/1/2018	2/2/2018	3.0 d.
	Acquire Results and Findings	3/2/2018	4/2/2018	2.0 d.
	Finalize Report	5/2/2018	9/2/2018	5.0 d.

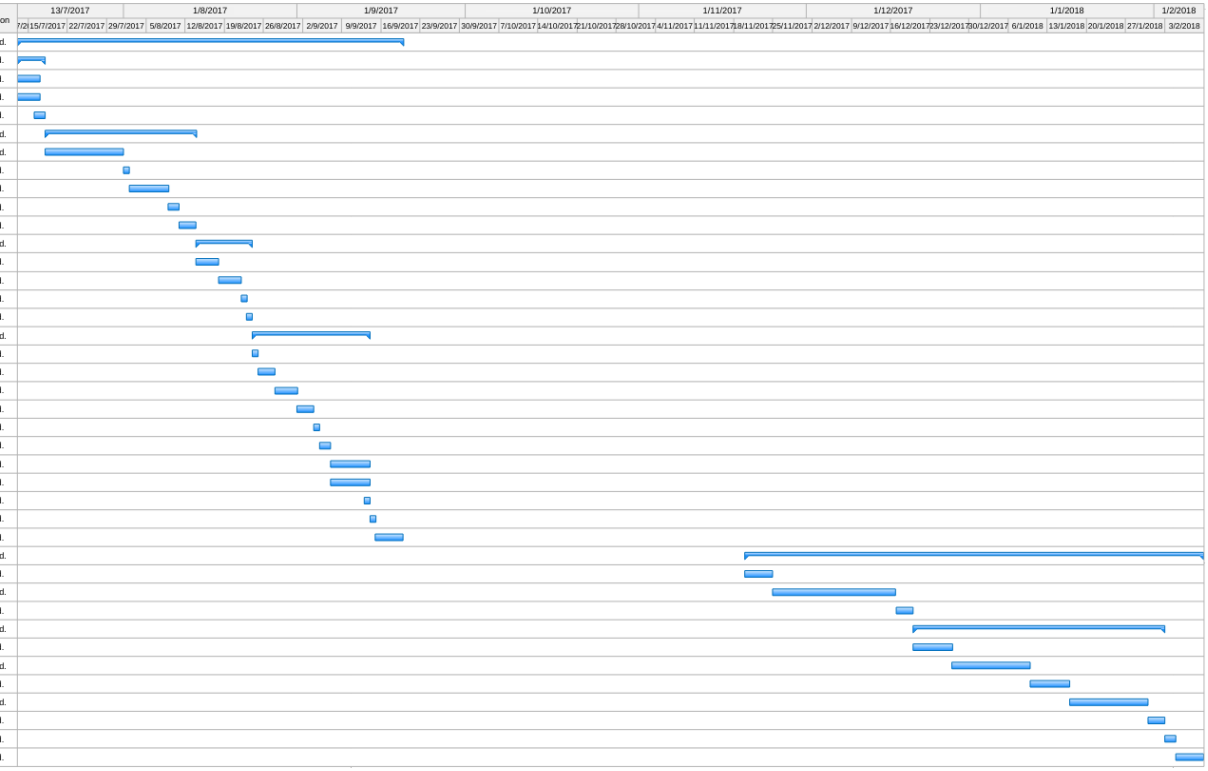


FIGURE 5.1: Gantt Chart Schedule

5.1.5 Milestone Deliverables

The milestone deliverables are:

1. A parallel and distributed data processing program.
2. A statistical report.
3. A report based on this project.

5.2 Planned Execution Activities

1. **Serial processing on project data.** Project data are processed in serial according to requirements and the execution time is recorded.
2. **Parallel processing on project data.** Project data are processed in parallel according to requirements and the execution time is recorded.
3. **Compare execution time between parallel and serial processing.** The execution time of serial and parallel processing is compared. The expected result for this comparison is parallel processing has a faster execution time than serial processing.

Chapter 6

Results and Findings

6.1 Project Data Preparation

We downloaded the XML data for this project from StackExchange data dumps. The data cover the following XML files (*Table 6.1, 6.2 and 6.3*). The detailed data structures (schema) for the above XML files and their relationships (linkages) are provided in Appendix A.

Table 6.1: StackOverflow data dump

StackOverflow			
Filename	**File Size**	**XML Rows**	**Brief Description**
Badges.xml	2.7 GB	22,996,031	Badges name and time earned
Comments.xml	15.3 GB	58,159,095	Details of comments made
Posts.xml	56.2 GB	36,149,133	Details of posts made
PostHistory.xml	95.5 GB	93,449,204	Details of changes made to posts
PostLinks.xml	516.3 MB	4,214,712	Mark if post is duplicated or linked
Users.xml	2.2 GB	7,250,739	Details about users
Votes.xml	12.6 GB	128,370,161	Details about posts vote type
Tags.xml	4.4 MB	49,306	Tags of each post

Table 6.2: SoftwareEngineering data dump

SoftwareEngineering			
Filename	**File Size**	**XML Rows**	**Brief Description**
Badges.xml	47 MB	411,314	Badges name and time earned
Comments.xml	132.6 MB	400,941	Details of comments made
Posts.xml	296.7 MB	186,129	Details of posts made
PostHistory.xml	512.4 MB	472,421	Details of changes made to posts
PostLinks.xml	2.4 MB	20,852	Mark if post is duplicated or linked
Users.xml	89.4 MB	224,058	Details about users
Votes.xml	207.1 MB	2,201,589	Details about posts vote type
Tags.xml	115.5 kB	1,631	Tags of each post

Table 6.3: DataScience data dump

DataScience			
Filename	**File Size**	**XML Rows**	**Brief Description**
Badges.xml	3 MB	26,501	Badges name and time earned
Comments.xml	4.5 MB	14,600	Details of comments made
Posts.xml	19.1 MB	12,920	Details of posts made
PostHistory.xml	29.5 MB	36,715	Details of changes made to posts
PostLinks.xml	48.1 kB	430	Mark if post is duplicated or linked
Users.xml	10.5 MB	25,271	Details about users
Votes.xml	3.8 MB	41,912	Details about posts vote type
Tags.xml	21.3 kB	326	Tags of each post

6.2 Program Codes

The list of Python and R scripts (program codes) for processing the XML files used in this project are provided in the table below. The detailed program listing are provided in Appendix B.

Table 6.4: Program source codes list

Appendix	Program Name	Applicable on XML files	Brief Description
B1	xml_accounting.py	All XML	Get total rows and how many rows have how many categories
B2	xml-to-json-mongodb.py	All XML	Convert XML into JSON and upload to MongoDB
B3	votetypeid-serial.py	Votes.xml	Get the number of rows for each vote type and processing time
B4	votetypeid-parallel.py	Votes.xml	Same as above but using multi-thread
B5	dom-parse-PostLinks.py	PostLinks.xml	DOM parse PostLinks.xml and get its memory consumption and processing time
B6	sax-parse-PostLinks.py	PostLinks.xml	SAX parse PostLinks.xml and get its memory consumption and processing time

B7	python-sax-parse.py	All XML	SAX parse specified XML and get its processing time and number of rows in Python
B8	r-sax-parse.r	All XML	Same as above but implemented in R
B9	keyword-search.py	Posts.XML	Find posts based on keywords inputted and save the results to file for view later and to avoid information bloating on terminal
B10	sequential-concurrent-parallel-sax-plot.py	PostHistory.xml Posts.XML	Plot the execution time against number of rows for sequential, concurrent and parallel SAX parse
B11	local-vs-remote-sax-plot.py	PostHistory.xml Posts.XML	Plot the execution time against number of rows for sequential and concurrent SAX parse for local and remote machine
B12	r-vs-python-sax-plot.py	PostHistory.xml Posts.XML	Plot the execution time against number of rows for sequential, concurrent and parallel SAX parse for Python and R

6.3 Phase 1

6.3.1 Data accounting and Performance Assessment

The session below is our initial run to assess the time taken to just count the rows of the largest XML file in this project, PostHistory.XML.

```
xiang@Xiang:~/Downloads/FYP2/$ python sax-09-PostHistory.py
Parsing PostHistory.xml...

PROGRAM STARTING: 2017-12-28 19:30:28.297431 at 5.10215759277e-05

ROW NUMBER PROCESSED:   0
Now time at: 2017-12-28 19:30:28.914725 at 0.6173421662745

ROW NUMBER PROCESSED:   1000000
Now time at: 2017-12-28 19:30:45.885227 at 17.5878441334

ROW NUMBER PROCESSED:   2000000
Now time at: 2017-12-28 19:31:03.646698 at 35.3493161201

...
...

ROW NUMBER PROCESSED:   92000000
Now time at: 2017-12-28 20:04:31.097275 at 2042.79989314

ROW NUMBER PROCESSED:   93000000
Now time at: 2017-12-28 20:05:03.271491 at 2074.97410917


Check global_Row_Count:  93449204


PROGRAM FINISHED: 2017-12-28 20:05:13.783942 at 2085.4865551
```

LISTING 6.1: Data accounting and performance assessment on PostHistory.xml

6.3.2 Serial Processing on Project Data

This task is successfully achieved. The execution time of serial processing on project data is recorded in Figure 6.1.

```
xiang@Xiang:~/Downloads/FYP/parallel-python$ python votetypeid-serial.py
************************
* Number of VoteTypeId *
************************
AcceptedByOriginator: 30533348
UpMod: 362875816
DownMod: 45280764
Offensive: 18368
Favorite: 34214252
Close: 5017696
Reopen: 22028
BountyStart: 630436
BountyClose: 627584
Deletion: 20154492
Undeletion: 1212920
Spam: 98816
InformModerator: 0
No lines in Votes.xml : 513480644
Execution time: 3002.089257s
```

FIGURE 6.1: Serial Processing Results

6.3.3 Parallel Processing on Project Data

This task is successfully achieved. The execution time of serial processing on project data is recorded in Figure 6.2.

```
xiang@Xiang:~/Downloads/FYP/parallel-python$ python votetypeid-parallel.py
************************
* Number of VoteTypeId *
************************
AcceptedByOriginator: 30533348
UpMod: 362875816
DownMod: 45280764
Offensive: 18368
Favorite: 34214252
Close: 5017696
Reopen: 22028
BountyStart: 630436
BountyClose: 627584
Deletion: 20154492
Undeletion: 1212920
Spam: 98816
InformModerator: 0
No lines in Votes.xml : 128370161
Total lines parsed : 513480644
Execution time: 2076.443763s
```

FIGURE 6.2: Parallel Processing Results

6.3.4 Execution Time Comparison

This task is successfully achieved. Result is shown in Table 6.1. As expected, the execution time of parallel processing is indeed faster than serial processing.

TABLE 6.5: Performance Comparison

Processing Type	Execution Time(s)	Performance Increase(%)
Serial	3002	30.8
Parallel	2076	

6.4 Phase 2

6.4.1 Comparison between DOM and SAX Parse

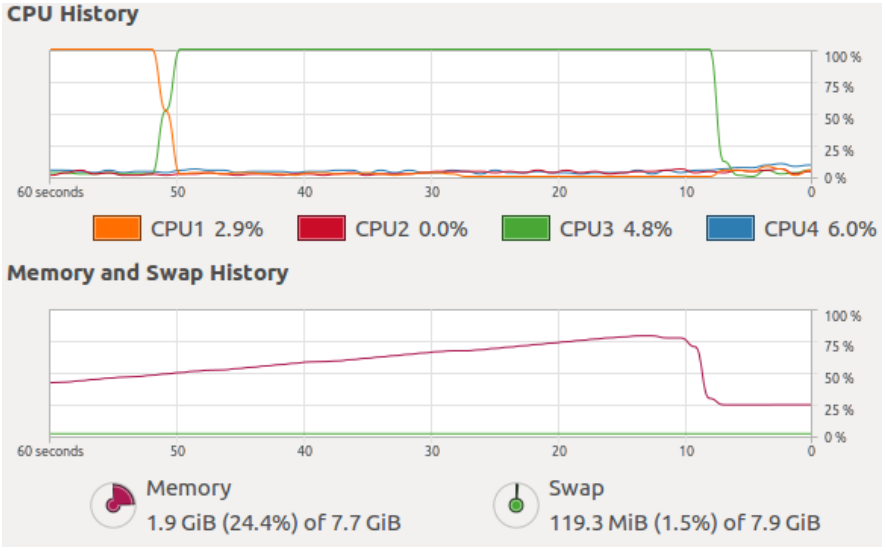


FIGURE 6.3: System Monitor during DOM Parse

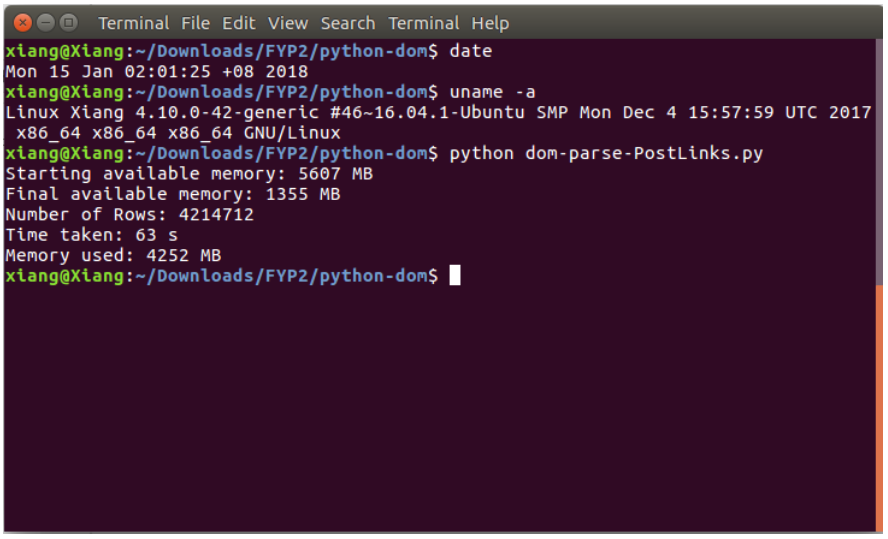


FIGURE 6.4: DOM Parse results

When using DOM to parse a XML file, we would experience around 5 10 times memory bloating when compared to the original XML file. The reason for DOM's memory bloat is because it requires 5 pointers for each node solely for representing the tree structures of XML, there are still many other pointers used for attributes and etc. [21]

As we can see from Figure 6.3 and Figure 6.4, it took 4.2 GB of memory in order to DOM parse PostsLinks.XML (516 MB). As stated above, PostLinks.XML experienced 8.2 times memory bloating when DOM parsed. It took around 76% of available memory (5.6 GB) in order to DOM parse PostLinks.XML which is only half a Gigabyte, so it would be impossible to parse XMLs that are larger than a Gigabyte on this machine.

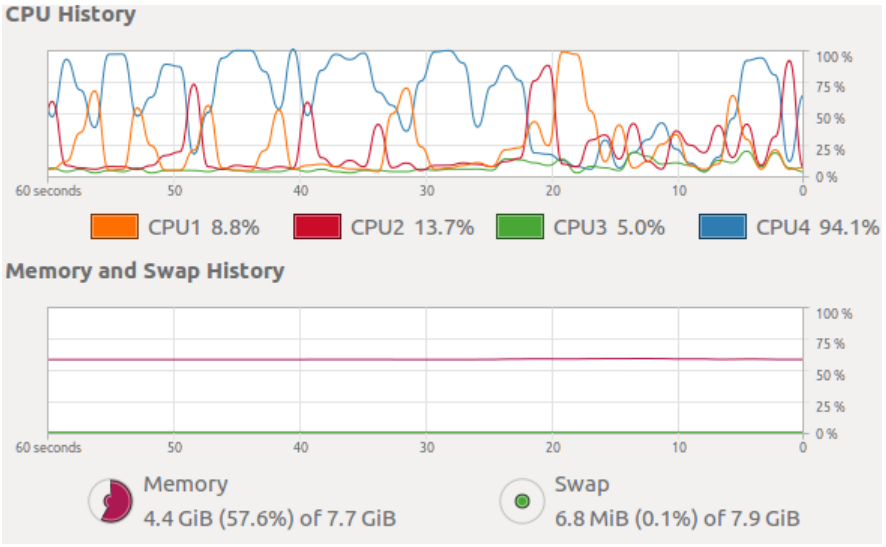


FIGURE 6.5: System Monitor during SAX Parse

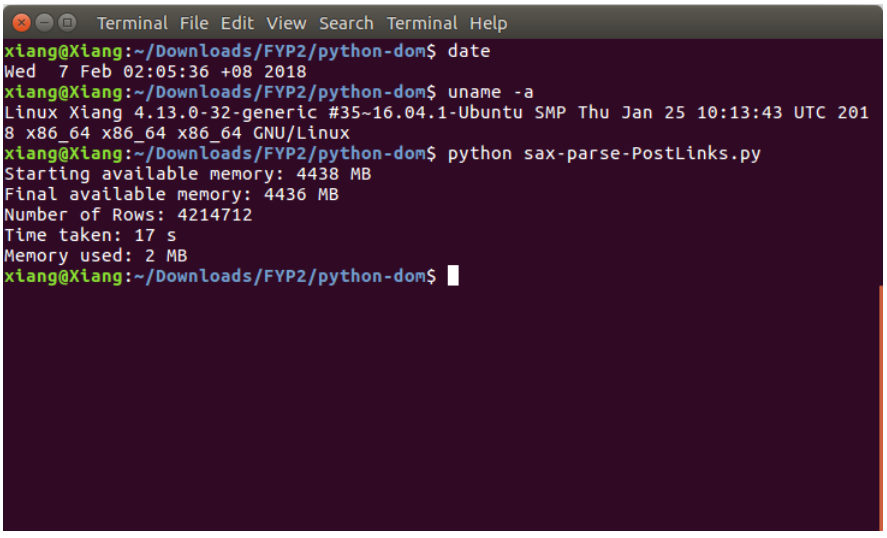


FIGURE 6.6: SAX Parse results

When using SAX to parse a XML file, we would not experience memory bloating like we do when using DOM. SAX parser is an event-driven parser, so it read XMLs in a streamline fashion and discards almost all of that information

once reported. Thus, SAX does not need to keep a record of the XML's elements and attributes in a tree structure like DOM.

From Figure 6.5 and Figure 6.6, we can see from System Monitor that the Memory History is straight line, which mean negligible amount of memory (2 MB) is used. At the same time, the CPUs are busy processing the XMLs which can be seen from CPU History. Moreover, the parsing speed is faster than DOM (46s faster) because SAX does not need to construct tree structure like DOM do.

6.4.2 Keyword Search Engine

We have developed a search engine based on flowchart in Figure 3.10 which utilizes SAX parser to go through our humongous project data and return relevant results based on the keywords provided.

Refer to Appendix H for details on how does the search engine works and its output.

6.4.3 Wrong Method in Parallel Processing Project Data in Phase 1

The result we obtained during Phase 1 of this project shows that utilizing multiprocessing to parallelize SAX parse does indeed speed up the processing speed. But the result obtained is actually biased.

The result is biased because the design of the SAX parse program is wrong in the first place. The program is actually SAX parsing the exact same file (4x stackoverflow's Votes.XML) multiple times. Thus, the result obtained is

obsolete and it does not simulate real-life situation because nobody is going to SAX parse the exact same XML file multiple times in parallel. The design caused the SAX parsing to be sped up when trying to parallel SAX parse in a single HDD as opposed to what we have discussed in Chapter 2.11, because we are essentially parsing only one file on the exact same spot on HDD's track.

6.4.4 Performance Comparison of Parallel, Concurrent and Sequential SAX Parse

In order to avoid cognitive burden when perceiving and comparing the data in graph below, we will only plot the result obtained from program implemented using Python.

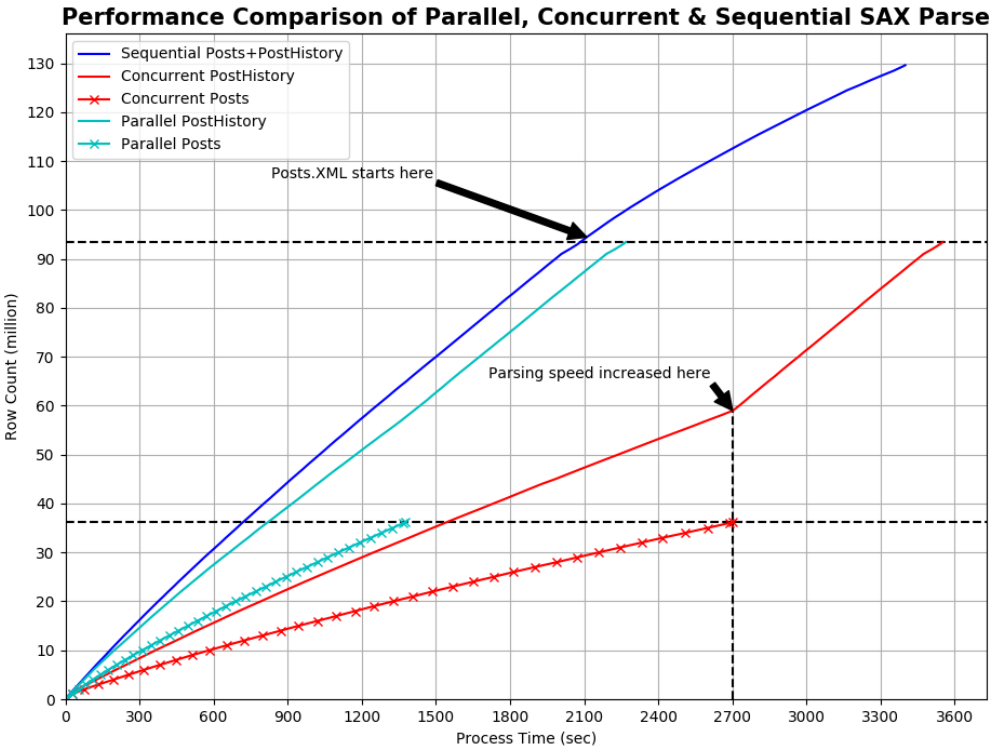


FIGURE 6.7: Performance comparison of parallel, concurrent and sequential SAX parse line graph

From the Figure 6.7, we can see that sequential SAX parse is slightly faster than concurrent SAX parse. This shows that there is no difference in performance between sequential and concurrent SAX parse. An interesting phenomenon is found in the graph – during concurrent SAX parse. When Posts.XML is finished parsing, parsing speed of PostHistory.XML is increased.

We can also see that there is a significant speed up in SAX parsing speed when we are SAX parsing PostHistory.XML and Posts.XML simultaneously (parallel) on separate HDDs. Posts.XML requires only around half the time to finish SAX parsing when compared to the its concurrent counterpart. We also found out that it takes longer to SAX parse Posts.XML because it has more information per row (1555 Bytes per row) compared to PostHistory.XML (1022 Bytes per row).

Refer to Appendix B.10 for Figure 6.7 source code.

6.4.5 Performance Comparison of Local and Remote SAX Parse

Due to resource limitation, we are unable to obtain result on parallel SAX parsing project data on remote server. Thus, we can only compare the performance of sequential and concurrent SAX parse between local machine and remote server.

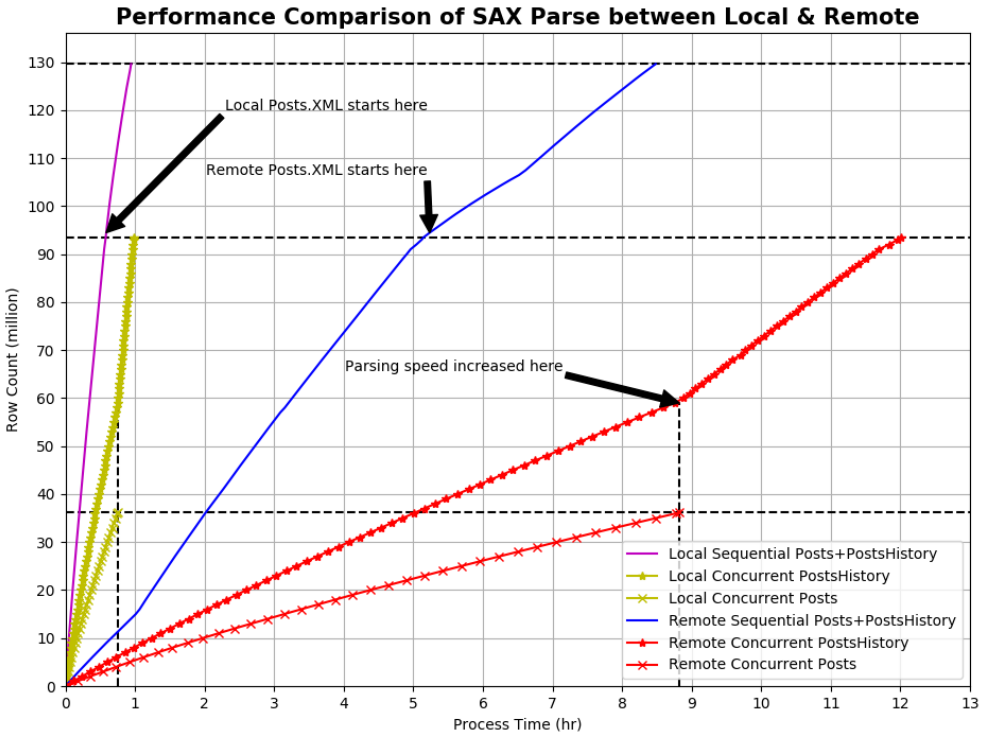


FIGURE 6.8: Performance comparison of local and remote SAX parse line graph

From the Figure 6.8, we can see that both sequential and concurrent SAX parsing on remote server is slower than on local machine by a large margin. There are numerous factors which could cause SAX parsing on remote server to be slower. The most critical factor would be the quality of internet connection.

Screenshot of System Monitor containing Network History during remote SAX parse can be found in Appendix I. We can get the time required to complete SAX parse can be calculated using equation below:

$$TimeRequired = \frac{FileSize}{Bandwidth}$$

So the time required to SAX parse PostHistory.XML (95.5 GB) and Posts.XML (56.2 GB) completely on network bandwidth of 5MB/s (See Appendix I, Figure I.1) will be:

$$TimeRequired = \frac{151.7GB}{5MB/s} \approx 8.43hours$$

Based on calculations, it would take around 8.43 hours in order to finish SAX parse Posts.XML and PostHistory.XML. We can verify that the formula is indeed correct from the line graph, it takes around 8.4 hours to finish SAX parse Posts.XML and PostHistory.XML sequentially.

Refer to Appendix B.11 for Figure 6.8 source code.

6.4.6 Performance Comparison of SAX Parse on R and Python

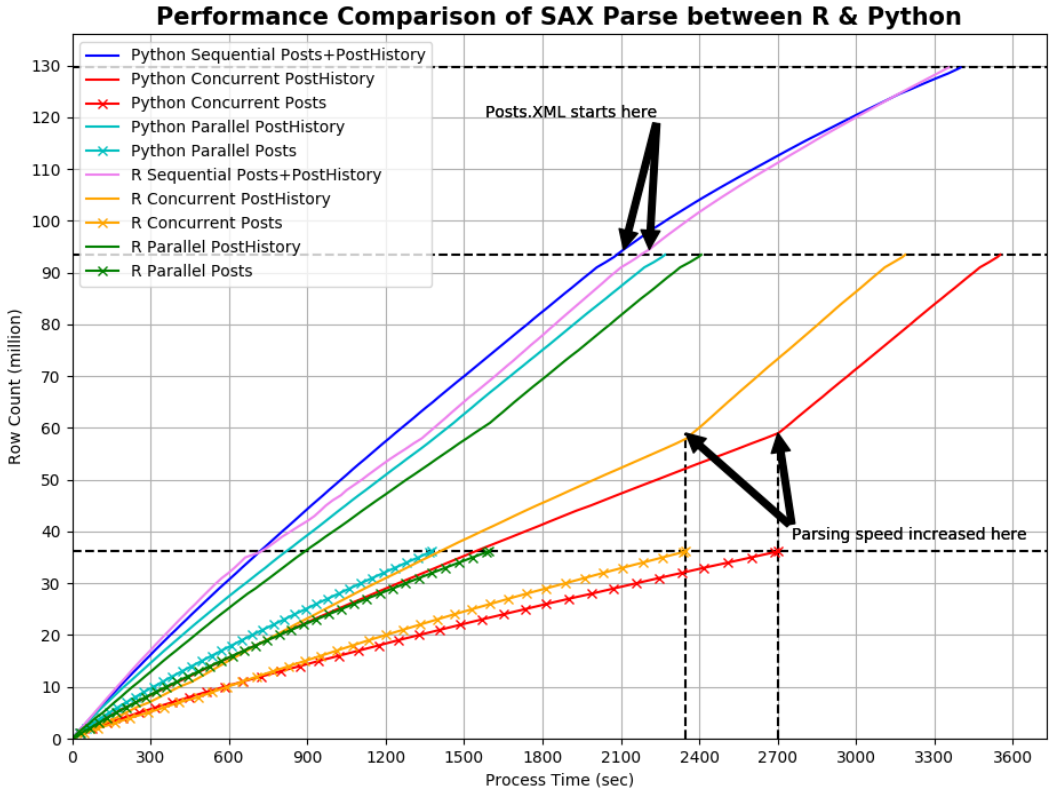


FIGURE 6.9: Performance comparison of R and Python SAX parse line graph

From Figure 6.9, we can see that R is slightly faster than Python in concurrent and parallel SAX parsing. The difference in performance between R and Python on sequential SAX parse is almost negligible.

Refer to Appendix B.12 for Figure 6.9 source code.

6.4.7 Discussion on Results and Findings

1. **DOM vs SAX.** We found that DOM is not suitable for parsing huge XMLs because it just eat up too much memory in order to construct a tree structure for XML parsed. We can conclude that DOM is definitely not suitable for processing humongous XMLs, SAX should be used instead. But if we have astronomical budget to purchase or rent supercomputer with hundreds of TiB memory and desperately need to access the document structure in complex orders, we can consider to adopt DOM parser.

2. **Keyword search engine.** This engine showcased one of the useful function of what we can do with SAX parser. SAX parser can search through the XML and find matching keywords faster than DOM parser, not to mention that we do not even have enough memory to load the XML into memory using DOM parser.

3. **Biased results obtained during Phase 1.** After this finding, we have designed new methodology to obtain an unbiased and correct result in Phase 2, which is to actually SAX parse different XMLs in parallel, instead of parallel parse multiple exact same XML.

4. **Parallel VS Concurrent VS Sequential SAX Parse.** During concurrent SAX parsing, it is found that PostHistory.XML parsing speed increased right after Posts.XML finished parsing. If it is truly parsing in parallel, PostHistory.XML would not get a speed boost after Posts.XML finished parsing because they should not be sharing the same resources. This proved that parallel read on multiple files cannot be achieved because of limitation of single HDD, which we have discussed in Chapter 2.11. It

becomes concurrent instead of parallel SAX parse when trying to SAX parse multiple files on a single HDD. This idea gets proven even further as we can see that we indeed get a significant speed up when we are SAX parsing PostHistory.XML and Posts.XML on separate HDD simultaneously.

5. **Local VS Remote SAX Parse.** There are large difference in processing time for sequential and concurrent SAX parse on remote server, whereas on local machine, the processing time is almost the same for sequential and concurrent SAX parse. This is suspected to be caused by network instability or high network traffic during that period. This result tells us that distributed processing is susceptible to network issues.

6. **R VS Python SAX Parse.** We do not have to worry whether to SAX parse our XMLs on R or Python as their performance is about the same. Rather, we should consider other factors such as project scopes and objectives in deciding which of the two languages to adopt in our project. For example, if the purpose of SAX parser in our project is to extract insight from the XMLs and perform statistical analysis on it, then R is better than Python in this case because R is a statistic-oriented language. But, if our usage of SAX parser is involved in development of an application, then Python is better than R because Python is a more flexible and general purpose programming language.

7. **Slight difference in SAX parsing speed for R VS Python.** From Figure 6.9, we can see a noticeable albeit slight difference in SAX parsing speed between R and Python. This might be due to difference in implementation for SAX parser. Even if the algorithm is same, it might

have different implementation because it is implemented by different person. Thus, it is like comparing apple to orange, an useless and unfair comparison. To make a fair performance comparison, we should compare the same built-in function of R and Python. For example, it is proven that for loop in R is actually nearly 7 times slower than Python.[24] But at the end of the day, we should choose the tool that is best for what we are doing, not the fastest.

Chapter 7

Discussions

7.1 Achievements

1. We successfully mapped remote server's file system to local machine.
2. We successfully did parallel and distributed processing on Big Data using both R and Python.
3. We compared performance of SAX parser in different environments (i.e., parallel, concurrent, sequential, local and remote) implemented on R and Python.
4. We developed a keyword search engine which utilizes SAX parser as its core.

7.2 Problems Encountered & Overcoming Them

7.2.1 New Programming Paradigm

The first problem I encountered during the project development is the unfamiliarity to programming style.

Although I have heard about multi-core processing, I have never tried to or even thought about writing my program in parallel manner. I took up this challenge, because I have never tried it. I crawled the Internet for every resource I could find about parallel programming. There are many times where the resources I tried out is not suitable for what I am trying to achieve. For example, when I am using multiprocessing package in Python, I encountered issue on storing data in an array because it is shared-nothing environment. I found a 'solution' which works but the performance impact is too high, the time it took is worse than sequential processing. Solution is still not found after hours of researching on the Internet.

In order to overcome this problem which I could not find a suitable solution for parallel SAX parse on the Internet, I went to StackOverflow and asked the expert community for help about my situation. The community there is very helpful and provided advices for my case. After following their advice and some trial and error, I finally succeed on my first multiprocessing program.

7.2.2 Unfamiliar Data Format

Although I am majoring in Data Science, I have never analyzed a semi-structured data. But the project is processing a semi-structured data format, XML.

Many of the techniques I learned from Data Science course in the university is not applicable in this project. For example, we are only processing structured data such as CSV (Comma-Separated Value) files in the course. So, when I am trying to do data preprocessing on our project data (XML), I am dumbfounded.

Fortunately, my supervisor Mr. Wan Ruslan Yusoff is very experienced. He shared to me his experience on how he did data preprocessing on semi-structured data like XML files. By following his words, I managed to do data accounting on the humongous project data.

7.2.3 Circumventing MongoDB

Halfway through the project, we found out that MongoDB is not suitable to be used in our project. However, MongoDB is excellent if we started the project from scratch where we begin to populate the data from zero following MongoDB schema. The reason that MongoDB is unsuitable in this project because it is ridiculous to split hundreds of millions of rows and convert them into documents (basic unit of data in MongoDB).

Since we started with XML, we might as well continue to do so, since in order to utilize features of MongoDB, we will need to convert our humongous project data into JSON, which does not make us learn much and requires extremely

great amount of time and effort, not to mention the added complication and possible data corruption during conversion. So, we deemed MongoDB unsuitable for this project.

7.2.4 Trace Compass

I have decided to not use Trace Compass to monitor and analyze the performance of the programs in this project. It provides detailed tracing view (e.g., CPU usage, I/O, kernel memory usage, system call latency, active thread and etc.) and is very useful in tracing applications which involve a lot of complex logics. In data science project, we usually only need to analyze data which does not require complex logics. Trace Compass is a great tool to have but in this project, the features it provided is an overkill.

Also, the learning curve of using Trace Compass is steep, not to mention the learning resources is very scarce and insufficient. It would be very time consuming to learn and put Trace Compass to use in this project, whereas I could gauge the performance of SAX parse program just by using processing time, which is one of the most essential metric in performance analysis.

In the end, I decided to use the most efficient and simple to understand metric - processing time. For example, if the data processing time is too long, it means the performance is bad and vice versa.

7.2.5 Corrupted Project Data on Clusterrocks Server

When we tried to SAX parse PostHistory.XML on clusterrocks server, we encountered error "SAXParseException: not well-formed (invalid token)". We force the SAX parser to continue parsing by ignoring the exception when caught.

When we did data accounting on this problematic XML, we found out that there are only 83 million rows, 13 million rows less than what we have on our local machine's PostHistory.XML. On further investigation, we found out that the corrupted PostHistory.XML is only 83 GB compared to 95 GB uncorrupted PostHistory.XML.

To solve this problem, we cloned the uncorrupted PostHistory.XML from local machine to clusterrocks server.

Chapter 8

Conclusions

8.1 Conclusions

In Phase 1, we have reviewed a lot of details on the implementation of parallel processing as well as experience on how parallel processing improves the data processing speed.

The project objectives are:

1. To learn and understand about the R and the Python programming language concepts and their parallel processing features.
2. To explore different techniques on data processing, parallel and distributed programming for big data.
3. To conduct performance comparisons between R and Python language parallel processing implementations for big data.

4. To implement the handling and conversion of big data into the MongoDB, a document-oriented, JSON-style distributed database.

5. To process combinations of data on multiple repositories in a network that represents a real time, parallel and distributed processing environment.

What we have achieved in Phase 1:

1. We have picked up R and Python sufficiently and established the fundamentals.

2. We have reviewed various types of parallel processing packages and modules in R and Python.

3. We have reviewed methods on processing XML data and successfully implemented it on test dataset.

4. We successfully implemented parallel processing on the test dataset.

5. We successfully converted raw XML data into JSON and uploaded into MongoDB collection.

6. We obtained results which proved that parallel processing has higher performance compared to serial processing.

To be achieved in Phase 2:

1. To implement parallel and distributed processing on real datasets in R and Python.

2. To implement parallel and distributed processing on raw XML and MongoDB data.

3. To implement performance analysis and monitoring using Trace Compass.

4. To compare performance of parallel and distributed processing on real datasets between R and Python.

What we have achieved in Phase 2:

1. We have implemented DOM parse program and compared against SAX parse program to prove that DOM is not suitable for huge XML processing.

2. We have implemented parallel and distributed processing on real datasets in R and Python.

3. We have compared performance of parallel and distributed processing on real datasets between R and Python.

4. We did not incorporate MongoDB in this project due to the project data's format and size, as discussed in Chapter 7.2.3.

5. We did not use Trace Compass to analyze and monitor performance. However, another simple and efficient metric (processing time) is used to monitor performance in its place.

8.2 Lessons Learned

1. **Consistent update with supervisor.** Supervisor can know that if the direction of our project is in the right direction or not. Supervisor is also very experienced, thus they can help us solving our problems and provide useful advices which can enhance our knowledge.

2. **Parallel programming.** Parallel programming is a new skill set I picked up in this project. I would never thought of trying out this programming style if not for this project. I also witnessed the advantage of utilizing parallel programming. The future of parallel programming is already here, we cannot avoid this anymore. Students would have an edge on their future jobs over their colleagues who do not have any experience with parallel programming.

3. **MongoDB.** This is the first time I ever experienced a document-oriented NoSQL database. When dealing with MongoDB, I have realized how different it is compared to conventional SQL databases. The concept of storing unstructured data and its schema totally different from SQL databases. I am very grateful to have the opportunity to actually use a document-oriented NoSQL database.

4. **Data Science is diverse.** Through this, I realized how naive I am and how diverse is Data Science. There is no exact recipe for each case, what techniques and tools should be used is very situational. In order to become a good data scientist, I need to have a lot of different skill sets and be resourceful. For example, I have never thought I could process such a humongous data, it would not even fit on my machine's memory!

5. **Start small but right.** This is one of the most important lesson I have learned from my supervisor. Before, I would try to tackle the big picture at once, which caused me to be counterproductive as my brain would be burned out due to the complexity caused by doing everything at once. After trying out "start small but right", it became more fun for me in solving problems as there are many "Aha!" and "So it goes like this" moments. It brings me joy when I solved the small problems and this in turn motivated me to slowly tackle bigger problems.

8.3 Recommendations for Future Work

1. **Create graphical user interface.** A good engine without a good GUI is wasted. A great UI design can make a program intuitive and thus, easy to learn and use, which results in greater user acceptance.

2. **Linking XMLs.** In this project, we only did SAX parse on each individual project data. However, some of the information in the project data can be linked to the other project data. If we could extract the informations and link them with each other, we might be able to discover some interesting insights.

3. **Borrow computing power through distributed computing.** In this project, we did not manage to borrow computing power from remote machine due to time constraint. Instead, the remote machine is only used as scalable storage. It would be interesting to find out how high the throughput will be when computing power of remote machines are utilized.

4. **Reduce workload before attempting a project.** I recommend future students who read my work and will be taking FYP or any project for that matter, to manage their workload before attempting a project. It will be hard to dedicate yourselves in developing a project if there are too much unrelated works distracting you.

Bibliography

[1] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989. ISBN 0-8053-0177-1.

[2] David Gewirtz. Volume, velocity, and variety: Understanding the three v's of big data. April 2016. URL `http://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/`. Retrieved on 01/08/2017.

[3] Katie McKissick. Just how does facebook store billions of photos? November 2015. URL `https://news.usc.edu/88075/how-does-facebook-store-billions-of-photos/`. Retrieved on 01/08/2017.

[4] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.

[5] Rstudio, 2016. URL `https://www.rstudio.com/products/rstudio/`. Retrieved on 03/08/2017.

[6] Pierre Carbonnelle. PYPL PopularitY of Programming Language, 2016. URL `http://pypl.github.io/PYPL.html`. Retrieved on 04/08/2017.

[7] Brainwy Software Ltda. PyDev, September 2017. URL `http://pydev.org/`. Retrieved on 05/08/2017.

[8] David Smith. The language of statistics. October 2010. URL `https://www.r-bloggers.com/the-language-of-statistics/`. Retrieved on 06/08/2017.

[9] F. Berman, G.C. Fox, and A.J.G. Hey. *GRID COMPUTING: MAKING THE GLOBAL INFRASTRUCTURE A REALITY.* Wiley India Pvt. Limited, 2010. ISBN 9788126527229.

[10] C. Musselle and R.S. Kate. Integrating python and r into a data analysis pipeline. 2015. URL `https://www.mango-solutions.com/blog/integrating-python-and-r-into-a-data-analysis-pipeline-part-1`. Retrieved on 07/08/2017.

[11] Bill Lattner. Python and r together at last: Writing cross language tools scipy 2016, July 2016. URL `https://www.youtube.com/watch?v=VgG4X-H-wdw`. Retrieved on 08/08/2017.

[12] FlashX. The User Guide of FlashR, November 2016. URL `https://flashxio.github.io/FlashX-doc/FlashR-user-guide.html`. Retrieved on 08/08/2017.

[13] WEBOPEDIA. SAX. URL `http://www.webopedia.com/TERM/S/SAX.html`. Retrieved on 09/08/2017.

[14] W3C DOM Interest Group. Document Object Model(DOM). URL `https://www.w3.org/DOM/`. Retrieved on 10/08/2017.

[15] MongoDB, Inc. What is MongoDB?, 2017. URL `https://www.mongodb.com/what-is-mongodb`. Retrieved on 10/8/2017.

[16] Thilbault D.U. Proulx, P. The LTTng Documentation. August 2017. URL `http://lttng.org/docs/v2.10/`. Retrieved on 11/08/2017.

[17] the GDB developers. GDB: The GNU Project Debugger, 2017. URL `https://www.gnu.org/software/gdb/`. Retrieved on 12/08/2017.

[18] The Linux Foundation. Common Trace Format v1.8.2, 2014. URL `http://diamon.org/ctf/`.

[19] Trace Compass. URL `http://tracecompass.org/`. Retrieved on 13/08/2017.

[20] Stack Exchange Inc, 2017. URL `https://stackexchange.com/about`. Retrieved on 14/8/2017.

[21] O.N. Delpratt. *Space Efficient In-memory Representation of XML Documents*. University of Leicester, 2009. Retrieved on 25/1/2018.

[22] IEEE. *IEEE Standard Glossary of Software Engineering Technology*. URL `http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf`. Retrieved on 13/9/2017.

[23] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill, 2010. Retrieved on 13/9/2017.

[24] Is Python faster than R?-Quora. URL `https://www.quora.com/Is-Python-faster-than-R`. Retrieved on 6/2/2018.

Appendices

Appendix A

Data Structure of Project Data


```
- Format: 7zipped
- Files:
  - **badges**.xml
                - UserId, e.g.: "420"
                - Name, e.g.: "Teacher"
                - Date, e.g.: "2008-09-15T08:55:03.923"
  - **comments**.xml
                - Id
                - PostId
                - Score
                - Text, e.g.: "@Stu Thompson: Seems possible to me - why not try it?"
                - CreationDate, e.g.:"2008-09-06T08:07:10.730"
                - UserId
  - **posts**.xml
                - Id
                - PostTypeId
                        - 1: Question
                        - 2: Answer
                - ParentID (only present if PostTypeId is 2)
                - AcceptedAnswerId (only present if PostTypeId is 1)
                - CreationDate
                - Score
                - ViewCount
                - Body
                - OwnerUserId
                - LastEditorUserId
                - LastEditorDisplayName="Jeff Atwood"
```

```
                - LastEditDate="2009-03-05T22:28:34.823"
                - LastActivityDate="2009-03-11T12:51:01.480"
                - CommunityOwnedDate="2009-03-11T12:51:01.480"
                - ClosedDate="2009-03-11T12:51:01.480"
                - Title=
                - Tags=
                - AnswerCount
                - CommentCount
                - FavoriteCount
- **posthistory**.xml
        - Id
        - PostHistoryTypeId
                - 1: Initial Title - The first title a question is asked with.
                - 2: Initial Body - The first raw body text a post is submitted with.
                - 3: Initial Tags - The first tags a question is asked with.
                - 4: Edit Title - A question's title has been changed.
                - 5: Edit Body - A post's body has been changed, the raw text is stored
                                 here as markdown.
                - 6: Edit Tags - A question's tags have been changed.
                - 7: Rollback Title - A question's title has reverted to a previous
                                      version.
                - 8: Rollback Body - A post's body has reverted to a previous version -
                                     the raw text is stored here.
                - 9: Rollback Tags - A question's tags have reverted to a previous version
                - 10: Post Closed - A post was voted to be closed.
                - 11: Post Reopened - A post was voted to be reopened.
                - 12: Post Deleted - A post was voted to be removed.
                - 13: Post Undeleted - A post was voted to be restored.
                - 14: Post Locked - A post was locked by a moderator.
                - 15: Post Unlocked - A post was unlocked by a moderator.
                - 16: Community Owned - A post has become community owned.
                - 17: Post Migrated - A post was migrated.
                - 18: Question Merged - A question has had another, deleted question
                                        merged into itself.
                - 19: Question Protected - A question was protected by a moderator
                - 20: Question Unprotected - A question was unprotected by a moderator
                - 21: Post Disassociated - An admin removes the OwnerUserId from a post.
                - 22: Question Unmerged - A previously merged question has had its answers
                                                                                and votes restored.
        - PostId
        - RevisionGUID: At times more than one type of history record can be recorded by a
                                                                      single action.  All of these will be grouped using
    the same                          RevisionGUID
        - CreationDate: "2009-03-05T22:28:34.823"
        - UserId
        - UserDisplayName: populated if a user has been removed and no longer referenced
                                                                        by user Id
        - Comment: This field will contain the comment made by the user who edited a post
        - Text: A raw version of the new value for a given revision
                - If PostHistoryTypeId = 10, 11, 12, 13, 14, or 15  this column will
                                                                    contain a JSON encoded string with all users who have voted
    for the                         PostHistoryTypeId
                - If PostHistoryTypeId = 17 this column will contain migration details of
                                                                       either "from <url>" or "to <url>"
                - CloseReasonId
                - 1: Exact Duplicate - This question covers exactly the same ground as
                                                                                earlier questions on this topic; its
    answers may be                                  merged with another identical question.
                - 2: off-topic
                - 3: subjective
                - 4: not a real question
                - 7: too localized
- **postlinks**.xml
        - Id
```

```
        - CreationDate
        - PostId
        - RelatedPostId
        - PostLinkTypeId
                - 1: Linked
                - 3: Duplicate
- **users**.xml
        - Id
        - Reputation
        - CreationDate
        - DisplayName
        - EmailHash
        - LastAccessDate
        - WebsiteUrl
        - Location
        - Age
        - AboutMe
        - Views
        - UpVotes
        - DownVotes
- **votes**.xml
        - Id
        - PostId
        - VoteTypeId
                - ' 1': AcceptedByOriginator
                - ' 2': UpMod
                - ' 3': DownMod
                - ' 4': Offensive
                - ' 5': Favorite - if VoteTypeId = 5 UserId will be populated
                - ' 6': Close
                - ' 7': Reopen
                - ' 8': BountyStart
                - ' 9': BountyClose
                - '10': Deletion
                - '11': Undeletion
                - '12': Spam
                - '13': InformModerator
        - CreationDate
        - UserId (only for VoteTypeId 5)
        - BountyAmount (only for VoteTypeId 9)
```

Listing A.1: Data structure of Project Data

Appendix B

Source Codes

B.1 xml_accounting.py

```
#File: xml_accounting.py
#Date: Fri Aug 11 14:07:24
#Envn: Linux Xiang 4.10.0-33-generic #37~16.04.1-Ubuntu SMP  UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

from lxml import etree

if __name__ == "__main__":

        filename = "/home/xiang/Downloads/FYP/datasets/datascience/Votes.xml"

        #initialize attribute counter
        one_cat = two_cat = three_cat = four_cat = five_cat = count  = 0

        context = etree.iterparse(filename)
        #Check how many rows have how many attributes
        for event, element in context:
                if element.tag == 'row':
                        if 'Id' in element.attrib:
                                count += 1
                        if 'PostId' in element.attrib:
                                count += 1
                        if 'VoteTypeId' in element.attrib:
```

```
                                count += 1
                        if 'UserId' in element.attrib:
                                count += 1
                        if 'CreationDate' in element.attrib:
                                count += 1

                        if count == 1:
                                one_cat += 1
                        elif count == 2:
                                two_cat += 1
                        elif count == 3:
                                three_cat += 1
                        elif count == 4:
                                four_cat += 1
                        elif count == 5:
                                five_cat += 1

                        count = 0
                        element.clear()

        #get total rows in file
        num_lines = sum(1 for line in open ("/home/xiang/Downloads/FYP/datasets/datascience/Votes.xml")) - 3

        print('Checking the file.....')
        print('Total rows in file : %d' % num_lines)
        print('Rows with 1 attribute: %d' % one_cat)
        print('Rows with 2 attribute: %d' % two_cat)
        print('Rows with 3 attribute: %d' % three_cat)
        print('Rows with 4 attribute: %d' % four_cat)
        print('Rows with 5 attribute: %d' % five_cat)
```

Listing B.1: Source code for XML data accounting

B.2 xml-to-json-mongodb.py

```
#File: xml-to-json-mongodb.py
#Date: Fri Aug 11 14:07:24
#Envn: Linux Xiang 4.10.0-33-generic #37~16.04.1-Ubuntu SMP  UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

import json
import xmltodict
import sys
import argparse
from pymongo import MongoClient

#Set argument for file to be converted and upload to MongoDB
parser = argparse.ArgumentParser(description='Convert an XML file to JSON.')
parser.add_argument('infile', nargs='?', type=argparse.FileType('rb'),
                    default=sys.stdin)
```

```
15
16 #parsing XML file into dictionary and then convert into JSON
17 def xmltojson(xmlstr):
18         xmlparse = xmltodict.parse(xmlstr,attr_prefix='')
19         jsonstr = json.dumps(xmlparse,indent=1)
20         jsonstr = json.loads(jsonstr)
21         print(jsonstr)
22         return(jsonstr)
23
24 if __name__ == "__main__":
25         args = parser.parse_args()
26
27         #Connect to MongoDB
28         client = MongoClient('localhost', 27017)
29
30         #Switch db to 'fyp'
31         db = client.fyp
32
33         #Use collection 'tester'
34         tester = db.tester
35
36         #pass in XML to convert into JSON
37         json_data = xmltojson(args.infile)
38
39         #insert converted JSON file into collection
40         result = tester.insert(json_data)
41
42         #close connection
43         client.close()
```

Listing B.2: Source code for XML-JSON conversion and upload to MongoDB

B.3 votetypeid-serial.py

```
1  #File: votetypeid_serial.py
2  #Date: Fri Aug 11 14:07:24
3  #Envn: Linux Xiang 4.10.0-33-generic #37~16.04.1-Ubuntu SMP  UTC 2017
4
5  from lxml import etree
6  import time
7  def sax_parse(filename):
8          context = etree.iterparse(filename, tag = 'row')
9          for event, element in context:
10                 row_vote_type_id = element.attrib.get('VoteTypeId')
11
12                 #increment the counter on the matched VoteTypeId
13                 for i in range(len(voteTypeId)):
14                         if row_vote_type_id == str(i+1):
15                                 voteTypeId[i] += 1
```

```
16                  element.clear()
17                  element.getparent().remove(element)
18          return voteTypeId
19
20  if __name__ == "__main__":
21          #calculate execution time
22          start = time.time()
23
24          file_list = ["/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml",
25                       "/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml",
26                       "/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml",
27                       "/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml"]
28
29          voteTypeId = [0,0,0,0,0,0,0,0,0,0,0,0,0]
30
31          for i in range(len(file_list)):
32                  sax_parse(file_list[i])
33
34          #calculate execution time
35          end = time.time()
36          exec_time = end - start
37
38          num_lines = sum(1 for line in open (file_list[0])) - 3
39          print('************************\n'+
40                '* Number of VoteTypeId *\n' +
41                '************************')
42          print('AcceptedByOriginator: %d' % voteTypeId[0])
43          print('UpMod: %d' % voteTypeId[1])
44          print('DownMod: %d' % voteTypeId[2])
45          print('Offensive: %d' % voteTypeId[3])
46          print('Favorite: %d' % voteTypeId[4])
47          print('Close: %d' % voteTypeId[5])
48          print('Reopen: %d' % voteTypeId[6])
49          print('BountyStart: %d' % voteTypeId[7])
50          print('BountyClose: %d' % voteTypeId[8])
51          print('Deletion: %d' % voteTypeId[9])
52          print('Undeletion: %d' % voteTypeId[10])
53          print('Spam: %d' % voteTypeId[11])
54          print('InformModerator: %d' % voteTypeId[12])
55          print('No lines in Votes.xml : %d' % (num_lines * len(file_list)))
56          print('Execution time: %fs' % (exec_time))
```

Listing B.3: Source code for serial data processing

B.4 votetypeid-parallel.py

```
1  #File: votetypeid-serial.py
2  #Date: Fri Aug 11 14:07:24
3  #Envn: Linux Xiang 4.10.0-33-generic #37~16.04.1-Ubuntu SMP  UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
```

```
4
5  from lxml import etree
6  import multiprocessing
7  from multiprocessing import Pool
8  from itertools import chain
9  import time
10
11 def sax_parse(filename, voteTypeId):
12
13         context = etree.iterparse(filename, tag = 'row')
14         for event, element in context:
15                 row_vote_type_id = element.attrib.get('VoteTypeId')
16
17                 #increment the counter on the matched VoteTypeId
18                 for i in range(len(voteTypeId)):
19                         if row_vote_type_id == str(i+1):
20                                 voteTypeId[i] += 1
21                 element.clear()
22                 element.getparent().remove(element)
23
24         return voteTypeId
25
26 if __name__ == "__main__":
27
28
29
30         #files to be parsed
31         file_list = ["/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml",
32                      "/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml",
33                      "/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml",
34                      "/home/xiang/Downloads/FYP/datasets/stackoverflow/Votes.xml"]
35
36         voteTypeId = [0,0,0,0,0,0,0,0,0,0,0,0]
37
38         #setup cores
39         no_cores = multiprocessing.cpu_count()
40         pool = Pool(no_cores)
41
42         #Set start time for parsing
43         start = time.time()
44
45         #parallel parse
46         results = pool.starmap(sax_parse, [(file_list[0],voteTypeId),
47                                            (file_list[1],voteTypeId),
48                                            (file_list[2],voteTypeId),
49                                            (file_list[3],voteTypeId)])
50
51         voteTypeId = [sum(x) for x in zip(*results)]
52
53         end = time.time() #Set end time for parsing
54         exec_time = end - start
55
56
57         num_lines_se = sum(1 for line in open (file_list[0])) - 3
58
59         print('************************\n'+
60               '* Number of VoteTypeId *\n' +
61               '************************')
```

```
62          print('AcceptedByOriginator: %d' % voteTypeId[0])
63          print('UpMod: %d' % voteTypeId[1])
64          print('DownMod: %d' % voteTypeId[2])
65          print('Offensive: %d' % voteTypeId[3])
66          print('Favorite: %d' % voteTypeId[4])
67          print('Close: %d' % voteTypeId[5])
68          print('Reopen: %d' % voteTypeId[6])
69          print('BountyStart: %d' % voteTypeId[7])
70          print('BountyClose: %d' % voteTypeId[8])
71          print('Deletion: %d' % voteTypeId[9])
72          print('Undeletion: %d' % voteTypeId[10])
73          print('Spam: %d' % voteTypeId[11])
74          print('InformModerator: %d' % voteTypeId[12])
75          print('No lines in Votes.xml : %d' % num_lines_se)
76          print('Total lines parsed : %d' % (num_lines_se * len(file_list)))
77          print('Execution time: %fs' % (exec_time))
```

Listing B.4: Source code for parallel data processing

B.5 dom-parse-PostLinks.py

```
1  import time,psutil
2  import xml.etree.ElementTree as ET
3
4  def domParse(handle):
5          document = ET.parse(handle)
6          root = document.getroot()
7          return root
8
9  if __name__ == "__main__":
10         row_parsed = set()
11         start_time = time.time()
12         start_mem = int(psutil.virtual_memory().available/(1024.0 ** 2))
13         # DOM parse starts here
14         with open("/home/xiang/Downloads/FYP1/datasets/stackoverflow/PostLinks.xml") as handle:
15                 for child in domParse(handle):
16                         Id = child.attrib.get("Id")
17                         row_parsed.add(Id)
18     # Print out the report.
19         end_time = time.time()
20         end_mem = int(psutil.virtual_memory().available/(1024.0 ** 2))
21         print("Starting available memory: {} MB".format(start_mem))
22         print("Final available memory: {} MB".format(end_mem))
23         print("Number of Rows: {}".format(len(row_parsed)))
24         print("Time taken: {} s".format(int((end_time - start_time))))
25         print("Memory used: {} MB".format(abs(end_mem - start_mem)))
```

Listing B.5: Source code for DOM parse PostLinks.XML

B.6 sax-parse-PostLinks.py

```
import time,psutil,xml.sax

class UsersHandler(xml.sax.ContentHandler):
        def __init__(self):
                self.Row_Count = 0
        def startElement(self, tag, attributes):
                if tag == "row":
                        self.Row_Count = self.Row_Count + 1

if __name__ == "__main__":
        start_time = time.time()
        start_mem = int(psutil.virtual_memory().available/(1024.0 ** 2))
        # Initiate SAX API
        parser = xml.sax.make_parser()
        parser.setFeature(xml.sax.handler.feature_namespaces, 0)
        Handler = UsersHandler()
        parser.setContentHandler(Handler)
        parser.parse("/home/xiang/Downloads/FYP1/datasets/stackoverflow/PostLinks.xml")
        # Print out the report.
        end_time = time.time()
        end_mem = int(psutil.virtual_memory().available/(1024.0 ** 2))
        print("Starting available memory: {} MB".format(start_mem))
        print("Final available memory: {} MB".format(end_mem))
        print("Number of Rows: {}".format(Handler.Row_Count))
        print("Time taken: {} s".format(int((end_time - start_time))))
        print("Memory used: {} MB".format(abs(end_mem - start_mem)))
```

Listing B.6: Source code for SAX parse PostLinks.XML

B.7 python-sax-parse.py

```
#! /usr/bin/env python

## MODIFIED BY WRY FROM REFERENCE
## https://www.tutorialspoint.com/python/python_xml_processing.htm
## Python Object Oriented Programming

## This is pure SAX processing.
import xml.sax
import datetime
import time
import argparse

argsParser = argparse.ArgumentParser()
argsParser.add_argument("--file", "-f", type=str, required=True)
```

```
15 args = argsParser.parse_args()
16
17 global_Row_Count = 0
18
19 # ======================================================
20 class UsersHandler( xml.sax.ContentHandler ):
21         def __init__(self):
22                 self.Row_Count = 0
23
24         # ======================================================
25         # Call when an element starts, that is the start <tag>.
26         # In our Users.xml case, the start <tag> is the <row> tag.
27         # We do not have any internal <tag>s inside our <row> tag.
28         # We only have attributes between our start <row> and end </row> tag.
29         # The attributes are the "dictionary type", that is, "key-value" pairs.
30         # ======================================================
31         def startElement(self, tag, attributes):
32                 self.CurrentData = tag
33                 if tag == "row":
34                         self.Row_Count = self.Row_Count + 1
35                         pass
36
37
38                 # UPDATE AND PRINT ROW COUNT
39                 # print
40
41                 millionthCount = (self.Row_Count % 1000000)
42
43                 if (millionthCount == 0):
44                         print "ROW NUMBER PROCESSED:\t", self.Row_Count
45                         print "Now time at:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
46
47                 global_Row_Count = self.Row_Count
48
49         # ======================================================
50         # Call when an elements ends i.e. start <tag> until end </tag>
51         # Attributes-Value pairs are inside the tag itself, normally the start <tag>
52         # ======================================================
53         def endElement(self, tag):
54
55                 # RESET THE ELEMENT
56                 self.CurrentData = ""
57
58         # ======================================================
59         # Call when a character is read
60         # The content is the "value" of the element, i.e. what is
61         # written between the start <tag> and end </tag>
62         # In our Users.xml case, contents between <row> and </row>.
63         # ======================================================
64         def characters(self, content):
65                 pass
66
67 if ( __name__ == "__main__"):
68
69         globalStartTime = time.time()
70         print "PARSING", args.file
71         print "\nPROGRAM STARTING:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
72
```

```
73          # create an XMLReader
74          parser = xml.sax.make_parser()
75
76          # turn off namepsaces
77          parser.setFeature(xml.sax.handler.feature_namespaces, 0)
78
79          # override the default ContextHandler
80          Handler = UsersHandler()
81          parser.setContentHandler(Handler)
82
83          # parse file path provided by user input
84          parser.parse(args.file)
85
86          try:
87                  print "\nNUMBER OF ROWS PARSED: ", Handler.Row_Count, "\n"
88          except:
89                  pass
90
91          print "\nPROGRAM FINISHED:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
```

Listing B.7: Source code for Python SAX parse program

B.8 r-sax-parse.r

```
1  #!/usr/bin/env Rscript
2
3  library("XML")
4  library("optparse")
5  library("tictoc")
6
7  globalStartTime = tic(quiet = TRUE)
8  globalStartTime = toc(quiet = TRUE)
9
10 #EVENT HANDLER
11 startElement = function(name, attrs, .state)  {
12   if(name == "row"){
13     .state = .state + 1
14   }
15   if(.state%%1000000 == 0){
16     temp <- tic(quiet = TRUE)
17     temp <- toc(quiet = TRUE)
18     timer <- temp$toc - globalStartTime$tic
19     cat("ROW NUMBER PROCESSED: ", .state , "\n")
20     cat("Now time at: ", format(Sys.time(), "%F %R:%S")
21         , "at" ,timer,"\n\n")
22     tic(quiet = TRUE)
23   }
24   .state
25 };
```

```
26
27 #ARGUMENTS TO SET FILEPATH
28 #REFERENCED FROM
29 #https://www.r-bloggers.com/passing-arguments-to-an-r-script-from-command-lines/
30 option_list = list(
31   make_option(c("-f", "--file"), type="character", default=NULL,
32               help="dataset file name", metavar="character")
33 );
34
35 opt_parser = OptionParser(option_list=option_list);
36 opt = parse_args(opt_parser);
37 if (is.null(opt$file)){
38   print_help(opt_parser)
39   stop("At least one argument must be supplied (input file).n", call.=FALSE)
40 }
41
42 #MAIN PROGRAM
43 #READ FILENAME FROM USER INPUT IN COMMAND LINE
44 fileName <- opt$file
45 cat("PROGRAM STARTING AT: ", format(Sys.time(), "%F %T"),"\n\n")
46 cat("PARSING ",fileName, "\n\n")
47
48 #START SAX PARSING HERE
49 parser <-xmlEventParse(fileName, handlers = list(startElement = startElement), state = 0)
50
51 endTime <- toc(quiet = TRUE)
52 endTime <- endTime$toc - globalStartTime$tic
53 cat("TOTAL ROW PARSED: ",parser,"\n\n")
54 cat("PROGRAM FINISHED AT: ", format(Sys.time(), "%F %T"), "at" ,endTime,"seconds\n")
```

Listing B.8: Source code for R SAX parse program

B.9 keyword-search.py

```
1  #!/usr/bin/env python
2
3  import xml.sax
4  import datetime
5  import time
6  import sys
7  from nltk.tokenize import word_tokenize
8  from nltk.corpus import stopwords
9
10 unix_timestamp = time.time()
11 start_local_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
12 start_file_datetime = start_local_time.strftime("%Y-%m-%d-UTC:%H:%M:%S")
13
14 file_datetime = start_local_time.strftime("%Y%m%d-UTC%H%M%S")
15 file_name = "search-results" + file_datetime + ".txt"
```

```
16
17  f = open(file_name, 'w+')
18
19  global_Row_Count = 0
20  globalPostsFound = 0
21
22  # ======================================================
23  class UsersHandler( xml.sax.ContentHandler ):
24          def __init__(self):
25                  self.Row_Count = 0
26                  self.CurrentData = ""
27
28                  self.Id = ""
29                  self.CreationDate = ""
30                  self.ViewCount = ""
31                  self.Body = ""
32                  self.Title = ""
33                  self.CommentCount = ""
34                  self.OwnerDisplayName = ""
35                  self.LastEditorDisplayName = ""
36
37                  self.word = ""
38                  self.keywordCount = 0
39                  self.postsFound = 0
40
41          # ======================================================
42          # Call when an element starts, that is the start <tag>.
43          # In our Users.xml case, the start <tag> is the <row> tag.
44          # We do not have any internal <tag>s inside our <row> tag.
45          # We only have attributes between our start <row> and end </row> tag.
46          # The attributes are the "dictionary type", that is, "key-value" pairs.
47          # ======================================================
48          def startElement(self, tag, attributes):
49                  self.CurrentData = tag
50                  if tag == "row":
51                          self.Row_Count = self.Row_Count + 1
52                          self.Body = attributes["Body"]
53                          for i in self.word:
54                                  if i in self.Body.lower():
55                                          self.keywordCount += 1
56
57                          if self.keywordCount == len(self.word):
58                                  self.postsFound += 1
59                                  print "============================================================"
60                                  f.write("============================================================"+ "\n")
61                                  try:
62                                          self.Id = attributes["Id"]
63                                          print "Id:\t\t\t", self.Id
64                                          f.write("Id:\t\t\t" + self.Id + "\n" )
65                                  except:
66                                          print "Id:\t\t\tNA"
67                                          f.write("Id:\t\t\tNA\n")
68
69                                  try:
70                                          self.ViewCount = attributes["ViewCount"]
71                                          print "ViewCount:\t\t", self.ViewCount
72                                          f.write("ViewCount:\t\t" + self.ViewCount + "\n" )
73                                  except:
```

```
74                                      print "ViewCount:\t\tNA"
75                                      f.write("ViewCount:\t\tNA\n")
76
77                              try:
78                                      self.CommentCount = attributes["CommentCount"]
79                                      print "CommentCount:\t\t", self.CommentCount
80                                      f.write("CommentCount:\t\t" + self.CommentCount + "\n" )
81                              except:
82                                      print "CommentCount:\t\tNA"
83                                      f.write("CommentCount:\t\tNA\n")
84                              try:
85                                      self.CreationDate = attributes["CreationDate"]
86                                      print "CreationDate:\t\t", self.CreationDate
87                                      f.write("CreationDate:\t\t" + self.CreationDate + "\n" )
88                              except:
89                                      print "CreationDate:\t\tNA"
90                                      f.write("CreationDate:\t\tNA\n")
91
92                              try:
93                                      self.OwnerDisplayName = attributes["OwnerDisplayName"]
94                                      print "OwnerDisplayName:\t", self.OwnerDisplayName
95                                      f.write("OwnerDisplayName:\t" + self.OwnerDisplayName + "\n" )
96                              except:
97                                      print "OwnerDisplayName:\tNA"
98                                      f.write("OwnerDisplayName:\tNA\n")
99
100                             try:
101                                     self.LastEditorDisplayName = attributes["LastEditorDisplayName"]
102                                     print "LastEditorDisplayName:\t", self.LastEditorDisplayName
103                                     f.write("LastEditorDisplayName:\t" + self.LastEditorDisplayName + "\n" )
104                             except:
105                                     print "LastEditorDisplayName:\tNA"
106                                     f.write("LastEditorDisplayName:\tNA\n")
107
108                             try:
109                                     self.Title = attributes["Title"]
110                                     print "Title:\t\t\t", self.Title
111                                     f.write("Title:\t\t\t" + self.Title + "\n" )
112                             except:
113                                     print "Title:\t\t\tNA"
114                                     f.write("Title:\t\t\tNA\n")
115                             try:
116                                     self.Body = attributes["Body"]
117                                     print "Body:\t\t\t", self.Body
118                                     f.write("Body:\t\t\t" + self.Body + "\n" )
119                             except:
120                                     print "Body:\t\t\tNA"
121                                     f.write("Body:\t\t\tNA\n")
122
123                     self.keywordCount = 0
124
125
126             # UPDATE AND PRINT ROW COUNT
127             # print
128
129             millionthCount = (self.Row_Count % 1000000)
130
131
```

```
132                 try:
133                         if (millionthCount == 0) and (self.Row_Count != 0):
134                                 print "ROW NUMBER PROCESSED:\t", self.Row_Count
135                                 f.write("ROW NUMBER PROCESSED:\t" + str(self.Row_Count) + "\n" )
136                                 print "Now time at:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
137                                 f.write("Now time at:"+ str(datetime.datetime.now())+ "at " +str(time.time() - globalStartTime)+ "\n")
138                                 #f.close()
139                                 #f = open("search1000000.txt", 'w+',0)
140                 except:
141                         pass
142
143                 global_Row_Count = self.Row_Count
144                 globalPostsFound = self.postsFound
145         # ==================================================
146         # Call when an elements ends i.e. start <tag> until end </tag>
147         # Attributes-Value pairs are inside the tag itself, normally the start <tag>
148         # ==================================================
149         def endElement(self, tag):
150
151                 # RESET THE ELEMENT
152                 self.CurrentData = ""
153
154         # ==================================================
155         # Call when a character is read
156         # The content is the "value" of the element, i.e. what is
157         # written between the start <tag> and end </tag>
158         # In our Users.xml case, contents between <row> and </row>.
159         # ==================================================
160         def characters(self, content):
161                 pass
162
163
164 if ( __name__ == "__main__"):
165
166         globalStartTime = time.time()
167         print "Parsing Posts.xml...\n"
168         print "PROGRAM STARTING:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
169
170         # create an XMLReader
171         parser = xml.sax.make_parser()
172
173         # turn off namepsaces
174         parser.setFeature(xml.sax.handler.feature_namespaces, 0)
175
176         # override the default ContextHandler
177         Handler = UsersHandler()
178         parser.setContentHandler(Handler)
179
180         # ask users to enter search keyword
181         keyword = raw_input("Please enter search keyword: ")
182         keyword = keyword.lower()
183
184         # use stopwords to filter sentence to improve searchability
185         stop = set(stopwords.words('english'))
186         tokenizedKeyword = word_tokenize(keyword)
187         tokenizedKeyword = [i for i in tokenizedKeyword if i not in stop]
188         print "\nSearching for keywords: ", tokenizedKeyword
189         Handler.word = tokenizedKeyword
```

```
190         try:
191                 #parser.parse("test.xml")
192                 parser.parse("/home/xiang/Downloads/FYP1/datasets/stackoverflow/Posts.xml")
193         except:
194                 pass
195
196         try:
197                 print "\nNumber of rows parsed: ", Handler.Row_Count, "\n"
198         except:
199                 pass
200
201         print "\nNumber of results: ", Handler.postsFound,"\n"
202         print "\nPROGRAM FINISHED:", datetime.datetime.now(), "at %s " %(time.time() - globalStartTime), "\n"
203         f.close()
```

LISTING B.9: Source code for keyword search program

B.10 sequential-concurrent-parallel-sax-plot.py

```
1  import pandas as pd
2  from matplotlib import pyplot as plt
3  import numpy as np
4  #============================+
5  # Sequential SAX PostHistory |
6  #============================+
7  postHistorySequential = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-sequential-R1.csv")
8  postHistorySequential['row_count'] = postHistorySequential['row_count']/1000000
9
10 #=====================+
11 # Sequential SAX Posts |
12 #=====================+
13 postsSequential = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-sequential-R1.csv")
14 postsSequential['row_count'] = postsSequential['row_count']/1000000
15
16 newPostsSequential = pd.DataFrame(postsSequential)
17 newPostsSequential.drop(newPostsSequential.index[0], inplace=True)
18 newPostsSequential['row_count'] += float(postHistorySequential.tail(1)['row_count'])
19 newPostsSequential['process_time_sec'] += int(postHistorySequential.tail(1)['process_time_sec'])
20
21 #concatenate two dataframes
22 resultSequential = pd.concat([postHistorySequential,newPostsSequential])
23 resultSequential= resultSequential.reset_index(drop=True)
24
25 #==================================+
26 # Concurrent SAX PostHistory & Posts |
27 #==================================+
28 postHistoryConcurrent = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-concurrent-R1.csv")
29 postHistoryConcurrent['row_count'] = postHistoryConcurrent['row_count']/1000000
30
```

```
postsConcurrent = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-concurrent-R1.csv")
postsConcurrent['row_count'] = postsConcurrent['row_count']/1000000

#==================================+
#  Parallel SAX PostHistory & Posts  |
#==================================+
postHistoryParallel = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-parallel-R1.csv")
postHistoryParallel['row_count'] = postHistoryParallel['row_count']/1000000

postsParallel = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-parallel-R1.csv")
postsParallel['row_count'] = postsParallel['row_count']/1000000

#================+
# Graph Plotting |
#================+
fig = plt.figure(figsize=(11,8))
ax = fig.add_subplot(111)

#Sequential Line
ax.plot(resultSequential['process_time_sec'], resultSequential['row_count'],label = 'Sequential Posts+PostHistory', color = 'b')

#Concurrent Lines
ax.plot(postHistoryConcurrent['process_time_sec'], postHistoryConcurrent['row_count'],label='Concurrent PostHistory', color='r')
ax.plot(postsConcurrent['process_time_sec'], postsConcurrent['row_count'],label='Concurrent Posts', color='r', marker = 'x')

#Parallel Lines
ax.plot(postHistoryParallel['process_time_sec'], postHistoryParallel['row_count'],label='Parallel PostHistory', color='c')
ax.plot(postsParallel['process_time_sec'], postsParallel['row_count'],label='Parallel Posts', color='c', marker = 'x')

#Setting up graph's parameters
plt.xticks(np.arange(min(resultSequential['process_time_sec']), max(resultSequential['process_time_sec'])+300 , 300))
plt.yticks(np.arange(min(resultSequential['row_count']), max(resultSequential['row_count'])+10 , 10))

plt.ylim(ymin=0)
plt.xlim(xmin=0)
plt.xlabel('Process Time (sec)')
plt.ylabel('Row Count (million)')
ax.set_title('Performance Comparison of Parallel, Concurrent & Sequential SAX Parse',fontweight = "bold",fontsize = 15)
handles, labels = ax.get_legend_handles_labels()
lgd = ax.legend(handles, labels, loc='upper left')
ax.grid('on')
ax.annotate('Posts.XML starts here', xy=(newPostsSequential.loc[1]['process_time_sec'],newPostsSequential.loc[1]['row_count']),  xycoords='data',
            xytext=(0.4, 0.8), textcoords='axes fraction',
            arrowprops=dict(facecolor='black'),
            horizontalalignment='right', verticalalignment='top',
            )
ax.annotate('Parsing speed increased here', xy=(postsConcurrent.loc[37]['process_time_sec'],59),  xycoords='data',
            xytext=(0.7,0.5), textcoords='axes fraction',
            arrowprops=dict(facecolor='black'),
            horizontalalignment='right', verticalalignment='top',
            )
plt.vlines(postsConcurrent.loc[37]['process_time_sec'], 0, 59, linestyle="dashed")
plt.hlines(postsConcurrent.loc[37]['row_count'], 0,3900, linestyle="dashed")
plt.hlines(postHistoryConcurrent.loc[94]['row_count'], 0,3900, linestyle="dashed")
plt.savefig('Performance Comparison of Parallel, Concurrent & Sequential SAX Parse.png')
```

Listing B.10: Source code for plotting performance comparison of parallel, concurrent and sequential SAX parse line graph

B.11 local-vs-remote-sax-plot.py

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
#==================================+
# Sequential SAX PostHistory Remote |
#==================================+
postHistorySequentialRemote = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV-remote-corrected/PostHistory-sequential
    .csv")
postHistorySequentialRemote['row_count'] = postHistorySequentialRemote['row_count']/1000000
postHistorySequentialRemote['process_time_sec'] = postHistorySequentialRemote['process_time_sec']/3600

#==================================+
# Sequential SAX PostHistory Local |
#==================================+
postHistorySequentialLocal = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-sequential-R1.csv")
postHistorySequentialLocal['row_count'] = postHistorySequentialLocal['row_count']/1000000
postHistorySequentialLocal['process_time_sec'] = postHistorySequentialLocal['process_time_sec']/3600

#============================+
# Sequential SAX Posts Remote |
#============================+
postsSequentialRemote = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV-remote-corrected/Posts-sequential.csv")
postsSequentialRemote['row_count'] = postsSequentialRemote['row_count']/1000000
postsSequentialRemote['process_time_sec'] = postsSequentialRemote['process_time_sec']/3600

newpostsSequentialRemote = pd.DataFrame(postsSequentialRemote)
newpostsSequentialRemote.drop(newpostsSequentialRemote.index[0], inplace=True)
newpostsSequentialRemote['row_count'] += float(postHistorySequentialRemote.tail(1)['row_count'])
newpostsSequentialRemote['process_time_sec'] += float(postHistorySequentialRemote.tail(1)['process_time_sec'])

#concatenate two dataframes
resultSequentialRemote = pd.concat([postHistorySequentialRemote,newpostsSequentialRemote])
resultSequentialRemote= resultSequentialRemote.reset_index(drop=True)

#===========================+
# Sequential SAX Posts Local |
#===========================+
postsSequentialLocal = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-sequential-R1.csv")
postsSequentialLocal['row_count'] = postsSequentialLocal['row_count']/1000000
postsSequentialLocal['process_time_sec'] = postsSequentialLocal['process_time_sec']/3600

newpostsSequentialLocal = pd.DataFrame(postsSequentialLocal)
newpostsSequentialLocal.drop(newpostsSequentialLocal.index[0], inplace=True)
newpostsSequentialLocal['row_count'] += float(postHistorySequentialLocal.tail(1)['row_count'])
newpostsSequentialLocal['process_time_sec'] += float(postHistorySequentialLocal.tail(1)['process_time_sec'])

#concatenate two dataframes
resultSequentialLocal = pd.concat([postHistorySequentialLocal,newpostsSequentialLocal])
resultSequentialLocal= resultSequentialLocal.reset_index(drop=True)

#=======================================+
# Concurrent SAX PostHistory & Posts Local |
#=======================================+
postHistoryConcurrentLocal = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-concurrent-R1.csv")
```

```
54 postHistoryConcurrentLocal['row_count'] = postHistoryConcurrentLocal['row_count']/1000000
55 postHistoryConcurrentLocal['process_time_sec'] = postHistoryConcurrentLocal['process_time_sec']/3600
56
57 postsConcurrentLocal = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-concurrent-R1.csv")
58 postsConcurrentLocal['row_count'] = postsConcurrentLocal['row_count']/1000000
59 postsConcurrentLocal['process_time_sec'] = postsConcurrentLocal['process_time_sec']/3600
60
61 #=========================================+
62 # Concurrent SAX PostHistory & Posts Remote |
63 #=========================================+
64 postHistoryConcurrentRemote = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV-remote-corrected/PostHistory-concurrent
       .csv")
65 postHistoryConcurrentRemote['row_count'] = postHistoryConcurrentRemote['row_count']/1000000
66 postHistoryConcurrentRemote['process_time_sec'] = postHistoryConcurrentRemote['process_time_sec']/3600
67
68 postsConcurrentRemote = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV-remote-corrected/Posts-concurrent.csv")
69 postsConcurrentRemote['row_count'] = postsConcurrentRemote['row_count']/1000000
70 postsConcurrentRemote['process_time_sec'] = postsConcurrentRemote['process_time_sec']/3600
71
72 #================+
73 # Graph Plotting |
74 #================+
75 fig = plt.figure(figsize=(11,8))
76 ax = fig.add_subplot(111)
77
78 #PostHistory Line Graph Local
79 ax.plot(resultSequentialLocal['process_time_sec'], resultSequentialLocal['row_count'],label = 'Local Sequential Posts+PostsHistory', color = 'm')
80 ax.plot(postHistoryConcurrentLocal['process_time_sec'], postHistoryConcurrentLocal['row_count'],label='Local Concurrent PostsHistory', color='y', marker = '*')
81 ax.plot(postsConcurrentLocal['process_time_sec'], postsConcurrentLocal['row_count'],label='Local Concurrent Posts', color='y', marker = 'x')
82
83 #PostHistory Line Graph Remote
84 ax.plot(resultSequentialRemote['process_time_sec'], resultSequentialRemote['row_count'],label = 'Remote Sequential Posts+PostsHistory', color = 'b')
85 ax.plot(postHistoryConcurrentRemote['process_time_sec'], postHistoryConcurrentRemote['row_count'],label='Remote Concurrent PostsHistory', color='r', marker = '*')
86 ax.plot(postsConcurrentRemote['process_time_sec'], postsConcurrentRemote['row_count'],label='Remote Concurrent Posts', color='r', marker = 'x')
87
88 #Setting up graph's parameters
89 plt.xticks(np.arange(min(postHistoryConcurrentRemote['process_time_sec']), max(postHistoryConcurrentRemote['process_time_sec'])+1 , 1))
90 plt.yticks(np.arange(min(resultSequentialRemote['row_count']), max(resultSequentialRemote['row_count'])+10 , 10))
91
92 plt.ylim(ymin=0)
93 plt.xlim(xmin=0)
94 plt.xlabel('Process Time (hr)')
95 plt.ylabel('Row Count (million)')
96 ax.set_title('Performance Comparison of SAX Parse between Local & Remote',fontweight = "bold",fontsize = 15)
97 handles, labels = ax.get_legend_handles_labels()
98 lgd = ax.legend(handles, labels, loc='lower right')
99 ax.grid('on')
100 ax.annotate('Local Posts.XML starts here', xy=(newpostsSequentialLocal.loc[1]['process_time_sec'],newpostsSequentialLocal.loc[1]['row_count']),  xycoords='data',
101             xytext=(0.4, 0.9), textcoords='axes fraction',
102             arrowprops=dict(facecolor='black'),
103             horizontalalignment='right', verticalalignment='top',)
104 ax.annotate('Remote Posts.XML starts here', xy=(newpostsSequentialRemote.loc[1]['process_time_sec'],newpostsSequentialRemote.loc[1]['row_count']),  xycoords='data',
105             xytext=(0.4, 0.8), textcoords='axes fraction',
106             arrowprops=dict(facecolor='black'),
107             horizontalalignment='right', verticalalignment='top',)
108 ax.annotate('Parsing speed increased here', xy=(postsConcurrentRemote.loc[37]['process_time_sec'],59),  xycoords='data',
109             xytext=(0.55,0.5), textcoords='axes fraction',
110             arrowprops=dict(facecolor='black'),
```

```
111             horizontalalignment='right', verticalalignment='top',)
112
113 plt.vlines(postsConcurrentRemote.loc[37]['process_time_sec'], 0, 59, linestyle="dashed")
114 plt.vlines(postsConcurrentLocal.loc[37]['process_time_sec'], 0, 59, linestyle="dashed")
115 plt.hlines(postsConcurrentLocal.loc[37]['row_count'], 0,3900, linestyle="dashed")
116 plt.hlines(postHistoryConcurrentLocal.loc[94]['row_count'], 0,3900, linestyle="dashed")
117 plt.hlines(resultSequentialLocal.loc[131]['row_count'], 0,3900, linestyle="dashed")
118 plt.savefig('Performance Comparison of SAX Parse of Local & Remote.png')
```

Listing B.11: Source code for plotting performance comparison of local and remote SAX parse line graph

B.12 r-vs-python-sax-plot.py

```
1  import pandas as pd
2  from matplotlib import pyplot as plt
3  import numpy as np
4  #=======================================+
5  # Python Sequential SAX PostHistory & Posts |
6  #=======================================+
7  postHistorySequentialPy = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-sequential-R1.csv")
8  postHistorySequentialPy['row_count'] = postHistorySequentialPy['row_count']/1000000
9
10 postsSequentialPy = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-sequential-R1.csv")
11 postsSequentialPy['row_count'] = postsSequentialPy['row_count']/1000000
12
13 newPostsSequentialPy = pd.DataFrame(postsSequentialPy)
14 newPostsSequentialPy.drop(newPostsSequentialPy.index[0], inplace=True)
15 newPostsSequentialPy['row_count'] += float(postHistorySequentialPy.tail(1)['row_count'])
16 newPostsSequentialPy['process_time_sec'] += int(postHistorySequentialPy.tail(1)['process_time_sec'])
17
18 #concatenate two dataframes
19 resultSequentialPy = pd.concat([postHistorySequentialPy,newPostsSequentialPy])
20 resultSequentialPy= resultSequentialPy.reset_index(drop=True)
21
22 #=======================================+
23 # Python Concurrent SAX PostHistory & Posts |
24 #=======================================+
25 postHistoryConcurrentPy = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-concurrent-R1.csv")
26 postHistoryConcurrentPy['row_count'] = postHistoryConcurrentPy['row_count']/1000000
27
28 postsConcurrentPy = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-concurrent-R1.csv")
29 postsConcurrentPy['row_count'] = postsConcurrentPy['row_count']/1000000
30
31 #=======================================+
32 # Python Parallel SAX PostHistory & Posts  |
33 #=======================================+
34 postHistoryParallelPy = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/PostHistory-CSV/PostHistory-parallel-R1.csv")
35 postHistoryParallelPy['row_count'] = postHistoryParallelPy['row_count']/1000000
36
```

```
postsParallelPy = pd.read_csv("/home/xiang/Downloads/FYP2/python-sax-parse-concurrent-sequential/Posts-CSV/Posts-parallel-R1.csv")
postsParallelPy['row_count'] = postsParallelPy['row_count']/1000000

#====================================+
# R Sequential SAX PostHistory & Posts |
#====================================+
postHistorySequentialR = pd.read_csv("/home/xiang/Downloads/FYP2/r-sax-parse/CSV/PostHistory-sequential.csv")
postHistorySequentialR['row_count'] = postHistorySequentialR['row_count']/1000000

postsSequentialR = pd.read_csv("/home/xiang/Downloads/FYP2/r-sax-parse/CSV/Posts-sequential.csv")
postsSequentialR['row_count'] = postsSequentialR['row_count']/1000000

newPostsSequentialR = pd.DataFrame(postsSequentialR)
newPostsSequentialR.drop(newPostsSequentialR.index[0], inplace=True)
newPostsSequentialR['row_count'] += float(postHistorySequentialR.tail(1)['row_count'])
newPostsSequentialR['process_time_sec'] += int(postHistorySequentialR.tail(1)['process_time_sec'])

#concatenate two dataframes
resultSequentialR = pd.concat([postHistorySequentialR,newPostsSequentialR])
resultSequentialR= resultSequentialR.reset_index(drop=True)

#====================================+
# R Concurrent SAX PostHistory & Posts |
#====================================+
postHistoryConcurrentR = pd.read_csv("/home/xiang/Downloads/FYP2/r-sax-parse/CSV/PostHistory-concurrent.csv")
postHistoryConcurrentR['row_count'] = postHistoryConcurrentR['row_count']/1000000

postsConcurrentR = pd.read_csv("/home/xiang/Downloads/FYP2/r-sax-parse/CSV/Posts-concurrent.csv")
postsConcurrentR['row_count'] = postsConcurrentR['row_count']/1000000

#====================================+
# R Parallel SAX PostHistory & Posts  |
#====================================+
postHistoryParallelR = pd.read_csv("/home/xiang/Downloads/FYP2/r-sax-parse/CSV/PostHistory-parallel.csv")
postHistoryParallelR['row_count'] = postHistoryParallelR['row_count']/1000000

postsParallelR = pd.read_csv("/home/xiang/Downloads/FYP2/r-sax-parse/CSV/Posts-parallel.csv")
postsParallelR['row_count'] = postsParallelR['row_count']/1000000

#================+
# Graph Plotting |
#================+
fig = plt.figure(figsize=(11,8))
ax = fig.add_subplot(111)

#Python Sequential Line
ax.plot(resultSequentialPy['process_time_sec'], resultSequentialPy['row_count'],label = 'Python Sequential Posts+PostHistory', color = 'b')

#Python Concurrent Lines
ax.plot(postHistoryConcurrentPy['process_time_sec'], postHistoryConcurrentPy['row_count'],label='Python Concurrent PostHistory', color='r')
ax.plot(postsConcurrentPy['process_time_sec'], postsConcurrentPy['row_count'],label='Python Concurrent Posts', color='r', marker = 'x')

#Python Parallel Lines
ax.plot(postHistoryParallelPy['process_time_sec'], postHistoryParallelPy['row_count'],label='Python Parallel PostHistory', color='c')
ax.plot(postsParallelPy['process_time_sec'], postsParallelPy['row_count'],label='Python Parallel Posts', color='c', marker = 'x')

#R Sequential Line
ax.plot(resultSequentialR['process_time_sec'], resultSequentialR['row_count'],label = 'R Sequential Posts+PostHistory', color = 'violet')
```

```
95
96  #R Concurrent Lines
97  ax.plot(postHistoryConcurrentR['process_time_sec'], postHistoryConcurrentR['row_count'],label='R Concurrent PostHistory', color='orange')
98  ax.plot(postsConcurrentR['process_time_sec'], postsConcurrentR['row_count'],label='R Concurrent Posts', color='orange', marker = 'x')
99
100 #R Parallel Lines
101 ax.plot(postHistoryParallelR['process_time_sec'], postHistoryParallelR['row_count'],label='R Parallel PostHistory', color='green')
102 ax.plot(postsParallelR['process_time_sec'], postsParallelR['row_count'],label='R Parallel Posts', color='green', marker = 'x')
103
104 #Setting up graph's parameters
105 plt.xticks(np.arange(min(resultSequentialPy['process_time_sec']), max(resultSequentialPy['process_time_sec'])+300 , 300))
106 plt.yticks(np.arange(min(resultSequentialPy['row_count']), max(resultSequentialPy['row_count'])+10 , 10))
107
108 plt.ylim(ymin=0)
109 plt.xlim(xmin=0)
110 plt.xlabel('Process Time (sec)')
111 plt.ylabel('Row Count (million)')
112 ax.set_title('Performance Comparison of SAX Parse between R & Python',fontweight = "bold",fontsize = 15)
113 handles, labels = ax.get_legend_handles_labels()
114 lgd = ax.legend(handles, labels, loc='upper left')
115 ax.grid('on')
116 ax.annotate('Posts.XML starts here', xy=(newPostsSequentialPy.loc[1]['process_time_sec'],newPostsSequentialPy.loc[1]['row_count']),  xycoords='data',
117             xytext=(0.6, 0.9), textcoords='axes fraction',
118             arrowprops=dict(facecolor='black'),
119             horizontalalignment='right', verticalalignment='top',
120             )
121 ax.annotate('Posts.XML starts here', xy=(newPostsSequentialR.loc[1]['process_time_sec'],newPostsSequentialPy.loc[1]['row_count']),  xycoords='data',
122             xytext=(0.6, 0.9), textcoords='axes fraction',
123             arrowprops=dict(facecolor='black'),
124             horizontalalignment='right', verticalalignment='top',
125             )
126 ax.annotate('Parsing speed increased here', xy=(postsConcurrentPy.loc[37]['process_time_sec'],59),  xycoords='data',
127             xytext=(0.98,0.3), textcoords='axes fraction',
128             arrowprops=dict(facecolor='black'),
129             horizontalalignment='right', verticalalignment='top',
130             )
131 ax.annotate('Parsing speed increased here', xy=(postsConcurrentR.loc[37]['process_time_sec'],59),  xycoords='data',
132             xytext=(0.98,0.3), textcoords='axes fraction',
133             arrowprops=dict(facecolor='black'),
134             horizontalalignment='right', verticalalignment='top',
135             )
136 plt.vlines(postsConcurrentPy.loc[37]['process_time_sec'], 0, 59, linestyle="dashed")
137 plt.vlines(postsConcurrentR.loc[37]['process_time_sec'], 0, 59, linestyle="dashed")
138 plt.hlines(postsConcurrentR.loc[37]['row_count'], 0,3900, linestyle="dashed")
139 plt.hlines(postHistoryConcurrentR.loc[94]['row_count'], 0,3900, linestyle="dashed")
140 plt.hlines(resultSequentialR.loc[131]['row_count'], 0,3900, linestyle="dashed")
141 plt.savefig('Performance Comparison of SAX Parse between R & Python.png')
```

Listing B.12: Source code for plotting performance comparison of R and Python SAX parse line graph

Appendix C

R Installation


```
============================================================
(1) Install R Language on Ubuntu 16.04
============================================================
xiang@Xiang:~$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9
[sudo] password for xiang:
Executing: /tmp/tmp.FT59jdSDL0/gpg.1.sh --keyserver
keyserver.ubuntu.com
--recv-keys
E298A3A825C0D65DFD57CBB651716619E084DAB9
gpg: requesting key E084DAB9 from hkp server keyserver.ubuntu.com
gpg: key E084DAB9: public key "Michael Rutter <marutter@gmail.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)

============================
xiang@Xiang:~$ sudo add-apt-repository 'deb [arch=amd64,i386] https://cran.rstudio.com/bin/linux/ubuntu xenial/'
xiang@Xiang:~$ sudo apt-get update
....
Get:8 https://cran.rstudio.com/bin/linux/ubuntu xenial/ InRelease [3,590 B]
Get:9 https://cran.rstudio.com/bin/linux/ubuntu xenial/ Packages [54.4 kB]
....
Fetched 4,149 kB in 4s (999 kB/s)
Reading package lists... Done
xiang@Xiang:~$

============================
xiang@Xiang:~$ sudo apt-get install r-base
Reading package lists... Done
Building dependency tree
```

```
Reading state information... Done
The following additional packages will be installed:
  r-base-core r-recommended
The following packages will be upgraded:
  r-base r-base-core r-recommended
3 upgraded, 0 newly installed, 0 to remove and 47 not upgraded.
Need to get 23.2 MB of archives.
After this operation, 2,903 kB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 https://cran.rstudio.com/bin/linux/ubuntu xenial/ r-base-core 3.4.1-2xenial0 [23.1 MB]
Get:2 https://cran.rstudio.com/bin/linux/ubuntu xenial/ r-base 3.4.1-2xenial0 [40.8 kB]
Get:3 https://cran.rstudio.com/bin/linux/ubuntu xenial/ r-recommended 3.4.1-2xenial0 [2,722 B]
Fetched 23.2 MB in 5s (4,105 kB/s)
....
Setting up r-base-core (3.4.1-2xenial0) ...
Installing new version of config file /etc/R/Makeconf ...
Installing new version of config file /etc/R/Renviron.site ...
Installing new version of config file /etc/R/ldpaths ...
Installing new version of config file /etc/R/repositories ...
Replacing config file /etc/R/Renviron with new version
Setting up r-recommended (3.4.1-2xenial0) ...
Setting up r-base (3.4.1-2xenial0) ...
xiang@Xiang:~$
============================
```

Listing C.1: R Installation Steps

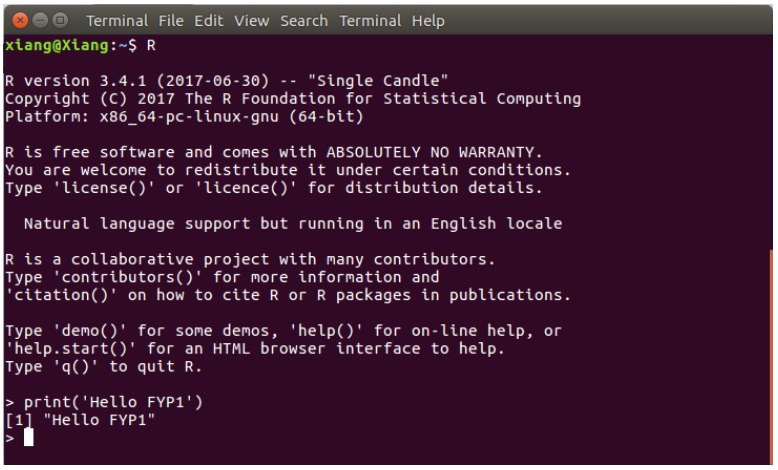


FIGURE C.1: Running R in terminal

After successfully installed R, you should be able to run it in terminal using *R* command.

Appendix D

RStudio Installation

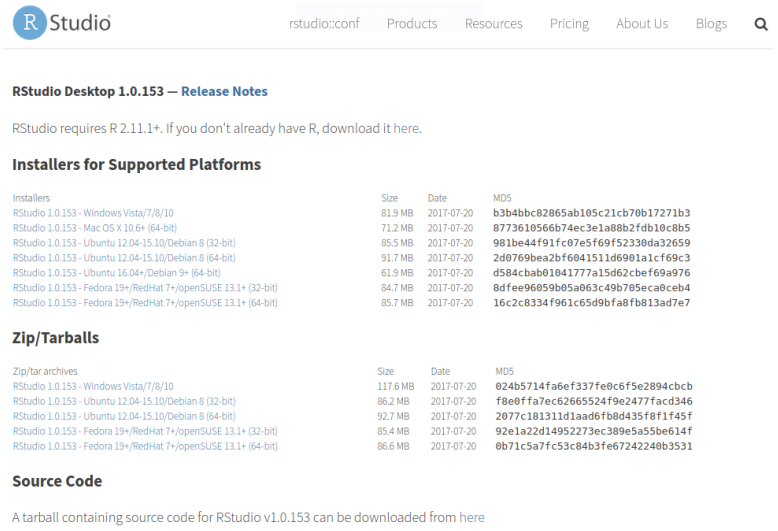


FIGURE D.1: RStudio installer in official website

Step 1 - Download installer from RStudio official website.

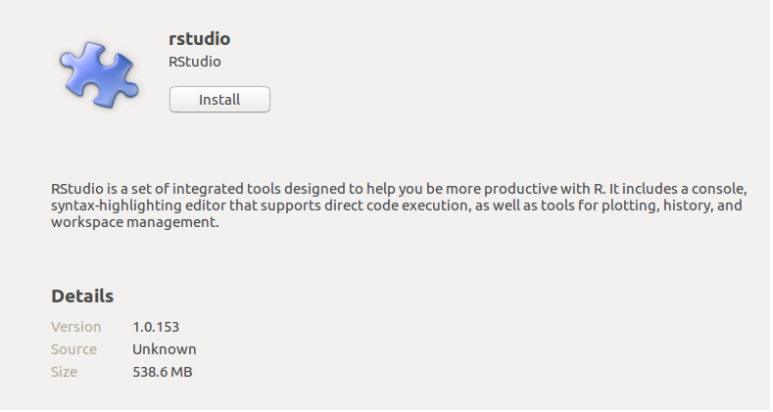


FIGURE D.2: Install RStudio from Ubuntu Software

Step 2 - Run the installer and press *Install*.

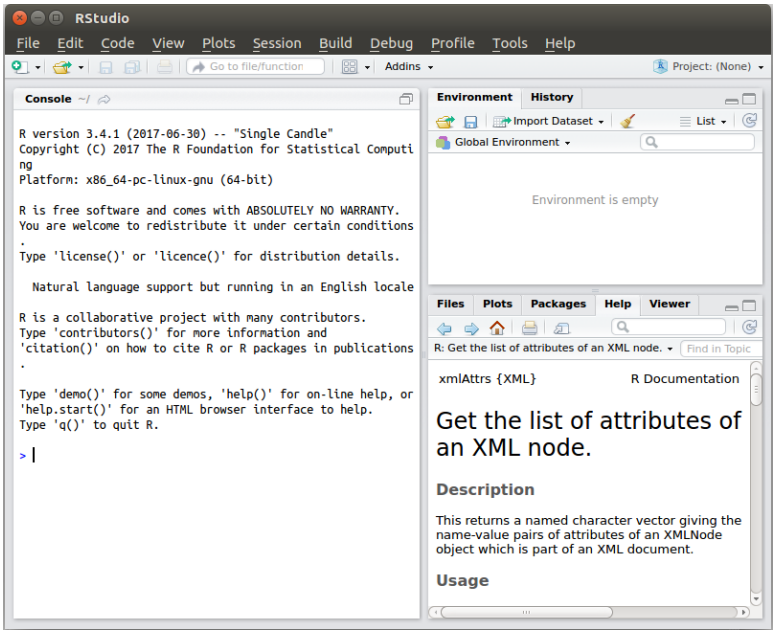


FIGURE D.3: Running RStudio

Step 3 - Run RStudio on your machine.

Appendix E

PyDev Installation

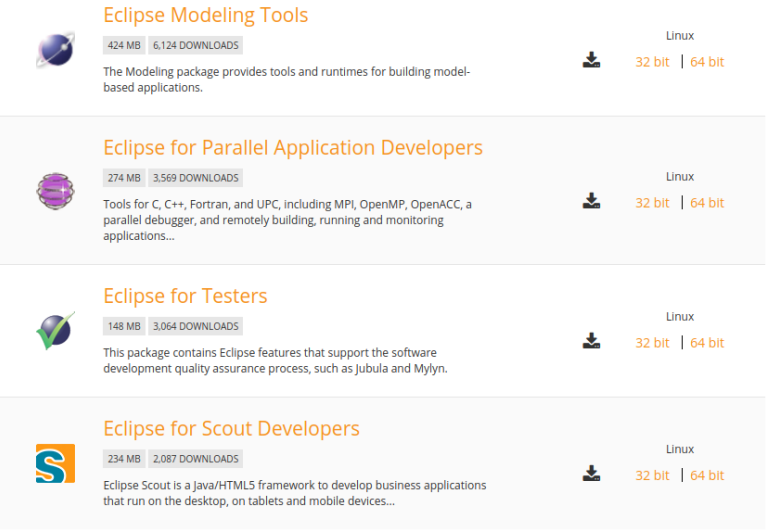


FIGURE E.1: Eclipse installer in official website

Step 1 - Download installer from Eclipse official website.

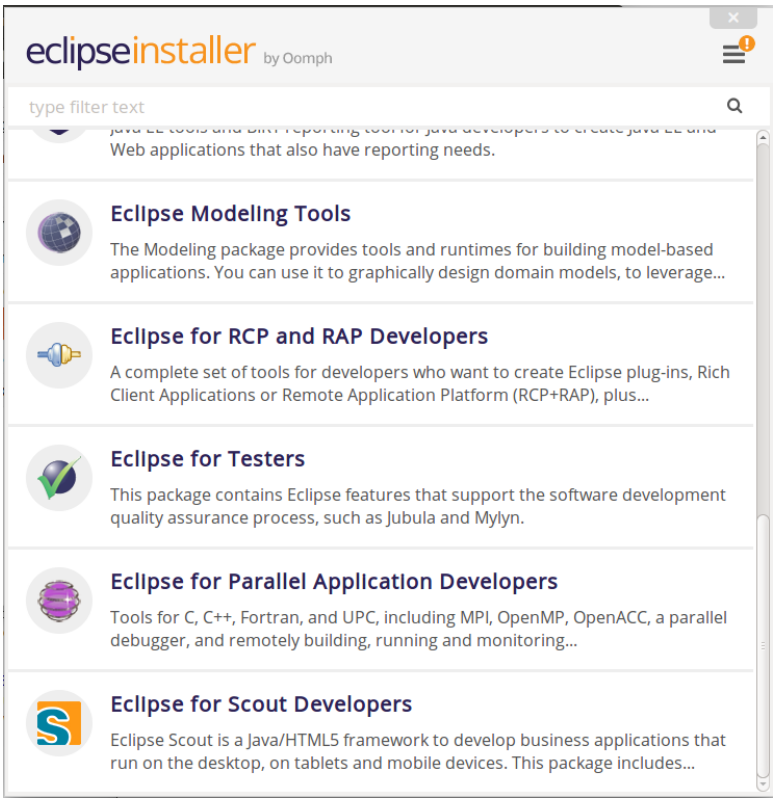


FIGURE E.2: Running Eclipse installer

Step 2 - Extract the installer and run *eclipse-inst*. A window like this would show up. Choose *Eclipse for Parallel Application Developers* and install it.

.

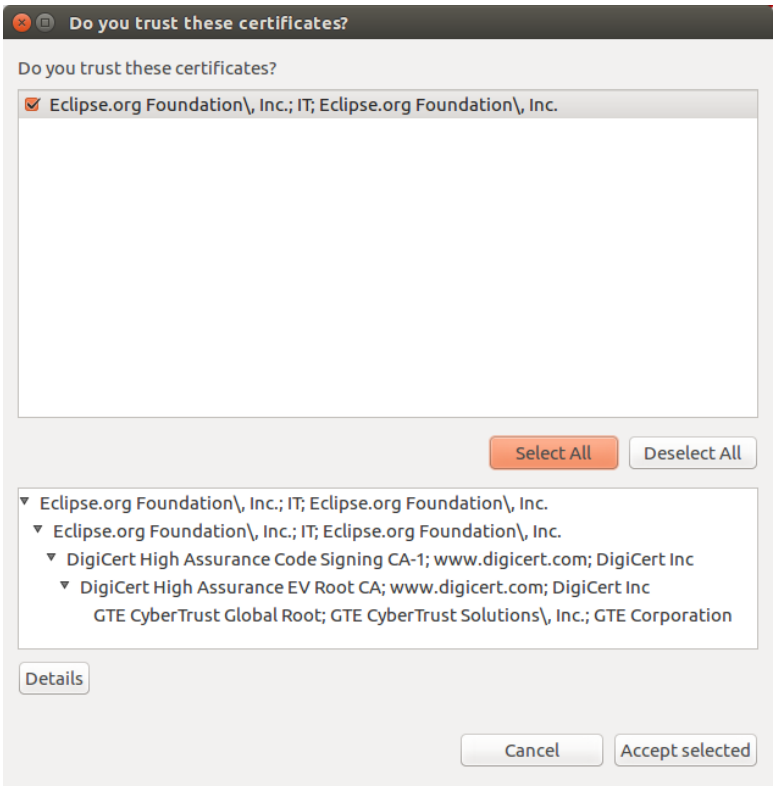


FIGURE E.3: Installing Eclipse

Step 3 - When prompted, tick the certificate and proceed to installation.

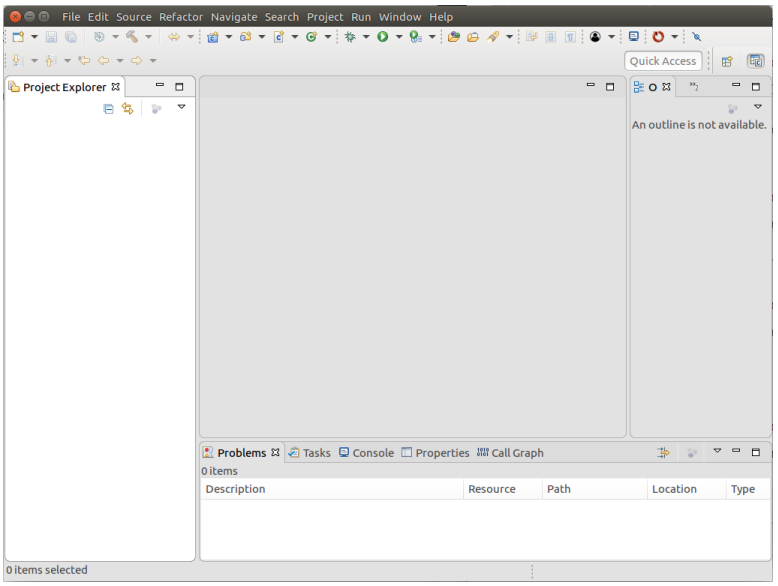


FIGURE E.4: Running Eclipse

Step 4 - Start your Eclipse.

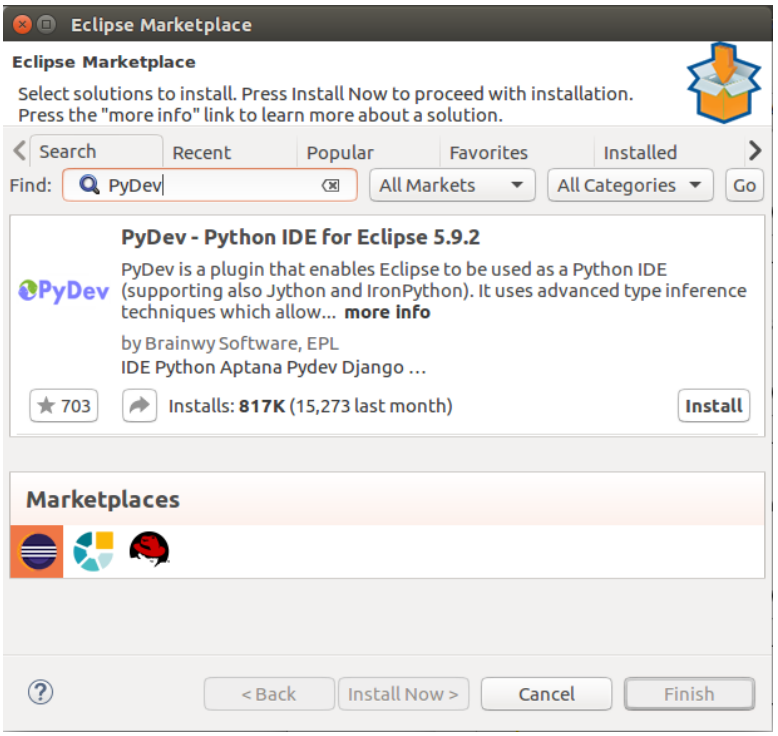


FIGURE E.5: Installing PyDev for Eclipse

Step 5 - Go to *Help-›Eclipse Marketplace* and search for *PyDev* for installation.

Step 6 - Restart Eclipse.

Step 7 - Go to *Window-›Preferences-›PyDev-›Interpreters-›Python Interpreter-›New*.

Step 8 - Set the Python Interpreter to python3.5.

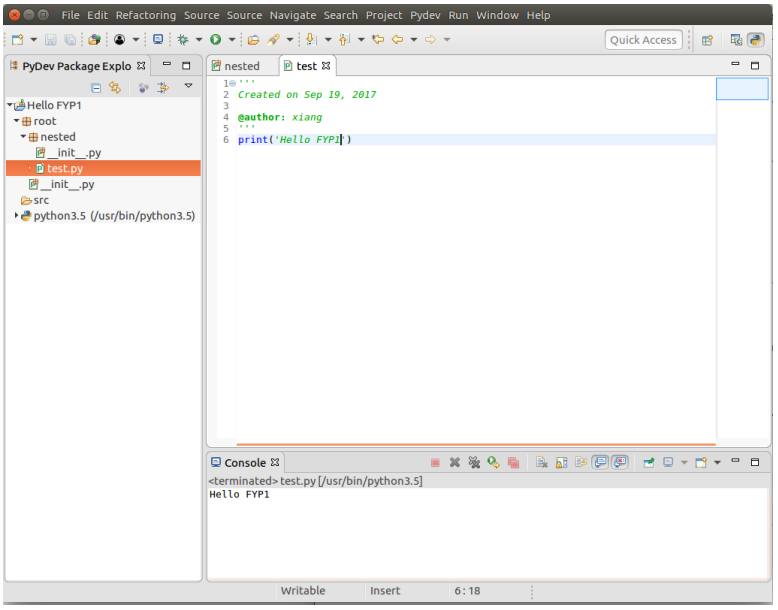


FIGURE E.6: Programming in Eclipse using PyDev

Now you can create Python program in Eclipse.

Appendix F

Technical Specifications

F.1 Personal Computer

```
xiang@Xiang:~$ uname -a
Linux Xiang 4.13.0-32-generic #35~16.04.1-Ubuntu SMP Thu Jan 25 10:13:43 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

============================================================
xiang@Xiang:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 69
model name      : Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
stepping        : 1
microcode       : 0x17
cpu MHz         : 2394.613
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
```

```
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp
     lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16
     xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti tpr_shadow vnmi flexpriority ept vpid
     fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts
bugs            : cpu_meltdown spectre_v1 spectre_v2
bogomips        : 4789.22
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 69
model name      : Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
stepping        : 1
microcode       : 0x17
cpu MHz         : 2394.613
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 2
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp
     lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16
     xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti tpr_shadow vnmi flexpriority ept vpid
     fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts
bugs            : cpu_meltdown spectre_v1 spectre_v2
bogomips        : 4789.22
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 69
model name      : Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
stepping        : 1
microcode       : 0x17
cpu MHz         : 2394.613
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 1
initial apicid  : 1
```

```
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp
     lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16
     xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti tpr_shadow vnmi flexpriority ept vpid
     fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts
bugs            : cpu_meltdown spectre_v1 spectre_v2
bogomips        : 4789.22
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 69
model name      : Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
stepping        : 1
microcode       : 0x17
cpu MHz         : 2394.613
cache size      : 4096 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 2
apicid          : 3
initial apicid  : 3
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp
     lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16
     xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti tpr_shadow vnmi flexpriority ept vpid
     fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts
bugs            : cpu_meltdown spectre_v1 spectre_v2
bogomips        : 4789.22
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

============================================================
xiang@Xiang:~$ cat /proc/meminfo
MemTotal:        8049328 kB
MemFree:          191832 kB
MemAvailable:     435228 kB
Buffers:           87652 kB
Cached:          1445852 kB
SwapCached:         2460 kB
Active:          6356108 kB
Inactive:        1098332 kB
Active(anon):    6150940 kB
Inactive(anon):   897060 kB
Active(file):     205168 kB
```

```
Inactive(file):   201272 kB
Unevictable:         752 kB
Mlocked:             752 kB
SwapTotal:       8269820 kB
SwapFree:        8168944 kB
Dirty:               568 kB
Writeback:             0 kB
AnonPages:       5919336 kB
Mapped:           847008 kB
Shmem:           1127064 kB
Slab:             180876 kB
SReclaimable:     114504 kB
SUnreclaim:        66372 kB
KernelStack:       18752 kB
PageTables:       105780 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    12294484 kB
Committed_AS:   19518456 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:      300484 kB
DirectMap2M:     7968768 kB
DirectMap1G:           0 kB

============================================================
HARD DISK
============================================================
xiang@Xiang:~$ sudo fdisk -l
[sudo] password for xiang:
Disk /dev/sda: 931.5 GiB, 1000204886016 bytes, 1953525168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: 0C2B7FEE-C350-4E3E-8265-950BB58D8F06

Device          Start        End    Sectors   Size Type
/dev/sda1        2048     923647     921600   450M Windows recovery environment
/dev/sda2      923648    1128447     204800   100M EFI System
/dev/sda3     1128448    1161215      32768    16M Microsoft reserved
/dev/sda4     1161216 1235691519 1234530304 588.7G Microsoft basic data
/dev/sda5  1235693568 1935953919  700260352 333.9G Linux filesystem
/dev/sda6  1935953920 1952493567   16539648   7.9G Linux swap
/dev/sda7  1952493568 1953521663    1028096   502M Windows recovery environment
```

```
Disk /dev/sdb: 238.5 GiB, 256060514304 bytes, 500118192 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: BC150EA8-C5DC-4401-83C6-7091874F799F

Device          Start       End   Sectors   Size Type
/dev/sdb1          34    262177    262144   128M Microsoft reserved
/dev/sdb2      264192 498509487 498245296 237.6G Microsoft basic data
/dev/sdb3   498509824 500115455   1605632   784M Windows recovery environment

============================================================
xiang@Xiang:~$ lspci
00:00.0 Host bridge: Intel Corporation Haswell-ULT DRAM Controller (rev 09)
00:02.0 VGA compatible controller: Intel Corporation Haswell-ULT Integrated Graphics Controller (rev 09)
00:03.0 Audio device: Intel Corporation Haswell-ULT HD Audio Controller (rev 09)
00:04.0 Signal processing controller: Intel Corporation Device 0a03 (rev 09)
00:14.0 USB controller: Intel Corporation 8 Series USB xHCI HC (rev 04)
00:16.0 Communication controller: Intel Corporation 8 Series HECI #0 (rev 04)
00:1b.0 Audio device: Intel Corporation 8 Series HD Audio Controller (rev 04)
00:1c.0 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 1 (rev e4)
00:1c.2 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 3 (rev e4)
00:1c.3 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 4 (rev e4)
00:1c.4 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 5 (rev e4)
00:1d.0 USB controller: Intel Corporation 8 Series USB EHCI #1 (rev 04)
00:1f.0 ISA bridge: Intel Corporation 8 Series LPC Controller (rev 04)
00:1f.2 SATA controller: Intel Corporation 8 Series SATA Controller 1 [AHCI mode] (rev 04)
00:1f.3 SMBus: Intel Corporation 8 Series SMBus Controller (rev 04)
00:1f.6 Signal processing controller: Intel Corporation 8 Series Thermal (rev 04)
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 10)
03:00.0 Network controller: Qualcomm Atheros AR9485 Wireless Network Adapter (rev 01)
04:00.0 3D controller: NVIDIA Corporation GM108M [GeForce 840M] (rev a2)

============================================================
xiang@Xiang:~$ lsusb
Bus 001 Device 002: ID 8087:8000 Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 002 Device 006: ID 0bda:0139 Realtek Semiconductor Corp. RTS5139 Card Reader Controller
Bus 002 Device 005: ID 04f2:b3d8 Chicony Electronics Co., Ltd
Bus 002 Device 010: ID 13d3:3402 IMC Networks
Bus 002 Device 003: ID 18f8:0f99
Bus 002 Device 002: ID 1b80:b508 Afatech
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
============================================================
```

Listing F.1: Personal computer technical specifications

F.2 Clusterrocks Server

```
[root@clusterrocks ~]# uname -a
Linux clusterrocks.mmu.edu.my 2.6.32-504.16.2.el6.x86_64 #1 SMP Wed Apr 22 06:48:29 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
[root@clusterrocks ~]#

============================================================
[root@clusterrocks ~]# cat /proc/cpuinfo
processor     : 0
vendor_id     : GenuineIntel
cpu family     : 6
model         : 23
model name     : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping     : 10
microcode     : 2571
cpu MHz         : 2833.519
cache size     : 6144 KB
physical id     : 0
siblings     : 4
core id         : 0
cpu cores     : 4
apicid         : 0
initial apicid     : 0
fpu         : yes
fpu_exception     : yes
cpuid level     : 13
wp         : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
      arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
bogomips     : 5667.03
clflush size     : 64
cache_alignment     : 64
address sizes     : 38 bits physical, 48 bits virtual
power management:

processor     : 1
vendor_id     : GenuineIntel
cpu family     : 6
model         : 23
model name     : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping     : 10
microcode     : 2571
cpu MHz         : 2833.519
cache size     : 6144 KB
physical id     : 1
siblings     : 4
core id         : 0
cpu cores     : 4
apicid         : 4
initial apicid     : 4
fpu         : yes
fpu_exception     : yes
cpuid level     : 13
wp         : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
      arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
```

```
bogomips    : 5665.87
clflush size    : 64
cache_alignment    : 64
address sizes    : 38 bits physical, 48 bits virtual
power management:

processor    : 2
vendor_id    : GenuineIntel
cpu family    : 6
model        : 23
model name    : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping    : 10
microcode    : 2571
cpu MHz        : 2833.519
cache size    : 6144 KB
physical id    : 0
siblings    : 4
core id        : 1
cpu cores    : 4
apicid        : 1
initial apicid    : 1
fpu        : yes
fpu_exception    : yes
cpuid level    : 13
wp        : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
     arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
bogomips    : 5667.03
clflush size    : 64
cache_alignment    : 64
address sizes    : 38 bits physical, 48 bits virtual
power management:

processor    : 3
vendor_id    : GenuineIntel
cpu family    : 6
model        : 23
model name    : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping    : 10
microcode    : 2571
cpu MHz        : 2833.519
cache size    : 6144 KB
physical id    : 1
siblings    : 4
core id        : 1
cpu cores    : 4
apicid        : 5
initial apicid    : 5
fpu        : yes
fpu_exception    : yes
cpuid level    : 13
wp        : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
     arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
bogomips    : 5665.87
clflush size    : 64
cache_alignment    : 64
address sizes    : 38 bits physical, 48 bits virtual
```

```
power management:

processor    : 4
vendor_id    : GenuineIntel
cpu family    : 6
model        : 23
model name    : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping    : 10
microcode    : 2571
cpu MHz        : 2833.519
cache size    : 6144 KB
physical id    : 0
siblings    : 4
core id        : 2
cpu cores    : 4
apicid        : 2
initial apicid    : 2
fpu        : yes
fpu_exception    : yes
cpuid level    : 13
wp        : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
     arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
bogomips    : 5667.03
clflush size    : 64
cache_alignment    : 64
address sizes    : 38 bits physical, 48 bits virtual
power management:

processor    : 5
vendor_id    : GenuineIntel
cpu family    : 6
model        : 23
model name    : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping    : 10
microcode    : 2571
cpu MHz        : 2833.519
cache size    : 6144 KB
physical id    : 1
siblings    : 4
core id        : 2
cpu cores    : 4
apicid        : 6
initial apicid    : 6
fpu        : yes
fpu_exception    : yes
cpuid level    : 13
wp        : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
     arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
bogomips    : 5665.87
clflush size    : 64
cache_alignment    : 64
address sizes    : 38 bits physical, 48 bits virtual
power management:

processor    : 6
vendor_id    : GenuineIntel
```

```
cpu family    : 6
model         : 23
model name    : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping    : 10
microcode    : 2571
cpu MHz         : 2833.519
cache size    : 6144 KB
physical id    : 0
siblings    : 4
core id         : 3
cpu cores    : 4
apicid         : 3
initial apicid    : 3
fpu        : yes
fpu_exception    : yes
cpuid level    : 13
wp         : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
     arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
bogomips    : 5667.03
clflush size    : 64
cache_alignment    : 64
address sizes    : 38 bits physical, 48 bits virtual
power management:

processor    : 7
vendor_id    : GenuineIntel
cpu family    : 6
model         : 23
model name    : Intel(R) Xeon(R) CPU           E5440  @ 2.83GHz
stepping    : 10
microcode    : 2571
cpu MHz         : 2833.519
cache size    : 6144 KB
physical id    : 1
siblings    : 4
core id         : 3
cpu cores    : 4
apicid         : 7
initial apicid    : 7
fpu        : yes
fpu_exception    : yes
cpuid level    : 13
wp         : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc
     arch_perfmon pebs bts rep_good aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dts tpr_shadow vnmi flexpriority
bogomips    : 5665.87
clflush size    : 64
cache_alignment    : 64
address sizes    : 38 bits physical, 48 bits virtual
power management:

============================================================
[root@clusterrocks ~]# cat /proc/meminfo
MemTotal:         6123456 kB
MemFree:           166580 kB
Buffers:           270448 kB
Cached:           3920092 kB
```

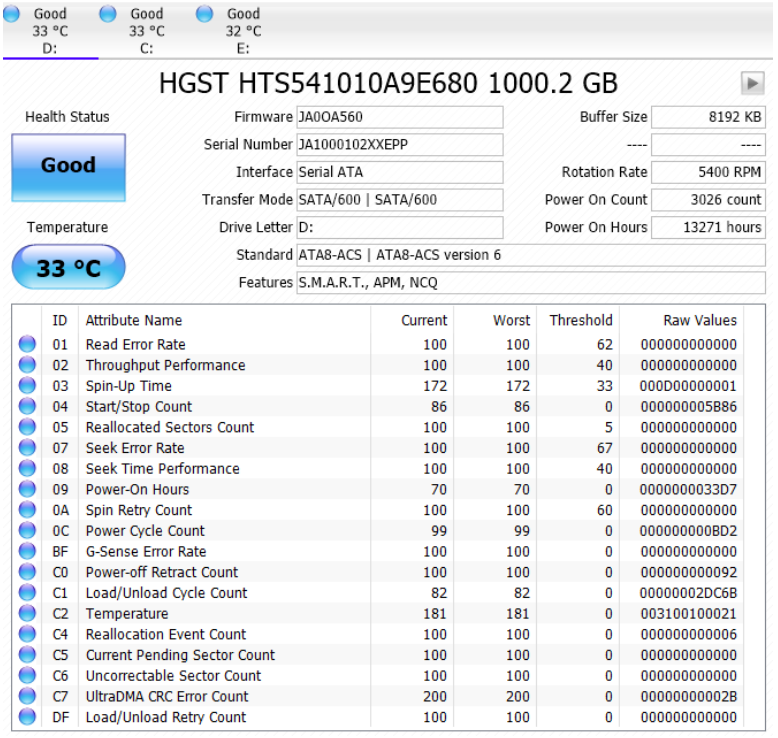
```
SwapCached:          5680 kB
Active:           1369980 kB
Inactive:         4251676 kB
Active(anon):      943660 kB
Inactive(anon):    516476 kB
Active(file):      426320 kB
Inactive(file):   3735200 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:       25599992 kB
SwapFree:        25557328 kB
Dirty:             204480 kB
Writeback:              0 kB
AnonPages:        1426436 kB
Mapped:             22728 kB
Shmem:              29380 kB
Slab:              202088 kB
SReclaimable:      162888 kB
SUnreclaim:         39200 kB
KernelStack:         5440 kB
PageTables:         21544 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:     28661720 kB
Committed_AS:     4408512 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       101692 kB
VmallocChunk:   34359630672 kB
HardwareCorrupted:      0 kB
AnonHugePages:    1239040 kB
HugePages_Total:        0
HugePages_Free:         0
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        2048 kB
DirectMap4k:         7104 kB
DirectMap2M:      6283264 kB

============================================================
HARD DISK
============================================================
[root@clusterrocks ~]# fdisk -l

Disk /dev/sda: 251.1 GB, 251059544064 bytes
255 heads, 63 sectors/track, 30522 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x5ed3b633

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1        2040    16384000   83  Linux
/dev/sda2            2040        2550     4096000   83  Linux
/dev/sda3            2550        2678     1024000   82  Linux swap / Solaris
/dev/sda4            2678       30523   223670312    5  Extended
/dev/sda5            2678       30523   223669248   83  Linux
```

```
Disk /dev/sdb: 251.1 GB, 251059544064 bytes
255 heads, 63 sectors/track, 30522 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x5ed3b633

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1       30523   245174272   fd  Linux raid autodetect

Disk /dev/sdc: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0x00052bb4

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1        3060    24576000   82  Linux swap / Solaris
/dev/sdc2   *        3060      121602   952184832   83  Linux

Disk /dev/md127: 251.1 GB, 251057266688 bytes
2 heads, 4 sectors/track, 61293278 cylinders
Units = cylinders of 8 * 512 = 4096 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

============================================================
[root@clusterrocks ~]# lspci
00:00.0 Host bridge: Intel Corporation 5400 Chipset Memory Controller Hub (rev 20)
00:01.0 PCI bridge: Intel Corporation 5400 Chipset PCI Express Port 1 (rev 20)
00:05.0 PCI bridge: Intel Corporation 5400 Chipset PCI Express Port 5 (rev 20)
00:09.0 PCI bridge: Intel Corporation 5400 Chipset PCI Express Port 9 (rev 20)
00:0f.0 System peripheral: Intel Corporation 5400 Chipset QuickData Technology Device (rev 20)
00:10.0 Host bridge: Intel Corporation 5400 Chipset FSB Registers (rev 20)
00:10.1 Host bridge: Intel Corporation 5400 Chipset FSB Registers (rev 20)
00:10.2 Host bridge: Intel Corporation 5400 Chipset FSB Registers (rev 20)
00:10.3 Host bridge: Intel Corporation 5400 Chipset FSB Registers (rev 20)
00:10.4 Host bridge: Intel Corporation 5400 Chipset FSB Registers (rev 20)
00:11.0 Host bridge: Intel Corporation 5400 Chipset CE/SF Registers (rev 20)
00:15.0 Host bridge: Intel Corporation 5400 Chipset FBD Registers (rev 20)
00:15.1 Host bridge: Intel Corporation 5400 Chipset FBD Registers (rev 20)
00:16.0 Host bridge: Intel Corporation 5400 Chipset FBD Registers (rev 20)
00:16.1 Host bridge: Intel Corporation 5400 Chipset FBD Registers (rev 20)
00:1b.0 Audio device: Intel Corporation 631xESB/632xESB High Definition Audio Controller (rev 09)
00:1c.0 PCI bridge: Intel Corporation 631xESB/632xESB/3100 Chipset PCI Express Root Port 1 (rev 09)
00:1d.0 USB controller: Intel Corporation 631xESB/632xESB/3100 Chipset UHCI USB Controller #1 (rev 09)
00:1d.1 USB controller: Intel Corporation 631xESB/632xESB/3100 Chipset UHCI USB Controller #2 (rev 09)
00:1d.2 USB controller: Intel Corporation 631xESB/632xESB/3100 Chipset UHCI USB Controller #3 (rev 09)
00:1d.3 USB controller: Intel Corporation 631xESB/632xESB/3100 Chipset UHCI USB Controller #4 (rev 09)
00:1d.7 USB controller: Intel Corporation 631xESB/632xESB/3100 Chipset EHCI USB2 Controller (rev 09)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev d9)
00:1f.0 ISA bridge: Intel Corporation 631xESB/632xESB/3100 Chipset LPC Interface Controller (rev 09)
00:1f.1 IDE interface: Intel Corporation 631xESB/632xESB IDE Controller (rev 09)
00:1f.2 IDE interface: Intel Corporation 631xESB/632xESB/3100 Chipset SATA IDE Controller (rev 09)
00:1f.3 SMBus: Intel Corporation 631xESB/632xESB/3100 Chipset SMBus Controller (rev 09)
02:00.0 VGA compatible controller: NVIDIA Corporation G92GL [Quadro FX 3700] (rev a2)
```

```
03:00.0 PCI bridge: Intel Corporation 6311ESB/6321ESB PCI Express Upstream Port (rev 01)
03:00.3 PCI bridge: Intel Corporation 6311ESB/6321ESB PCI Express to PCI-X Bridge (rev 01)
04:00.0 PCI bridge: Intel Corporation 6311ESB/6321ESB PCI Express Downstream Port E1 (rev 01)
07:00.0 Ethernet controller: Intel Corporation 82575EB Gigabit Network Connection (rev 02)
07:00.1 Ethernet controller: Intel Corporation 82575EB Gigabit Network Connection (rev 02)
08:08.0 FireWire (IEEE 1394): Texas Instruments TSB43AB22A IEEE-1394a-2000 Controller (PHY/Link) [iOHCI-Lynx]

============================================================
[root@clusterrocks ~]# lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
==================
```

Listing F.2: Clusterrocks server technical specifications

F.3 1 TB HGST HTS541010A9E680 HDD



ID	Attribute Name	Current	Worst	Threshold	Raw Values
01	Read Error Rate	100	100	62	000000000000
02	Throughput Performance	100	100	40	000000000000
03	Spin-Up Time	172	172	33	000D00000001
04	Start/Stop Count	86	86	0	000000005B86
05	Reallocated Sectors Count	100	100	5	000000000000
07	Seek Error Rate	100	100	67	000000000000
08	Seek Time Performance	100	100	40	000000000000
09	Power-On Hours	70	70	0	0000000033D7
0A	Spin Retry Count	100	100	60	000000000000
0C	Power Cycle Count	99	99	0	000000000BD2
BF	G-Sense Error Rate	100	100	0	000000000000
C0	Power-off Retract Count	100	100	0	000000000092
C1	Load/Unload Cycle Count	82	82	0	00000002DC6B
C2	Temperature	181	181	0	003100100021
C4	Reallocation Event Count	100	100	0	000000000006
C5	Current Pending Sector Count	100	100	0	000000000000
C6	Uncorrectable Sector Count	100	100	0	000000000000
C7	UltraDMA CRC Error Count	200	200	0	00000000002B
DF	Load/Unload Retry Count	100	100	0	000000000000

FIGURE F.1: 1 TB HGST HTS541010A9E680 HDD technical specifications

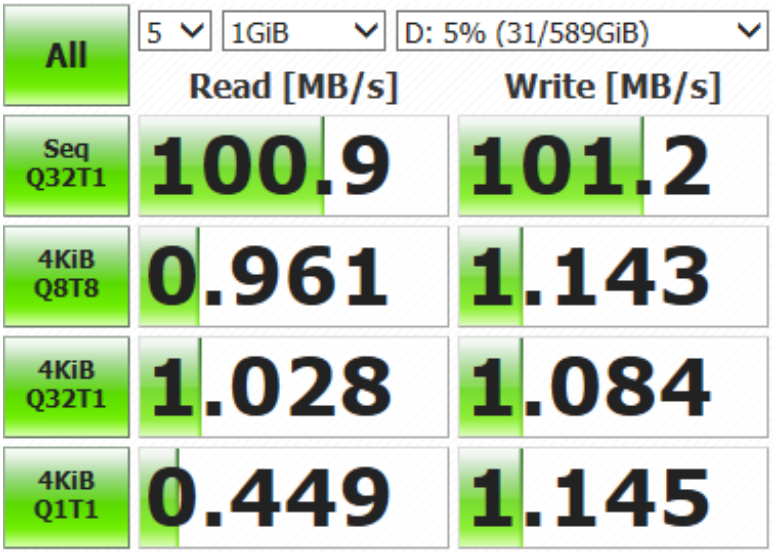
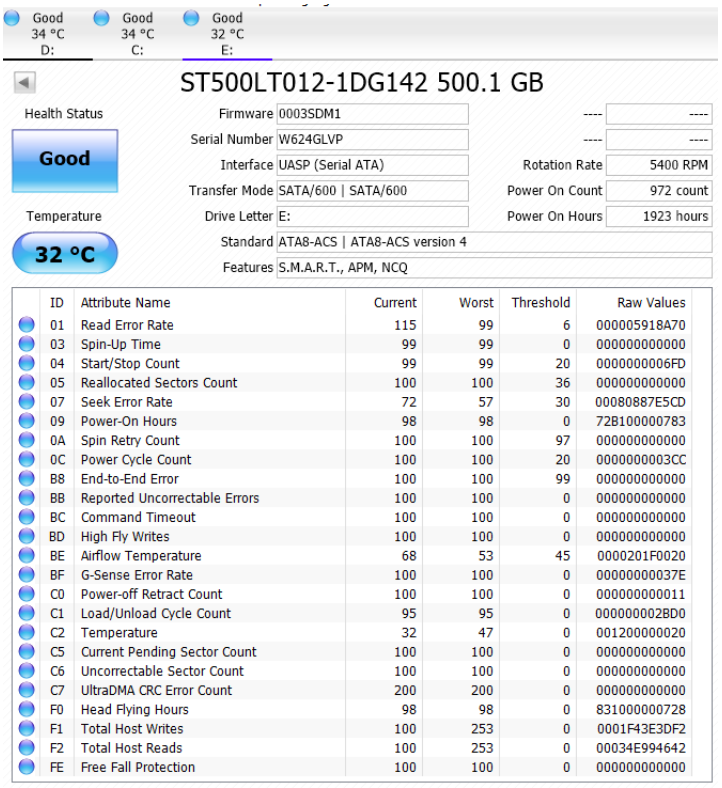


FIGURE F.2: 1 TB HGST HTS541010A9E680 HDD Benchmark

F.4 500 GB ATA ST500LT012-1DG14 HDD



ID	Attribute Name	Current	Worst	Threshold	Raw Values
01	Read Error Rate	115	99	6	000005918A70
03	Spin-Up Time	99	99	0	000000000000
04	Start/Stop Count	99	99	20	0000000006FD
05	Reallocated Sectors Count	100	100	36	000000000000
07	Seek Error Rate	72	57	30	00080887E5CD
09	Power-On Hours	98	98	0	72B100000783
0A	Spin Retry Count	100	100	97	000000000000
0C	Power Cycle Count	100	100	20	0000000003CC
B8	End-to-End Error	100	100	99	000000000000
BB	Reported Uncorrectable Errors	100	100	0	000000000000
BC	Command Timeout	100	100	0	000000000000
BD	High Fly Writes	100	100	0	000000000000
BE	Airflow Temperature	68	53	45	0000201F0020
BF	G-Sense Error Rate	100	100	0	00000000037E
C0	Power-off Retract Count	100	100	0	000000000011
C1	Load/Unload Cycle Count	95	95	0	000000002BD0
C2	Temperature	32	47	0	001200000020
C5	Current Pending Sector Count	100	100	0	000000000000
C6	Uncorrectable Sector Count	100	100	0	000000000000
C7	UltraDMA CRC Error Count	200	200	0	000000000000
F0	Head Flying Hours	98	98	0	831000000728
F1	Total Host Writes	100	253	0	0001F43E3DF2
F2	Total Host Reads	100	253	0	00034E994642
FE	Free Fall Protection	100	100	0	000000000000

FIGURE F.3: 500 GB ATA ST500LT012-1DG14 HDD technical specifications

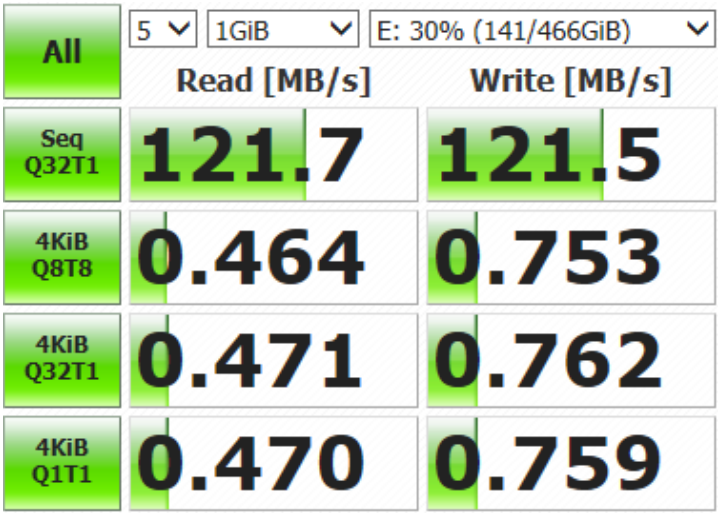


FIGURE F.4: 500 GB ATA ST500LT012-1DG14 HDD Benchmark

Appendix G

Mapping Clusterrocks File System to Local Machine

Step 1 - Install Cisco Anyconnect Secure Mobility Client.

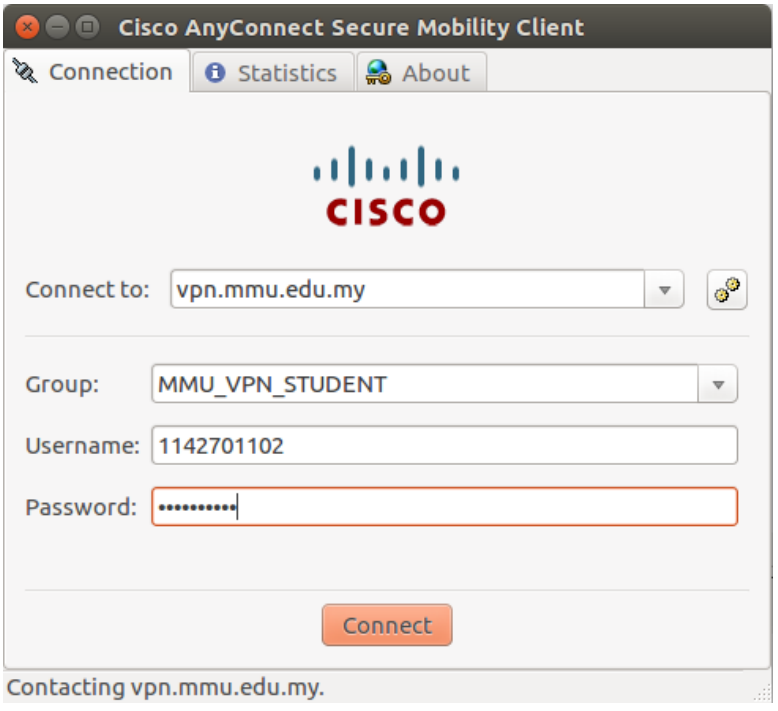


FIGURE G.1: Connecting to MMU VPN

Step 2 - Connect to MMU VPN using program we installed in Step 1.

```
xiang@Xiang:~$ sudo mkdir /mnt/xmldata
[sudo] password for xiang:
xiang@Xiang:~$ sudo mkdir /mnt/xmldata/stov
xiang@Xiang:~$

xiang@Xiang:~$ sudo sshfs -o allow_other xmldata@10.106.50.214:/ /mnt/xmldata/stov
#---Password Here---#

xiang@Xiang:/mnt/xmldata/stov$ cd /mnt/xmldata/stov/oneTBHD/export/home/xmldata/stov

xiang@Xiang:/mnt/xmldata/stov/oneTBHD/export/home/xmldata/stov$ ls -al
total 168325072
drwxrwxr-x 1 513 513         4096 Sep 20 18:05 .
drwx------ 1 513 513         4096 Sep 20 18:07 ..
-rw-r--r-- 1 513 513   2699001898 Jun 13  2017 Badges.xml
-rw-r--r-- 1 513 513 15297145999 Jun 13  2017 Comments.xml
-rw-rw-r-- 1 513 513 82818920448 Jul 22 03:49 PostHistory.xml
-rw-rw-r-- 1 513 513    516341879 Jun 13  2017 PostLinks.xml
-rw-rw-r-- 1 513 513 56228456649 Jun 13  2017 Posts.xml
-rw-rw-r-- 1 513 513      4389345 Jun 13  2017 Tags.xml
-rw-rw-r-- 1 513 513   2231285941 Jun 13  2017 Users.xml
-rw-rw-r-- 1 513 513 12569307763 Jun 13  2017 Votes.xml
```

Listing G.1: Instructions to map clusterrocks directory to local machine

Step 3 - Follow instruction shown in Listing G.1.

Step 4 - If files in clusterrocks's directories are listed, we have successfully mapped clusterrocks file system to our local machine.

Appendix H

Keyword Search Engine

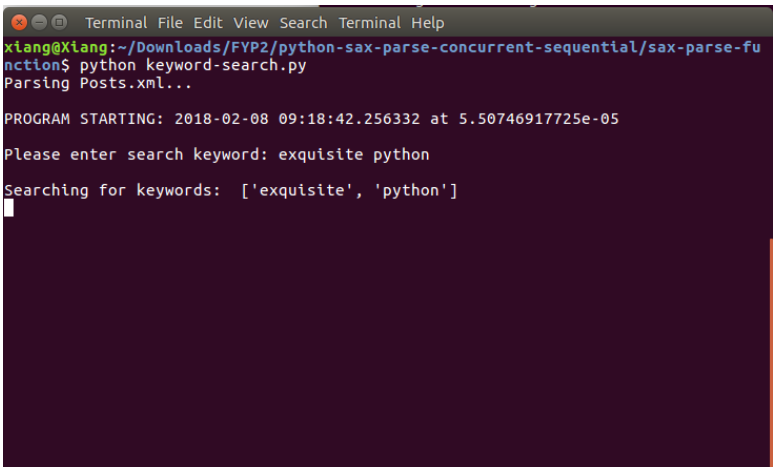


FIGURE H.1: Input search keywords

Step 1 - Input keywords we wanted to search for. In Figure H.1, we used an example "exquisite python".

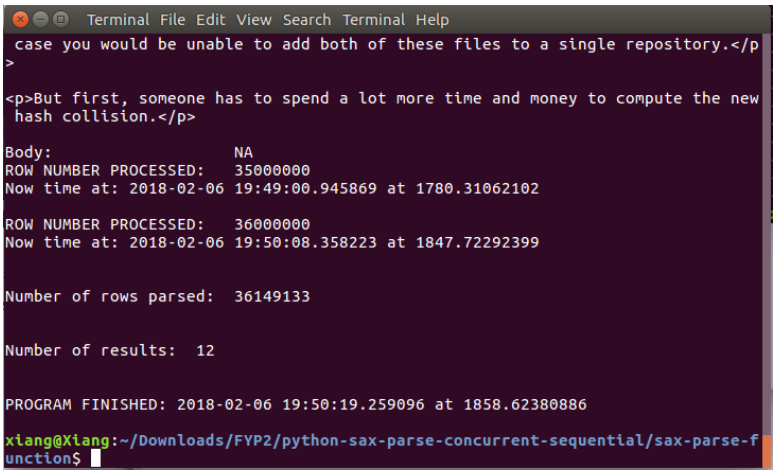


FIGURE H.2: Program output

Step 2 - If matching results are found, it will be displayed immediately and saved to file at the same time. Data accounting are also implemented to verify the XML has been scanned thoroughly.

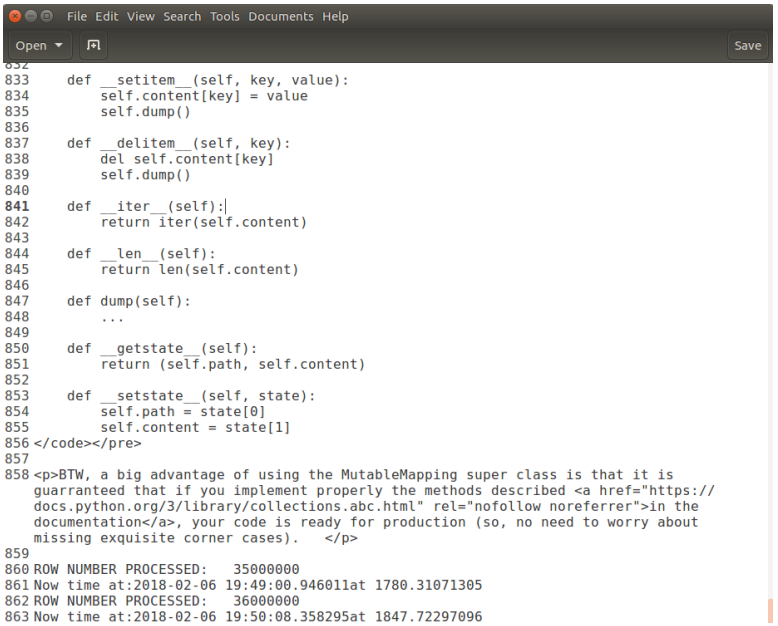


FIGURE H.3: Saved results

Step 3 - The relevant results are saved in a text file which could be access offline at anytime.

Appendix I

System Monitor during Remote SAX Parse

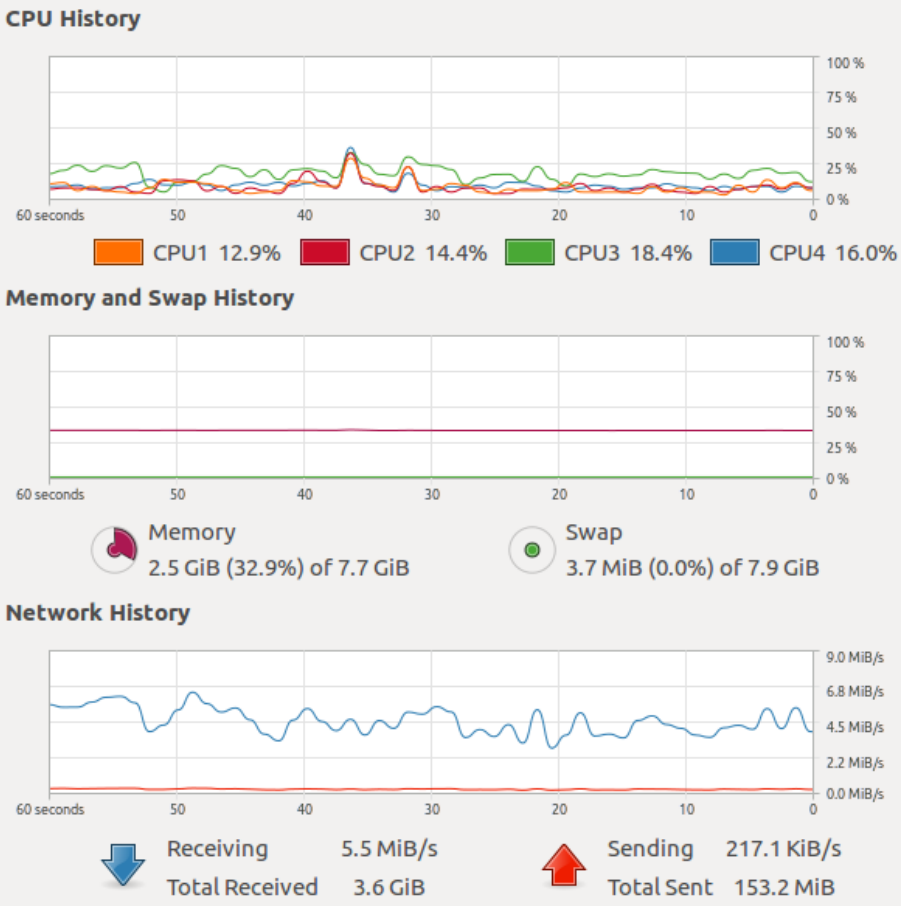


FIGURE I.1: System Monitor during remote SAX parse

Appendix J

Meeting Logs

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 4/7/2017	**MEETING NO.: Week 1**
PROJECT ID: 654	
PROJECT TITLE :Utilizing Parallel Programming to Speed Up Data Processing Time in R-Language	
SESSION : 1	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Meet up with my supervisor Wan Ruslan Yusoff to discuss about the project.
(2) Consider to use R programming language to demonstrate parallel programming capabilities in data processing.

2. WORK TO BE DONE
(1) Read about parallel programming.
(2) Learn how to use R programming language.
(3) Set up Ubuntu on my machine.

3. PROBLEMS ENCOUNTERED

(1) Finding datasets big enough to demonstrate the usefulness of parallel programming.
(2) Not using Linux operating system.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…… ……..

Supervisor's Signature Student's Signature

……..

Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 11/7/2017	**MEETING NO.: Week 2**
PROJECT ID: 654	
PROJECT TITLE :Utilizing Parallel Programming to Speed Up Data Processing Time in R-Language	
SESSION : 2	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Inform my supervisor Wan Ruslan Yusoff to write to Dr. Yeoh to change the specialization from Software Engineering to Data Science so I could register the FYP Title.

(2) Inform Dr. Yeoh that I already changed specialization to Data Science on Trimester 2, 2016/2017.

(3) Set up Ubuntu partition on my laptop.

(4) Installed R programming language and its IDE Rstudio.

(5) Read guides and tutorial on R programming language.

(6) Understand the concept of parallel programming.

2. WORK TO BE DONE

(1) Upgrade Rstudio and R to latest version

(2) Read about concurrent programming

(3) Practice parallel programming in R

3. PROBLEMS ENCOUNTERED

(1) Supervisor is unable to register the project under my name because the project's specialization is software engineering.

(2) Confusion about difference between concurrent and parallel programming.

(3) Initial allocated space too small for Ubuntu partition, when resizing the partition I just realized I cannot do that while mounting Ubuntu partition. And I had to use Gparted so I cannot log onto Windows and resize it using Disk fragmentation. I had to learn how to use liveUSB and resize the partition using Gparted on it.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…… ……..

Supervisor's Signature Student's Signature

……..

Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 18/7/2017	**MEETING NO.: Week 3**
PROJECT ID: 654	
PROJECT TITLE :Utilizing Parallel Programming to Speed Up Data Processing Time in R-Language	
SESSION : 3	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Upgraded RStudio and R to latest version

(2) Read tutorial and tried out parallel programming in R

(3) Read about environments in R as it is closely related to concept of lexical scoping and it is a big deal if I wanted to do parallel programming in R

(4) Read about concurrent programming and know its difference from parallel programming

(5) Installed TeXStudio on my machine

(6) Read about Golang, a programming language good for concurrent programming

2. WORK TO BE DONE
(1) Downloads and extract Stackoverflow datasets (185GB)
(2) Read about Simple API for XML (SAX) and Document Object Model (DOM)
(3) Start writing drafts for FYP Report

3. PROBLEMS ENCOUNTERED

(1) Encountered some error like package is missing while installing TeXStudio

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…… ……..

Supervisor's Signature Student's Signature

……..

Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 25/7/2017	**MEETING NO.: Week 4**
PROJECT ID: 654	
PROJECT TITLE :Utilizing Parallel Programming to Speed Up Data Processing Time in R-Language	
SESSION : 4	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Downloaded and extracted Stackoverflow datasets (185GB, XML format)

(2) Read about Simple API for XML(SAX) and Document Object Model (DOM), both are XML parser.

(3) Installed Synaptic Package Manager that enable me to handle packages in Ubuntu comfortably, notable features is it is point-and-click and enable you to browse and search for list of available packages with ease.

2. WORK TO BE DONE
(1) Write executive summary, introduction and literature review for FYP report.
(2) Learn how to use Simple API for XML (SAX) parser in R language.

3. PROBLEMS ENCOUNTERED

(1) Filesize is too huge (35.6GB compressed), it takes a lot of time to download the files.
(2) Uncompressing the files takes too long (185GB uncompressed), it took me whole night to uncompress them.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…… ……..

Supervisor's Signature Student's Signature

……..

Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 2/8/2017	**MEETING NO.: Week 5**
PROJECT ID: 654	
PROJECT TITLE : Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 5	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Read and understand concepts closely related to parallel programming such as mutual exclusion, deadlock, livelock, starvation and etc.
(2) Learned some basics on SAX parser in R-language.
(3) Written management summary, introduction and literature review for draft report.
(4) Propose to change title to “Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons”

2. WORK TO BE DONE
(1) Download and install Eclipse-Parallel Oxygen.
(2) Setup Python3.5 with PyDev GUI in Eclipse-Parallel Oxygen.
(3) Download and install MongoDB Community Server.
(4) Learn XPath query language.

3. PROBLEMS ENCOUNTERED
(1) -ERROR: configuration failed for package ‘XML’.
(2) Difficulty in using XML-package on R because of inexperience.
(3) Datasets used in most online tutorials for XML is formatted differently from the project’s dataset (self closing tag).
(4) XPath query language is needed for good manipulation of XML documents.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…… ……..

Supervisor's Signature Student's Signature

……..

Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 9/8/2017	**MEETING NO.: Week 6**
PROJECT ID: 654	
PROJECT TITLE : Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 6	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE

[Please write the details of the work done after the last meeting.]

(1) Setup Eclipse-Parallel Oxygen on my machine.
(2) Setup PyDev in Eclipse-Parallel Oxygen.
(3) Setup separate Eclipse-Parallel Oxygen IDE for XML and JSON.
(4) Setup MongoDB Community Server on my machine.
(5) Setup LTTng (Linux Trace Toolkit Next Generation) on my machine.
(6) Setup babeltrace on my machine.
(7) Setup GNU Debugger on my machine.
(8) Learn and understand about JSON.
(9) Parsing JSON and XML files on both R and Python.

2. WORK TO BE DONE

(1) Write second draft report for checking by supervisor.
(2) Install Trace Compass 3.0 on machine.
(3) Try to run Rscript in Python and vice versa.
(4) Tracing performance of program.

3. PROBLEMS ENCOUNTERED

(1) Codes run for Python 2 but not Python 3.
(2) GNU Debugger install failed because no texinfo package.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…… ……...

Supervisor's Signature Student's Signature

……..

Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 16/8/2017	**MEETING NO.: Week 7**
PROJECT ID: 654	
PROJECT TITLE : Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 7	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Written improvised draft report for supervisor to check.
(2) Installed Trace Compass 3.0 on my machine.
(3) Understand BabelTrace, a reference implementation of Linux trace format.
(4) Understand and note down my Critical Success Factor (CSF).

2. WORK TO BE DONE
(1) Read “MongoDB : The Definitive Guide”.
(2) Setup MongoDB on my local machine.
(3) Uploading sample JSON file to MongoDB.
(4) Improvise draft report based on previous draft report.

3. PROBLEMS ENCOUNTERED
(1) tracklang.sty for datetime2 is missing, need to install texlive-generic-extra for texstudio

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…… ……..

Supervisor's Signature Student's Signature

……..

Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 23/8/2017	**MEETING NO.: Week 8**
PROJECT ID: 654	
PROJECT TITLE : Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 8	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Understand the fundamentals of MongoDB.
(2) Setup MonngoDB on my local machine.
(3) Uploading sample JSON file to MongoDB for testing.

2. WORK TO BE DONE
(1) Write serial program for data processing in Python.
(2) Convert XML to JSON in Python.

3. PROBLEMS ENCOUNTERED
(1) batchInsert on MongoDB does not work anymore, use insert instead for multiple insertion in one go.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

……
Supervisor's Signature

……..
Student's Signature

……..
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 30/8/2017	**MEETING NO.: Week 9**
PROJECT ID: 654	
PROJECT TITLE : Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 9	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Written serial program for data processing in Python.
(2) Written a script to convert XML to JSON in Python.

2. WORK TO BE DONE
(1) Write parallel program for data processing in Python.
(2) Upload converted JSON to MongoDB using Python.

3. PROBLEMS ENCOUNTERED
(1) iterParse() will make memory full even if element is cleared while parsing, must remove the element from the tree to prevent memory full.
(2) Cannot parse XML file using xmltodict() if file is inserted using argparse(), have to specify file type to be 'rb' beforehand.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

…….
Supervisor's Signature

……..
Student's Signature

……..
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 6/9/2017	**MEETING NO.: Week 10**
PROJECT ID: 654	
PROJECT TITLE : Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 10	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Written parallel program for data processing in Python.
(2) Upload converted JSON to MongoDB using Python.

2. WORK TO BE DONE
(1) Compare execution time between serial and parallel program in data processing.
(2) Writeup draft report to be checked by supervisor.

3. PROBLEMS ENCOUNTERED
(1) Failed to install mpi4py and threading in Python because openmpi is missing.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

……
Supervisor's Signature

……..
Student's Signature

……..
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 13/9/2017	**MEETING NO.: Week 11**
PROJECT ID: 654	
PROJECT TITLE : Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 11	**SUPERVISOR : Wan Ruslan Yusoff**
STUDENT ID & Name: 1142701102 Wong Zi Xiang	**CO- SUPERVISOR :**

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Compared execution time between serial and parallel program in data processing.
(2) Finished draft report to be checked by supervisor

2. WORK TO BE DONE
(1) Finalize interim report and submit to faculty.

3. PROBLEMS ENCOUNTERED
(1) Parallel program is slower than serial program, have to do many research to find out the root problem (Manager dict is expensive, Process must use Manager for shared list) and fix it(Use Pool instead).

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

……
Supervisor's Signature

……..
Student's Signature

……..
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 22/11/2017	MEETING NO.: Week 1
PROJECT ID: 654	
PROJECT TITLE : **Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons**	
SESSION : 1	SUPERVISOR : Wan Ruslan Yusoff
STUDENT ID & Name: 1142701102 Wong Zi Xiang	CO- SUPERVISOR :

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Meet up with my supervisor to discuss about project scope in Phase 2 and is any changes needed to be made.
(2) Setup account for me to access clusterrocks server.
(3) Tested the access to clusterrocks server.
(4) Cloned project data to clusterrocks server.

2. WORK TO BE DONE
(1) Install VPN at home.
(2) Try to test connection between home and clusterrocks server (MMU)

3. PROBLEMS ENCOUNTERED
4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

.....
Supervisor's Signature

.....
Student's Signature

.....
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 29/11/2017	MEETING NO.: Week 2
PROJECT ID: 654	
PROJECT TITLE :Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 2	SUPERVISOR : Wan Ruslan Yusoff
STUDENT ID & Name: 1142701102 Wong Zi Xiang	CO- SUPERVISOR :

1. WORK DONE
[Please write the details of the work done after the last meeting.]

(1) Installed VPN on my machine.
(2) Tested connection from home to clusterrocks server, all is well.

2. WORK TO BE DONE
(1) SAX parse project data on local machine and record the output.
(2) Try to SAX parse project data on clusterrocks server from home using VPN.

3. PROBLEMS ENCOUNTERED
(1) When installing VPN on Ubuntu, errors occurred where dependencies are missing.
4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia

.....
Supervisor's Signature

.....
Student's Signature

.....
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 13/12/2017	MEETING NO.: Week 4
PROJECT ID: 654	
PROJECT TITLE :Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 34	SUPERVISOR : Wan Ruslan Yusoff
STUDENT ID & Name: 1142701102 Wong Zi Xiang	CO- SUPERVISOR :

1. WORK DONE

[Please write the details of the work done after the last meeting.]

(1) Successfully SAX parsed project data on local machine and clusterrocks server.
(2) Performance of SAX parse is recorded for both local machine and clusterrocks server.

2. WORK TO BE DONE

(1) Plot graphs to compare performance of SAX parse between on local and remote machine.
(2) Create useful program for SAX parser such as querying data. (Search function)
(3) Clone the working PostHistory.XML into clusterrocks server

3. PROBLEMS ENCOUNTERED

(1) Error "XML not well-formed" encountered when parsing PostHistory.XML on clusterrocks server, the file seems to be corrupted.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

.....
Supervisor's Signature

.....
Co-Supervisor's Signature

.....
Student's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 27/12/2017	MEETING NO.: Week 6
PROJECT ID: 654	
PROJECT TITLE :Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 4	SUPERVISOR : Wan Ruslan Yusoff
STUDENT ID & Name: 1142701102 Wong Zi Xiang	CO- SUPERVISOR :

1. WORK DONE

[Please write the details of the work done after the last meeting.]

(1) Plotted performance comparison graphs for parallel and sequential SAX parse of project data on local machine VS clusterrocks server.
(2) Written program which can search for a keyword in Posts.XML
(3) Cloned PostHistory.XML from my machine to clusterrocks server.
(4) Recorded technical specification of clusterrocks server.

2. WORK TO BE DONE

(1) Reconsider that whether MongoDB is really suitable for this project or not.
(2) Prove that it is impossible to DOM parse huge XMLs.

3. PROBLEMS ENCOUNTERED
4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

.....
Supervisor's Signature

.....
Student's Signature

.....
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 10/1/2017	MEETING NO.: Week 8
PROJECT ID: 654	
PROJECT TITLE :Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 5	SUPERVISOR : Wan Ruslan Yusoff
STUDENT ID & Name: 1142701102 Wong Zi Xiang	CO- SUPERVISOR :

1. WORK DONE

[Please write the details of the work done after the last meeting.]

(1) Written program to DOM parse project data (PostLinks.XML 516 MB).
(2) Written program to SAX parse project data.
(3) Recorded the memory consumption of DOM and SAX.

2. WORK TO BE DONE

(1) Find out whether hard disk bottlenecked parallel SAX.
(2) Write same program for SAX parsing on R.
(3) Compare performance between Python and R.

3. PROBLEMS ENCOUNTERED

(1) Parallel SAX is not faster than sequential SAX, might be because of hard disk limitation.

4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatic .
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

.....
Supervisor's Signature

.....
Student's Signature

.....
Co-Supervisor's Signature

MULTIMEDIA UNIVERSITY

Faculty of Information Technology
Final Year Project Meeting Log

MEETING DATE: 24/1/2017	MEETING NO.: Week 10
PROJECT ID: 654	
PROJECT TITLE :Parallel and Distributed processing with R and Python programming languages on Big Data - Performance Comparisons	
SESSION : 6	SUPERVISOR : Wan Ruslan Yusoff
STUDENT ID & Name: 1142701102 Wong Zi Xiang	CO- SUPERVISOR :

1. WORK DONE

[Please write the details of the work done after the last meeting.]

(1) Researched about hard disk drive and confirmed that single hard disk will limit read speed of multiple files.
(2) Gotten new hard disk to perform parallel SAX parse again.
(3) Record performance of parallel SAX parse on dual hard disk.
(4) Plotted performance comparison graphs for sequential, concurrent and parallel SAX parse.
(5) Compared SAX performance between R and Python.

2. WORK TO BE DONE

(1) Write draft report to be proofread by supervisor

3. PROBLEMS ENCOUNTERED
4. COMMENTS

WAN RUSLAN YUSOFF
Lecturer
Faculty of Computing and Informatics
Multimedia University
Persiaran Multimedia, 63100 Cyberjaya
Selangor Darul Ehsan, Malaysia.

.....
Supervisor's Signature

.....
Student's Signature

.....
Co-Supervisor's Signature