1. **INTRODUCTION**

**1.1 PROJECT OBJECTIVE & SCOPE**

Processing large volumes of data, often called big data analytics, has been one of the most important tasks that most corporations need, established enterprises and start-up companies alike. As examples, corporations need to analyse logs from customer activities, make recommendations based on histories of user browsing or purchases, and deliver advertisements to those that may be most interested in them. In the era of big data analytics, the volume of data to be processed grows exponentially, and the need for processing such volumes of data becomes more pressing.

Modern data centres are deployed around the world, in a geographically distributed fashion, to process large volumes of data in a distributed manner using data parallel frameworks, such as Apache Hadoop and Spark. Traditionally, these data parallel frameworks are designed to process data within the same data centre, where jobs typically run within the same cluster, and the data to be processed is locally stored in the Hadoop Distributed File System (HDFS).

However, as the volume of data grows, storing such data within the same data centre is no longer feasible, and they naturally need to be distributed across multiple data centres. This is further motivated by the fact that the data to be processed, such as user activity logs, are generated in a geographically distributed fashion. It is more efficient to store the data where they are generated, perhaps in Apache Hive, a data warehouse infrastructure designed to query and analyses data in distributed storage. Since data to be processed are increasingly stored across multiple data centres around the world, existing data parallel frameworks that are designed to work well in a local cluster, such as Apache Hadoop and Spark, no longer meet the pressing need for big data analytics across multiple data centres. In the literature, the problem of processing data across multiple data centres is often referred to as wide-area data analytics.

The naive solution to process data across multiple data centres is to first migrate all the data to one data centre, and then process them locally, as illustrated in Fig. 1.1 Naturally, the volume of data to be processed, in the order or terabytes, makes it costly and inefficient to perform such wide-area network transfers. First, such an approach consumes a significant amount of network bandwidth [1], which incurs a high monetary cost. Even if the corporation has no budgetary concerns, the capacity of the inter-data centre wide-area network is not increasing at the same rate as the volume of data to be analysed [2], and such a solution is not going to be sustainable over the long run. Finally, migrating all the data to one data centre takes time, and the longer it takes, the worse the performance.

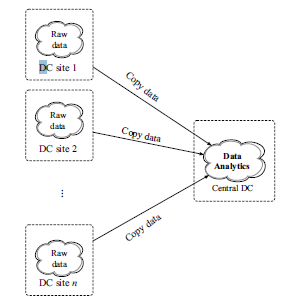


Figure1.1: Migrating all the data to one data centre: A Naive solution for wide-area data analytics across geo-distributed data centres.

**1.2 PROBLEM STATEMENT**

The problem of wide-area data analytics has been widely acknowledged in the recent literature, and a number of solutions have been proposed. In this paper, we will focus on several representative solutions in the literature towards this research direction. Due to the pressing need of processing large volumes of data across multiple geo-distributed data centres, these proposed solutions are exciting and highly relevant, and may soon be utilized in real-world data analytic applications. We begin with a brief introduction of the background of batch and streaming processing frameworks. With several examples, we will then proceed to present the basic ideas at a high level of these proposed solutions, and compare them when the need arises. We will also analyse these solutions, and point out their limitations and disadvantages, and provide our insights towards future work.

**1.3 PROPOSED SYSTEM**

To use the geographically distributed data, the data are been processed using the streaming process. One data element or a small size of data in the stream at a time and the data are processed immediately upon arrival. Then the batch processing framework done through, two key information about Map Reduce jobs we need to know in advance. First, the numbers of map tasks and reduce tasks for each Map Reduce job should be provided. For map tasks, we can deduce its number based on the sizes of input data and data block. For reduce tasks, we can get its number based on a setting guidance. Second, we also need to get the execution time for map and reduce tasks. One way is based on historical information for those jobs that run periodically. Another feasible method is performing some proﬁling run for Map Reduce jobs with small-size input data ﬁrst, to obtain the average execution time for map and reduce tasks

**1.4 LITERATURE SURVEY**

**1.4.1 Big Data**

In last year’s organizations are generating huge amounts of information about their customers. Usually to be successful in their branch of activity, there is often a necessity of storing, as much useful information as is possible, for further analysis.

Financial institutions store much more information about their clients, than usually people expect. They observe records about various clients’ accounts, such as income, payments, periodically balance state and many other. All these records are processed to create a vision of client’s expected behaviour. It seemed to be very useful information for financial institutions like banks and insurance companies. After they are able to obtain client’s expected behaviour, they can decrease risk in making decisions, about providing or not providing different kind of services (such as loans) to the client.

Social networks are nowadays getting more and more popular. People upload their photos and videos, update their statuses, and message their friends. It is an example of case where the problem is not related to the processing of huge data sets of log files, but to the storage resources.

Naturally the general aim, why companies in all these sectors, are storing and analysing data is to make more profit. Many of these greatest organizations, which obtain a lot of records about people and their activities sometimes even, cooperate with security agencies, which collect and analyse data about people for safety reasons.

And now the term, Big Data, it represents all those previously mentioned large data sets, which seems become a key basis of competition, underpinning new waves of productivity growth and innovation.

**1.4.2 Hadoop Framework**

The Apache Hadoop software library is free open source framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. Mostly, Hadoop term is known for programming model Map Reduce and for its distributed file system – HDFS, but in fact the name is used for several Apache projects related to distributed computing and data processing.

Previously was mentioned, that distributed data processing leads to several complexities. Hadoop manages with all those mentioned problems of distributed data storage systems. That’s why it becomes to be popular and recently often used. For people who never heard about Hadoop, it can be described with three main properties which make it so useful.

**Concurrency** — everything works in parallel. Data are splitted into partitions and stored in distributed file system. Later computing processes over the data are being executed simultaneously on different machines.

**Reliability** — Hadoop is a reliable system, that takes care of data replication (with a suitable replication count) and in case of some failure it ensures that data are not lost.

**Scalability** — whenever there is a need to handle larger data, Hadoop scales linearly, just by adding another node to the cluster.

In general Hadoop is the thing that allows users to use complex distributed system simply, as locally working system. Software developers don’t have to care about whole complexity of distributed system and can easily focus on the aims of their work.

As the distributed computing is getting popular, there are appearing different projects, not necessarily under Apache, providing complementary services to Hadoop. Currently Hadoop base consists of four sub-projects, providing the base functionality of the system.

**Hadoop Common** — A middle-ware, set of the common utilities and libraries that support the other Hadoop modules.

**Hadoop Distributed File System (HDFS)** – Distributed file system, designed for storing very large data sets by scaling out across a cluster of hosts. Files are splitted into so called block of given size. It achieves high availability through replication.

**Map Reduce** — Programming framework used for performing distributed computing processes. Data are processed in two main steps mapping and reducing.

Except for three main Hadoop components, enabling base functionality of distributed system, there are plenty of other complementary projects, supplementing additional facilities.

**1.4.3 HDFS**

Hadoop Distributed File System, or just HDFS – it is main storage system, used by Hadoop components. It was designed for storing huge amounts of information (terabyte even petabyte), simultaneously providing high rate of access speed. Files stored in HDFS are splits onto blocks, replicated and distributed over the nodes of cluster, this provides high rate of reliability and computing speed. The system is designed in the way that, files in HDFS are written only once, and editing of already written files is not possible. As the HDFS is block-structured file system, files during their loading have to be physically splits onto the blocks of fixed size. Size of the blocks is set in cluster’s configuration file (default size is 64 MB). Each file block is then replicated with the preset replication quotient (default value is 3). Replicated blocks are stored on physically separated machines, there are used suitable mechanisms to spread copied blocks over cluster nodes, what ensures that in case of some failure, needed data are not lost and could be read from another node. As the HDFS was designed for storing big files, it is normal that files stored in HDFS could easily exceed storage capacity of single machine’s hard-drive, nothing guarantees that blocks of one file will be stored on the same machine; blocks are spread over the cluster automatically using the suitable hash function.

HDFS provides all mentioned mechanisms by running set of daemons. These daemons could be understood as an individual program, each performing its specific role. Some daemons are running on one server, and another are running across multiple machines. Currently HDFS includes these daemons:

**Name Node**— the Name Node is the centrepiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept.

**Data Node** — a Data Node stores data in the HadoopFileSystem. A functional file system has more than one Data Node, with data replicated across them. On start-up, a Data Node connects to the Name Node; spinning until that service comes up. It then responds to requests from the Name Node for file system operations. Client applications can talk directly to a Data Node, once the Name Node has provided the location of the data. Similarly, MapReduce operations farmed out to Task Tracker instances near a Data Node, talk directly to the Data Node to access the files. Task Tracker instances can indeed should be deployed on the same servers that host Data Node instances, so that MapReduce operations are performed close to the data.

**Secondary Name Node** — The Secondary Name Node (SNN) is an assistant daemon for monitoring the state of the HDFS. Like the Name Node, each cluster has one SNN, and it typically resides on its own machine as well. No other Data Node or Task Tracker daemons run on the same server. The SNN differs from the Name Node in that this process doesn’t receive or record any real-time changes to HDFS. Instead it communicates with the Name Node to take snapshots of the HDFS metadata at intervals defined by the Hadoop configuration. The Name Node is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data. Nevertheless, a Name Node failure requires human intervention to reconfigure the cluster to use the SNN as the primary Name Node.

**Task Tracker**— A Task Tracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a Job Tracker. Every Task Tracker is configured with a set of slots*,* these indicate the number of tasks that it can accept. When the Job Tracker tries to find somewhere to schedule a task within the MapReduce operations, it first looks for an empty slot on the same server that hosts the Data Node containing the data, and if not, it looks for an empty slot on a machine in the same rack. The Task Tracker spawns a separate JVM processes to do the actual work; this is to ensure that process failure does not take down the task tracker. The Task Tracker monitors these spawned processes, capturing the output and exit codes. When the process finishes, successfully or not, the tracker notifies the Job Tracker. The Task Trackers also send out heartbeat messages to the Job Tracker, usually every few minutes, to reassure the Job Tracker that it is still alive. These messages also inform the Job Tracker of the number of available slots, so the Job Tracker can stay up to date with where in the cluster work can be delegated.

**Job Tracker**— the Job Tracker Daemon is the liaison between application and Hadoop. Once you submit your code to your cluster, the Job Tracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they’re running, Should a task fail, the Job Tracker will automatically relaunch the task, possibly on a different node, upto a predefined limit of retries. There is only one Job Tracker daemon per Hadoop cluster. It is typically run on a server as a master node of the cluster.

**1.4.4. Benefit of Hadoop usage**

Apache Hadoop has been the driving force behind the growth of the big data industry. Hadoop brings the ability to cheaply process large amounts of data, regardless of its structure. Existing enterprise data warehouses and relational databases excel at processing structured data and can store massive amounts of data, though at a cost. This requirement for structure restricts the kinds of data that can be processed and it imposes an inertia that makes data warehouses unsuited for agile exploration of massive heterogeneous data. The amount of effort required to warehouse data often means that valuable data sources in organizations are never minded. This is where Hadoop can make a big difference.

* 1. **PROJECT BACKGROUND**

In this section, we would like to briefly introduce the background of batch and stream processing frameworks.

**1.5.1 Batch Processing Frameworks**

When a short response time is not strictly required, batch processing is a widely used way to process considerable volumes of data without any user intervention. For batch processing, input data is collected beforehand, and then processed in batches.

Hadoop is a batch processing framework and data to be processed are stored in the HDFS [3], a powerful tool designed to manage large datasets with high fault-tolerance. MapReduce [4], the heart of Hadoop, is a programming model that allows processing a substantial amount of data in parallel. Figure 1.2 shows an example of the MapReduce model. It has three major processing phases: Map, Shuffle, and Reduce. Traditional relational database organizes data into rows and columns and stores the data in tables. MapReduce uses a different way, it uses key/value pairs. The Map function performs sorting and filtering by keys, and then shuffles the intermediate results to the downstream operators which perform reduce tasks. The Reduce function applies summary operations on the intermediate data generated by Map.

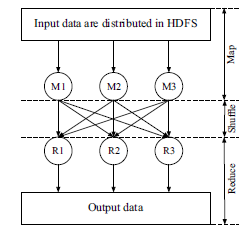


Figure1.2: Process of MapReduce: Map, Shuffle, and Reduce.

Shuffle is one of the dependencies between the operators and their parents. Generally speaking, there are three kinds of dependencies: One-to-one, Shuffle, and Join [1]. One-to-one is the case when the node has only one parent and conversely the output of the node is consumed by at most one downstream operator. Shuffle has been shown in Fig. 1.2, which is the case when each node gives its output to all downstream nodes. When a node has either one-to-one dependency or shuffle dependency with each of its parent nodes, it is called Join.

Spark [5] has been a prevailing framework for batch processing since proposed in 2010. When it runs programs in memory, it achieves up to 100\_ faster than Hadoop. The upshot for Spark is it introduces an immutable, fault tolerant and parallel data structure called Resilient Distributed Dataset (RDD)[5].

The biggest problem for batch processing is high latency (for minutes), which is the delay between inputs and outputs. Furthermore, since batch processing deals with large volumes of data at one time, as long as there are any changes of the data, reprocessing of the batch job is required. Computations for batch processing could be complex due to the large data size.

**1.5.2. Stream Processing Frameworks**

The natural question that arises is: can we find a faster way to process data? Stream processing processes one data element or a small size of data in the stream at a time and the data are processed immediately upon arrival. For stream processing, computations are relatively simple and independent and it benefits from lower latency, typically seconds.

In order to support stream processing, Spark Streaming is proposed [6]. It divides the stream of data into batches of very small time intervals, which are defined as Discretized Streams. Spark Streaming is built on Spark and these Discretized Streams are treated as RDDs to perform computations. Strictly speaking, Spark Streaming cannot do real stream processing but does micro-batching jobs. Micro-Batching is a special case of batch processing and it processes data with a very small batch size, which can be seen as a mix between batch processing and stream processing. Figure 1.3 shows the relations among batch processing, stream processing, and micro-batching. How to select a proper way to process data? It depends on the data size and requirements of the response time. Table 1.1 shows a comparison of these data processing approaches.

Apache Storm is a stream processing framework, it operates continuous stream of data. Apache Storm uses tuples, named lists of values, as its data model, and defines a stream as an unbounded sequence of tuples. Unlike Hadoop run MapReduce jobs, Storm runs “topologies”. A topology is a Directed Acyclic Graph (DAG) that users submit to Apache Storm for computations. Like Spark, Apache Storm is a fast, scalable, and high fault-tolerant parallel framework.

These frameworks are designed for the data processing within the same data centre since they do not consider complicated situations of communication and scheduling in the wide-area data analytics. For example, in Spark, bandwidths across different sites are assumed to be uniformly distributed. Consequently, many representative works designed novel mechanisms for the wide area analytics based on these frameworks.

**1.6 Optimization Issues**

In order to achieve better performance, there are a lot of optimization issues we need to consider in the data analytics.

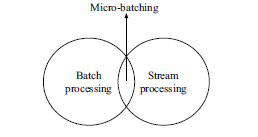


Figure 1.3: Relations among batch processing, stream processing, and micro-batching.

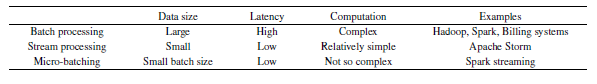


Table 1.1 Comparison of data processing approaches.

**Latency**: In general, latency can be defined as a delay between receiving the request and generating the response. We build data centre’s geographically with the purpose of achieving low latencies for local users [7]. Nevertheless, as data volumes keep increasing at a tremendous rate, it is still time consuming to transfer such substantial amount of data across data centres[8].Many cloud services have very stringent requirements for latency, even a delay of one second can make a great difference[9]. A large body of academic works have focused on optimizing latency.

**Bandwidth**: Bandwidth is the data transferred at one time. As bandwidth is scarce and expensive in the Wide Area Network (WAN)[10], optimizing bandwidth becomes another important issue in the analytics of geo-distributed data. Low latency may lead to the use of additional bandwidth, thus there is a trade-off between bandwidth and latency. In this paper, bandwidth within the same datacenter is called intradatacenter bandwidth, while bandwidth among different data centres is called inter-datacenter bandwidth.

**Fault-tolerance**: High fault-tolerance is a big challenge when performing large scale data processing across datacentres. Fault-tolerance is the way that a system responses to a variety of network failures. A high fault-tolerant data processing system can continue operating when some components of the system fail [11], which can reduce costs and time for the reprocessing when failures happen.

**Overhead**: In order to achieve the optimal performance, sometimes we need to do extra work

That can cause overhead. Overhead can be any excess resource like bandwidth, memory or computation time.

* 1. **Mechanisms for Data Analytics**

Since the volume of data grows exponentially, the traditional centralized approach presents a number of limitations. In the wide area data analytics, data is generated in a geo-distributed fashion and some new constraints need to be considered, such as privacy concern.

Distributed execution is a strategy widely used in the wide area analytics. This strategy is to

push computations down to local datacenter and then aggregate the intermediate results to do further processing. We use a motivation example to show the high-level idea of this strategy. A social network provider wants to get hot search words for every ten minutes. Click logs and search logs are two kinds of input data sources. Click logs store web server logs of user activities and search logs are the records of user requests for information. Base data is born distributed across datacenter, what we want to do is to give our execution strategy to minimize data traffic across different datacenter. If we use a centralized execution in Fig. 1.4, then we will observe that data traffic across datacenter is 600 GB per day. However, if we use a distributed execution strategy depicted in Fig. 5, then data size will be much smaller after pre-processing in the local datacenter, data traffic across datacenter is only 5 GB per day. Moreover, lower latency can be achieved by using the distributed execution.

Table 1.2 shows the summary of the wide area analytics. Many mechanisms are proposed for this problem. In this section, we will discuss high-level ideas of these representative mechanisms with some examples and give our thoughts about the proposed solutions.

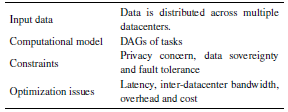


Table 1.2 Summary of the Data analytics.

**1.7.1. Pixida**

When a job is submitted for execution, we can get the job’s task-level graph and locations of input data partitions from distributed storage systems like HDFS. Thus the data traffic minimization problem can be translated into the graph partitioning problem, where the job’s task-level graph is split into partitions, each partition contains the tasks in the same datacenter. Intra-datacenter bandwidth is cheaper than interdatacenter bandwidth, thus the objective is to minimize data traffic across different datacenter.

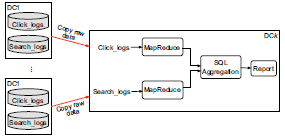


Figure: 1.4 A motivation example: Centralized approach

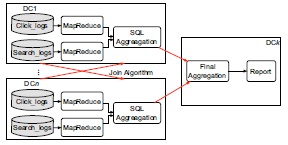


Figure: 1.5 Distributed execution of the motivation example: Pre-process in the local datacenter, then apply join algorithms to the intermediate results and do final aggregation to get the report.

Pixida [1] is a scheduler designed to minimize data traffic across geo-distributed datacenter. This scheduler models the scheduling goal by using the graph partitioning method. It uses a new topology abstraction called “SILO”, which is a group of nodes that belong to the same location. Pixida transforms the task-level graph into the SILO-level graph, which can reduce the size of the graph. Tasks for the same operator that run in the same location are merged into a single node. After getting the job’s SILO-level graph, Pixida will assign tasks to different SILOs. Here, SILOs can also be regarded as datacenter.

It is obvious that the job’s task-level graph and the locations of input data partitions can be known as soon as the job is submitted for execution, yet how to know the output data size of each task in the graph? Pixida designs the Tracer to solve this problem. In the Tracer phase, Pixida selects a sample of the input partitions like 20% to run the job, and then the Tracer extrapolates the output data size of each task.

Figure 1.6 gives an example of a job’s SILO-level graph. In1, In2, and In3 represent input data partitions in three datacenter. Different graph partitions result in different inter-datacenter traffic. We use cost to represent the data transfer size across different datacenter. Traditionally, we could generate the graph partitioning problem as a min k-cut problem: “Given a weighted graph G(V,E,W ) and a set of k terminals, find a minimum weight set of edges E0 such that removing E0 from G separates all terminals”[1]. Here k represents k SILOs (datacenter). We can solve this traditional min-k cut problem by using the Edmonds-Karp algorithm, and get the partitions with a cost of7 C 8 C 5 C 6 D 26 in the left side of Fig. 1.7.However, we can make it “cheaper”. Here we did not consider the case of “Dataflow Forking”, when an operator forwards its output to more than one downstream operator. Pixida formulates a generalized in k-cut problem and presents a novel flow-based approximation algorithm (follows the structure of Edmonds-Karp algorithm) to solve this problem. The basic idea of solving the case of “Dataflow Forking” is to add an extra vertex Ve between M2 and its children in the graph. Figure 8 shows this idea. Edge (M2, Ve) represents a single cross-SILO transfer of the same data.

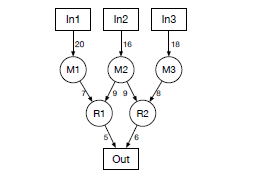


Figure1.6: Jobs SILO-level graph, with the output data size of each task based on the statistics of the Tracer phase.

Thus we could get the optimal graph partitioning in the right-side of Fig. 1.7, it has a cost of 7C9C8 D 24, which is better than the left-side partitioning. As R1 and R2are in the same SILO S4, one cross-SILO transfer from

M2 in SILO S3 to R1 or R2 is enough.

Pixida is appropriate for batch processing. After integrated with Spark, it achieves up to 9\_ bandwidth reduction compared with Spark. However, there are also some constraints about the graph partitioning method. First, it assumes that cross-SILO transfers are considered of equal cost. When we consider a complex pattern, the partitioning problem will become more complicated.

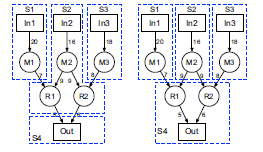


Figure 1.7 Partitions of the job’s SILO-level graph. The left-side partitioning has a cost of 26, which does not consider the case of “Dataflow Forking”. The right-side partitioning is optimal with a cost of 24.

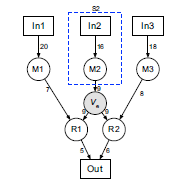


Figure1.8: Extra vertex is added between M2 and its children

Besides, it cannot be used for real time processing where data is processed upon arrival, since the input data partitions are static. Moreover, the Tracer phase used in Pixida adds computational and time overhead. Finally, Pixida only considers data transfers across datacenter, it does not tackle with the issue of latency.

**1.7.2 WANalytics**

WANalytics[12] is a Hadoop-based system, it also targets in minimizing inter-datacenter traffic. It is an extended version of Pixida.

**Caching**: WANalytics uses the idea of “caching” to cache all intermediate results to reduce data transfers. Figure 1.9 shows the basic idea of caching. At start, DC2 asks DC1 for the result of running query q0, then DC1 runs the query and sends the result of q0 to DC2. In the meantime, DC1 also caches the copy of q0’s result. If DC1 asks DC2 for the result of a new query q1, then DC2 runs the query but only sends the difference of q0’s result and q1’s result. By this way, data transfers across datacenter are greatly decreased when there are repetitive queries.



Figure1.9: Data transfer optimization: Caching.

This caching method actually worsens CPU and storage use, it solely reduces data transfers across different datacenter. Distributed queries are always seen as DAGs, when multiple DAGs share common sub queries, this method greatly helps to reduce data transfers as a result of sending the difference between new results and old results.

**Optimizing execution**: Given a set of recurrent DAGs of tasks, with constraints of sovereignty, WANalytics uses a greedy heuristic for the optimizing execution. It processes all DAGs in parallel. In each DAG, it goes over tasks in the topological order, and greedily chooses the lowest-cost available strategy for each task. There are two things the optimizing execution decides: (1) Strategy for each task, e.g., hash join or semi-join; (2) Which task goes to which datacenter, like the task graph partitioning problem.

**Pseudo-distributed measurement**: Similar as the Tracer phase in Pixida, Pseudo-distributed measurement is used to measure the cost of each execution strategy for a DAG. For some settings, measuring all options considered by the greedy heuristics could be very slow, which is a limitation for this measurement.

Figure 1.10 shows the architecture of WANalytics. WANalytics consists of two main components: a runtime layer and a workload analyser. In the runtime layer, there is a coordinator in a master datacenter that interacts with datacenter managers at each datacenter. In each datacenter manager, there is a caching mechanism. Analyst submits DAGs of queries, and then the coordinator asks the workload analyser for the best distributed execution plan.

After getting DAGs of queries, the workload analyser gives a distributed execution plan by using greedy heuristics and pseudo-distributed measurements. The workload analyser uses a robust evolutionary approach. It starts by supporting the existing centralized approach, then uses a continuous adaptation: It firstly comes up with some DAG execution plans of the workload, secondly measures their costs by using pseudo distributed measurements, then computes a new best plan by using the optimizing execution, finally it deploys the new best plan.

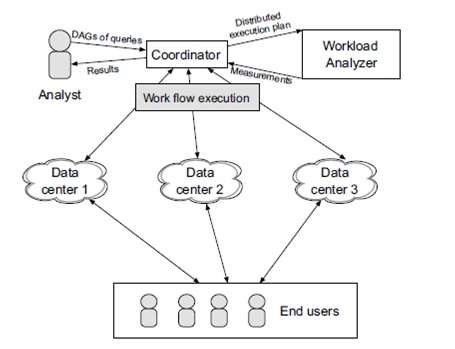


Figure 1.10: WANalytics architecture

WANalytics focuses on the optimization of data transfers, but fault tolerance and latency are not addressed. Besides, this system only partially supports the requirement of data sovereignty. It considers data storage requirements but allows arbitrarily queries on the data.

**1.7.3. Geode**

Geode is a system built upon Hive that presented in Ref. [2]. It is an extended version of WANalytics [12].This system uses a relational model and supports SQL analytics on the geo-distributed data. Figure 1.11 shows its architecture, which is similar as WANalytics. The

core of Geode is the command layer. The command layer receives SQL queries, and then gets a distributed query execution plan from the workload optimizer. After getting the plan, the command layer runs the plan and outputs results.

In the workload optimizer, Geode gives the query execution plan by solving an Integer Linear Program (ILP). The objective function of the ILP is to minimize total data transfers in the DAG. Constraints are the requirements of fault tolerance and data sovereignty. WANalytics uses the greedy heuristic for determining the query execution plan, but in some cases the heuristic approach fails to get the optimal solution. However, the ILP can only support up to ten datacenter in the experiment, greedy heuristic scales much better than the ILP. Moreover, the ILP is more accurate but it runs slower than the greedy heuristic. Geode gives a tradeoff between the running time and the solution quality.

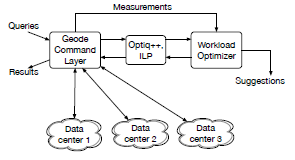


Figure 1.11: Geode architecture.

**1.7.4 Iridium**

Pixida, WANalytics, and Geode only consider minimizing data transfers across datacenter but ignore the issue of latency, which is significant for low-latency processing applications. Iridium [13] is a system for the low latency geo-distributed analytics. It uses the task placement to reduce query response time. Figure 1.12 shows an example of simple MapReduce tasks across datacenter. DC1 and DC2 are two datacenter. DC1 has a downlink bandwidth of 100 MB/s, but its uplink bandwidth is low and only 10 MB/s. DC1 downloads the data of DC2 to do the Map task, and then generates a substantial amount of the intermediate data. There are no reduce tasks in DC1, so it needs to upload the intermediate data to DC2 to do reduce tasks. Because of the very low uplink bandwidth, query response time will be affected significantly. We call the link with low bandwidth as the bottleneck link. Usually, query response time is determined by the bottleneck link. If we could put more reduce tasks in DC1, there will be less data uploaded to DC2 via uplink, thus query response time will be greatly reduced. This task placement problem can be formulated as a Linear Program (LP) with the objective of minimizing query response time. But the LP is appropriate for tasks of a single query. For DAGs of tasks, Iridium uses a greedy approach to perform the task placement by using the LP independently in each stage of the query. However, the best task placement is still limited by data locations. In order to better reduce query response time, Iridium also uses the data placement.

For the example in Fig. 1.12, we have already found the bottleneck link is the downlink of DC1, then

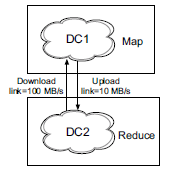


Figure 1.12: Example of simple MapReduce tasks across

Datacenter.

We could move the data out of DC1 to DC2 before the query arrives, query response time will be greatly reduced. For example, assume the next query will arrive in 24sthe intermediate data in DC1 is 150 MB. Before the query arrives, we move all the intermediate data out of DC1 to DC2. Moving time is 150/10= 15 s, which is smaller than the query arrival time. For the data placement, the basic intuition is to identify the bottleneck link first, then move data out of the bottleneck site. When the intermediate data was generated at time t0, and the next query arrives at time t1, t1 - t0 is called the query lag. Iridium uses a greedy algorithm in the data placement, it seeks to move the high-value datasets and move the dataset that has the smallest query lag. For complex DAGs of tasks, sometimes results are not global optimal but local optimal. This data placement can be combined with the task placement, after completing the data placement before the query arrives, we could continue to use the task placement to reduce query response time.

One problem about data placement is how to estimate query arrivals? For repetitive queries, it is easy to estimate the query lag based on the old query. However, for other situations, it is hard to estimate. Iridium makes the following simple assumption that works well: for instance, if the dataset was generated at time t and two queries arrived at time (t + a) and time (t + b) , and a<=b. Then we will assume that the next two queries will arrive at time (t + b) +a and time (t + b) + b.

One of the advantages of Iridium is that it supports both stream processing as well as batch processing. This system achieves low query response time by optimizing data and task placement, and it also considers the trade-off between the bandwidth cost and query response time.

Task and data placement are also used in other academic works. NetStitcher [14] is a system for online data processing. It uses data placement to stitch unutilized bandwidth, and rescues up to five times additional bandwidth. Guet al.[15] minimized the cost of servers and communication in geo-distributed datacenter, and formulated the cost minimization problem as a mixed-integer nonlinear programming to answer how to apply data and task placement under constraints of the remote data loading and quality of the service satisfaction.

**1.7.5 Swag**

Finishing some tasks of a job does not mean faster job completion time. Job scheduling is another aspect that we need to consider in the wide area analytics. Hung etal. [16] targeted on reducing the average job completion time by using novel job scheduling algorithms, and achieved up to 50% improvement in the average job completion time with low overhead. They used two scheduling algorithms: Reordering and Workload- Aware Greedy Scheduling (SWAG).

Now we use an example to show the idea of these two scheduling algorithms. There are three jobs computed in three datacenter, Table 1.3 presents the subjobsizes in each datacenter, which is the number of tasks for jobs. There are two assumptions in Hung et al.[16]: (1) Each datacenter has one computation source; (2) Each datacenter serves one task per second. Figure 1.13 shows the approach of First Come First Scheduling (FCFS) approach across three datacenter. Job order of FCFS is A ->B ->C, x axis represents the queue length (number of tasks to be served). If we use this FCFS scheduling, the average job completion time will be 13.3 s. However, we find that tasks of job A at DC2 has a higher completion time, then we may delay job A in favour of other jobs with faster completion time at datacenters. This is the basic idea of Reordering. Figure 1.14 shows this reordering approach, which moves some jobs later in the local queue, as long as delaying them will not increase the average job completion time. Reordering approach for our example achieves average job completion time of 13 s, which is better than FCFS scheduling. The core idea of reordering approach is “no harm”, which means that this approach provides “non-decreasing performance improvement for any scheduling algorithm [16]”.

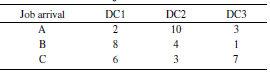


Table1.3: Sub-job sizes in datacenter

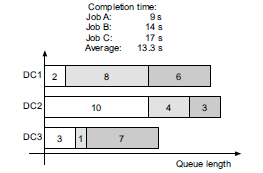


Figure1.13: First Come First Serve Scheduling, with the job order: A->B->C.

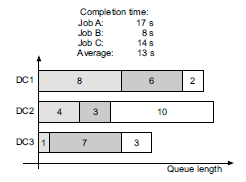


Figure1.14: Reordering Approach, with the job order: B -> C->A.

In order to achieve better average job completion time, Hung et al. [16] presented the SWAG algorithm. The basic idea for this algorithm is to greedily serve the job that finishes the fastest. When we schedule jobs by their finishing time, we also need to take the local queue size into consideration. For our example, job C finishes the fastest, so we serve job C first, and then the job B. Job A finishes the slowest, since it has 10 tasks to be finished in DC2, thus we serve job A at last. This scheduling approach by using SWAG achieves better average job completion time of 12.7 s than reordering. Figure 1.15 shows the idea of this scheduling method. Scheduling approach by using SWAG achieves better average job completion time of 12.7 s than reordering. Figure 15 shows the idea of this scheduling method.

**1.8 Discussion of Existing Mechanisms**

Analytics for geo-distributed datacenter in the wide area network have several aspects. Some mechanisms are batch processing, some are stream processing. Bandwidth and latency are two important optimization issues we consider in the wide area analytics.

Table 1. 4 show a comparison of these mechanisms we have discussed.

Graph Partitioning by Pixida: This graph partitioning method is appropriate for the

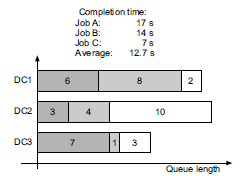
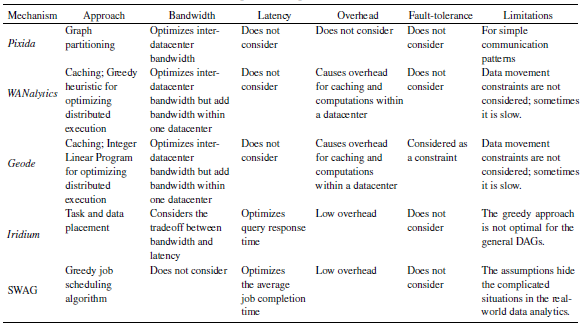


Figure: 1.15 SWAG algorithm with the job order: C->B->A.

Table 1.4: Comparison of representative mechanisms

Simple task graph since cross-SILO transfers are considered to be equal, which does not cover the general cases in real life.

Distributed Query Planning: WANalytics and Geode use a workload optimizer to find the best distributed execution plan. However, sometimes it may be slow for the workload optimizer to give the best execution strategy. Moreover, arbitrary queries are allowed on the data, which does not consider data movement constraints.

Task and Data Placement: Iridium first finds the bottleneck link and then uses task and data placement to optimize query response time. However, it is hard to estimate the query lag, and sometimes the estimation will not be so accurate. Another limitation for the data placement is the data movement constraint, for some situations we cannot move data out of a datacenter arbitrarily.

Job scheduling: The idea is to schedule job by finishing time with the consideration of tasks at datacenter. It is simple but useful for optimizing the average job completion time. SWAG uses a greedy scheduling algorithm, yet it is not appropriate for a general job whose DAG consists of multiple stages. Furthermore, the assumptions for SWAG hide the complexity of real world data analytic jobs.

**1.9. FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

**1.9.1 Economical Feasibility**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### 1.9.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**1.9.3. Social Feasibility**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**1.10 Organization of the Project**

The rest of the thesis is organized as follows: Chapter 2 specifies the software requirements specifications. Chapter 3 presents the design of the project. Chapter 4 describes the implementation. Chapter 5 presents the various testing we undertook along with the test results, and finally concludes the thesis.

**2. SOFTWARE REQUIREMENTS AND ANALYSIS**

**2.1 INTRODUCTION**

With the enormous increase in the amounts, variety and velocity of online data. It is a benefit to store the data in an efficient, reliable and maintainable way and processing that huge amount of data is very useful to organizations as well as customers or end users.

* + 1. **Purpose**

Traditionally, collections of data are stored and processed in a single data center. As the volume of data grows at a tremendous rate, it is less efficient for only one data enter to handle such large volumes of data from a performance point of view.

**2.1.2 Scope**

* Gathering dataset from various social network sites
* Comparing various social network sites storage capacity for future use.

**2.1.3 Overview**

Finally, compares various social network sites providing number user and number storage space occupied by data centre geographically representing using line and Graph

**2.2 OVERALL DESCRIPTION**

This project works only for large amounts of data. It gives very poor performance for small datasets and for structured smaller datasets also this project will not suit.

**Sample input data**

{"Age": 73,"Education": " High school graduate","MaritalStatus": " Widowed","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1700.09,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 58,"Education": " Some college but no degree","MaritalStatus": " Divorced","Gender": " Male","TaxFilerStatus": " Head of household","Income": 1053.55,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 18,"Education": " 10th grade","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 991.95,"Parents": " Not in universe","CountryOfBirth": " Vietnam","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 0}

{"Age": 9,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1758.14,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 10,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1069.16,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 48,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 162.61,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 42,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1535.86,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 28,"Education": " High school graduate","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Single","Income": 898.83,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 30}

{"Age": 47,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 1661.53,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 34,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1146.79,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 8,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 2466.24,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 32,"Education": “High school graduate","MaritalStatus": “Never married","Gender": “Female","TaxFilerStatus": “Nonfiler","Income": 2021.27,"Parents": “Not in universe","CountryOfBirth": “?","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 0}

{"Age": 51,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 2441.22,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 46,"Education": " High school graduate","MaritalStatus": " Divorced","Gender": " Female","TaxFilerStatus": " Single","Income": 978.16,"Parents": " Not in universe","CountryOfBirth": " Columbia","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 52}

{"Age": 26,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Single","Income": 2604.91,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 13,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1520.08,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 47,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Single","Income": 404.9,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 39,"Education": “10th grade","MaritalStatus": “Married-civilian spouse present","Gender": “Female","TaxFilerStatus": " Joint both under 65","Income": 1274.04,"Parents": " Not in universe","CountryOfBirth": " Mexico","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 0}

{"Age": 16,"Education": " 10th grade","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1555.29,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 35,"Education": " High school graduate","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1790.75,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 49}

{"Age": 12,"Education": " Children","MaritalStatus": " Never married","Gender": " Male","TaxFilerStatus": " Nonfiler","Income": 455.02,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 27,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1004.69,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 56,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 1500.08,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 32}

{"Age": 46,"Education": " Masters degree(MA MS MEng MEd MSW MBA)","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 999.46,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 55,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 1483.69,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 2,"Education": " Children","MaritalStatus": " Never married","Gender": " Male","TaxFilerStatus": " Nonfiler","Income": 1660.53,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 1,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 848.25,"Parents": " Mother only present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 37,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 2671.99,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 4,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1188.42,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 37,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Never married","Gender": " Male","TaxFilerStatus": " Single","Income": 1331.35,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

**2.3 SPECIFIC REQUIREMENTS**

**2.3.1 External Interface Requirements**

**2.3.1.1 Hardware Interfaces**

* Processor : above 1.4GHz
* RAM : above 2GB
* Hard Disk : above 50GB

**2.3.1.2 Software Interfaces**

* Ubuntu
* Hadoop cluster
* Java & php
* Virtual Box Oracle

**2.3.2 Functional Requirements**

**2.3.2.1 Modules**

**Stream processing frameworks**

In the wide area data analytics, data is generated in a geo-distributed fashion and some new constraints need to be considered, such as privacy concern.

Stream processing processes one data element or a small size of data in the stream at a time and the data are processed immediately upon arrival. For stream processing, computations are relatively simple and independent and it benefits from lower latency, typically seconds.

The latency can be defined as a delay between receiving the request and generating the response. We build datacenter geographically with the purpose of achieving low latencies for local users. Nevertheless, as data volumes keep increasing at a tremendous rate, it is still time consuming to transfer such substantial amount of data across datacenter. Many cloud services have very stringent requirements for latency, even a delay of one second can make a great difference.

**Batch processing frameworks**

Hadoop is a batch processing framework and data to be processed are stored in the HDFS, a powerful tool designed to manage large datasets with high fault-tolerance. MapReduce, the heart of Hadoop, is a programming model that allows processing a substantial amount of data in parallel. An example of the MapReduce model has three major processing phases: Map, Shuffle, and Reduce. Traditional relational database organizes data into rows and columns and stores the data in tables. MapReduce uses a different way, it uses key/value pairs.

**2.3.2.2 Input**

Here actually the input data is in the form of big data which can be obtained from the various social network sites at website.

**2.3.2.3 Output**

The output would be storing the big data efficiently, obtaining the positivity of data and an easy querying on various social network sites in the form of line and pie grap.

**2.3.3 Non-functional Requirements**

* **Reliability**

It depends on the maintaining the pseudo distributed mode with required prerequisites.

* **Performance**

Performance of stock exchange data analysis is measured by storing the big data in HDFS file system and the time taking by the Hive to process the data on top of the HDFS file system.

* **Quality**

Quality will be dependent on successfully maintaining the pseudo distributed mode as per requirements and the queries that work on top of Hadoop.

* **Maintainability**

The design is done in a secure way and easy to maintain.

* **Testability**

We can observe the nodes running on the machine through browser.

1. **SYSTEM DESIGN**

**3.1 Architecture Diagram**

3.1.1 Hadoop Architecture

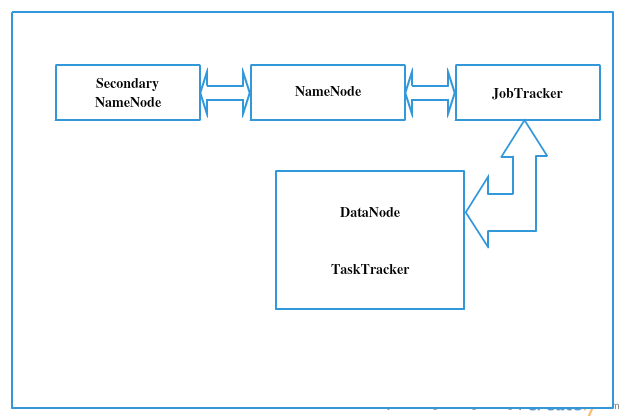


Figure 3.1.1: Hadoop Architecture

**3.2 UML Diagrams**

**3.2.1. Class Diagram**

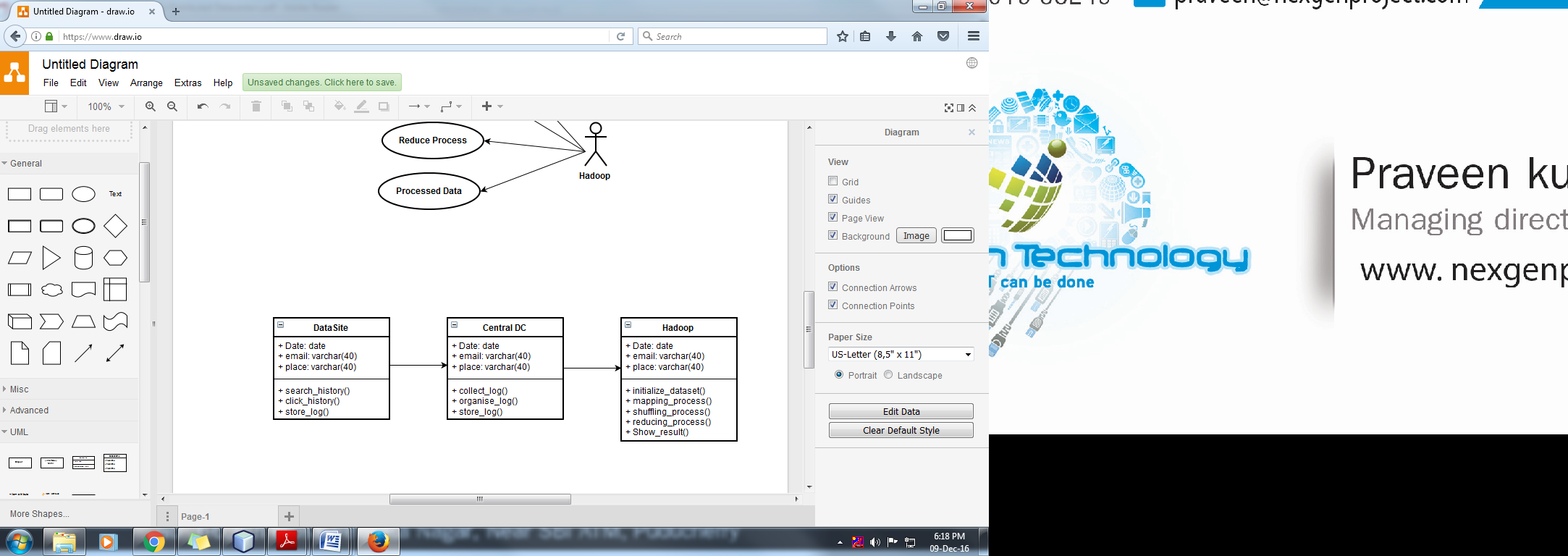


Figure 3.1.3: Class Diagram

**3.2.2 Use-case Diagram**

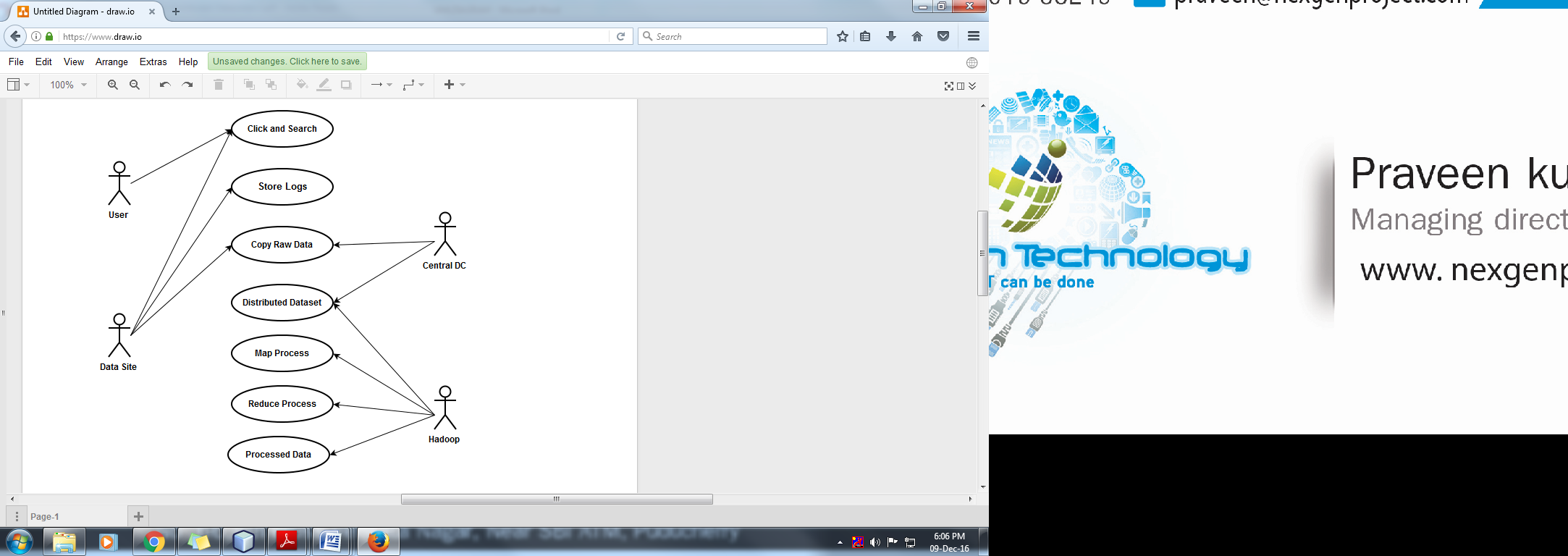


Figure 3.1.2: Use case Diagram

**3.2.3 Sequence Diagram**

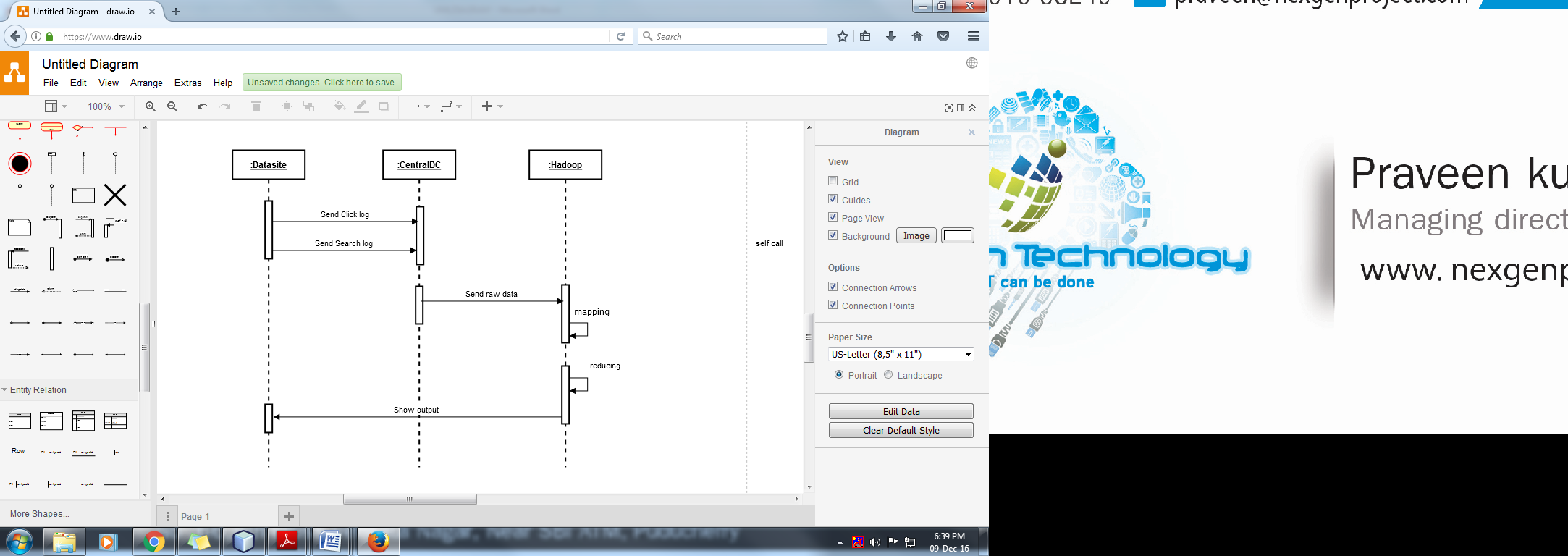


Figure3.1.4: Sequence Diagram

**3.2.4Activity Diagram**

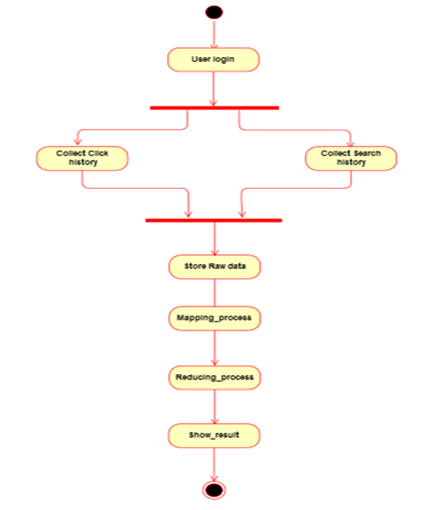


Figure 3.1.5: Activity Diagram

* 1. **Basic Design of this project**
     1. **Data Flow Diagram**

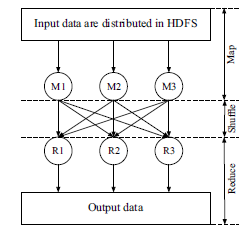


Figure3.2.1: Data Flow Diagram

1. **IMPLEMENTATION**

This project mainly contains three modules and they are implemented as follows

1. Gathering the Input data.
2. Forming the Pseudo Distributed Mode
3. Generating the output Data

**4.1 Gathering the Input data**

This module involves the following steps

**Sample input data**

{"Age": 73,"Education": " High school graduate","MaritalStatus": " Widowed","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1700.09,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 58,"Education": " Some college but no degree","MaritalStatus": " Divorced","Gender": " Male","TaxFilerStatus": " Head of household","Income": 1053.55,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 18,"Education": " 10th grade","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 991.95,"Parents": " Not in universe","CountryOfBirth": " Vietnam","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 0}

{"Age": 9,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1758.14,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 10,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1069.16,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 48,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 162.61,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 42,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1535.86,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 28,"Education": " High school graduate","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Single","Income": 898.83,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 30}

{"Age": 47,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 1661.53,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 34,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1146.79,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 8,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 2466.24,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 32,"Education": “High school graduate","MaritalStatus": “Never married","Gender": “Female","TaxFilerStatus": “Nonfiler","Income": 2021.27,"Parents": “Not in universe","CountryOfBirth": “?","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 0}

{"Age": 51,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 2441.22,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 46,"Education": " High school graduate","MaritalStatus": " Divorced","Gender": " Female","TaxFilerStatus": " Single","Income": 978.16,"Parents": " Not in universe","CountryOfBirth": " Columbia","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 52}

{"Age": 26,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Single","Income": 2604.91,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 13,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1520.08,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 47,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Single","Income": 404.9,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 39,"Education": “10th grade","MaritalStatus": “Married-civilian spouse present","Gender": “Female","TaxFilerStatus": " Joint both under 65","Income": 1274.04,"Parents": " Not in universe","CountryOfBirth": " Mexico","Citizenship": " Foreign born- Not a citizen of U S ","WeeksWorked": 0}

{"Age": 16,"Education": " 10th grade","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1555.29,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 35,"Education": " High school graduate","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1790.75,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 49}

{"Age": 12,"Education": " Children","MaritalStatus": " Never married","Gender": " Male","TaxFilerStatus": " Nonfiler","Income": 455.02,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 27,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 1004.69,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 56,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 1500.08,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 32}

{"Age": 46,"Education": " Masters degree(MA MS MEng MEd MSW MBA)","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 999.46,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 55,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Female","TaxFilerStatus": " Joint both under 65","Income": 1483.69,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 2,"Education": " Children","MaritalStatus": " Never married","Gender": " Male","TaxFilerStatus": " Nonfiler","Income": 1660.53,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 1,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 848.25,"Parents": " Mother only present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 37,"Education": " Some college but no degree","MaritalStatus": " Married-civilian spouse present","Gender": " Male","TaxFilerStatus": " Joint both under 65","Income": 2671.99,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

{"Age": 4,"Education": " Children","MaritalStatus": " Never married","Gender": " Female","TaxFilerStatus": " Nonfiler","Income": 1188.42,"Parents": " Both parents present","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 0}

{"Age": 37,"Education": " Bachelors degree(BA AB BS)","MaritalStatus": " Never married","Gender": " Male","TaxFilerStatus": " Single","Income": 1331.35,"Parents": " Not in universe","CountryOfBirth": " United-States","Citizenship": " Native- Born in the United States","WeeksWorked": 52}

Source java File

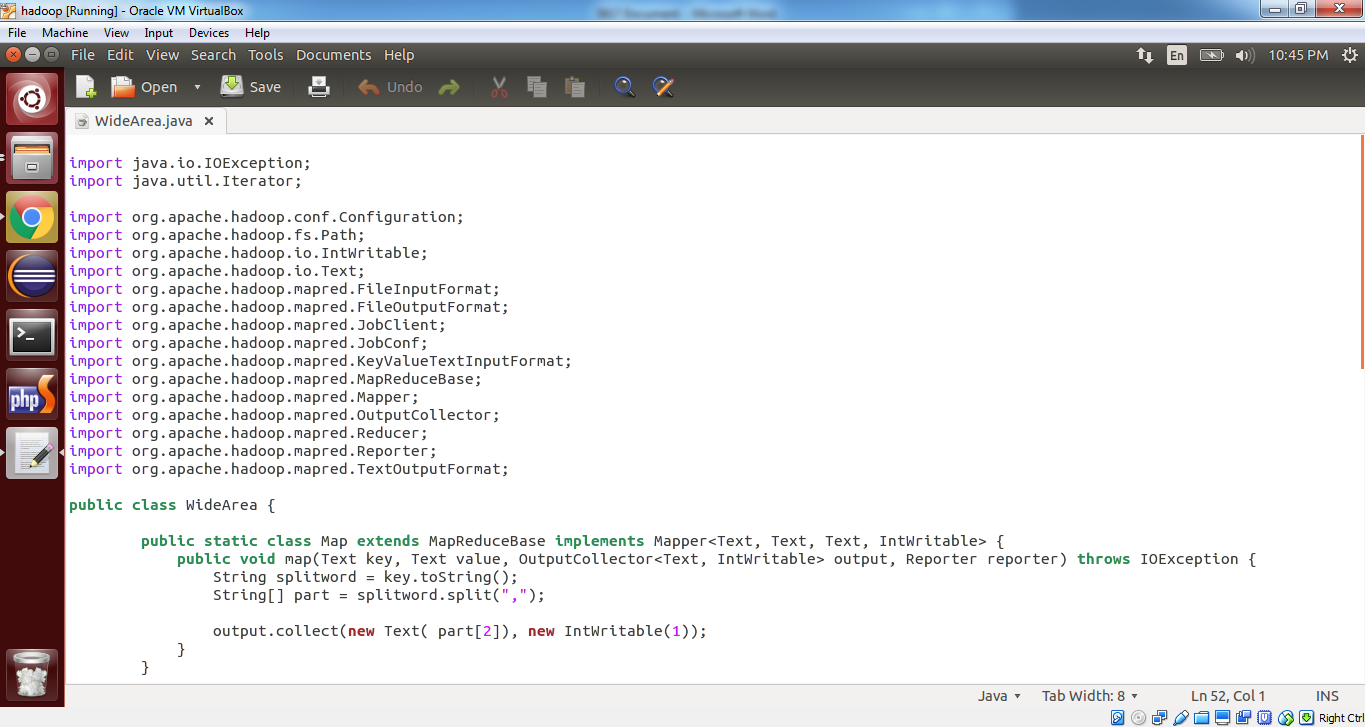


Figure 4.1: Source java file wideArea.java Page1

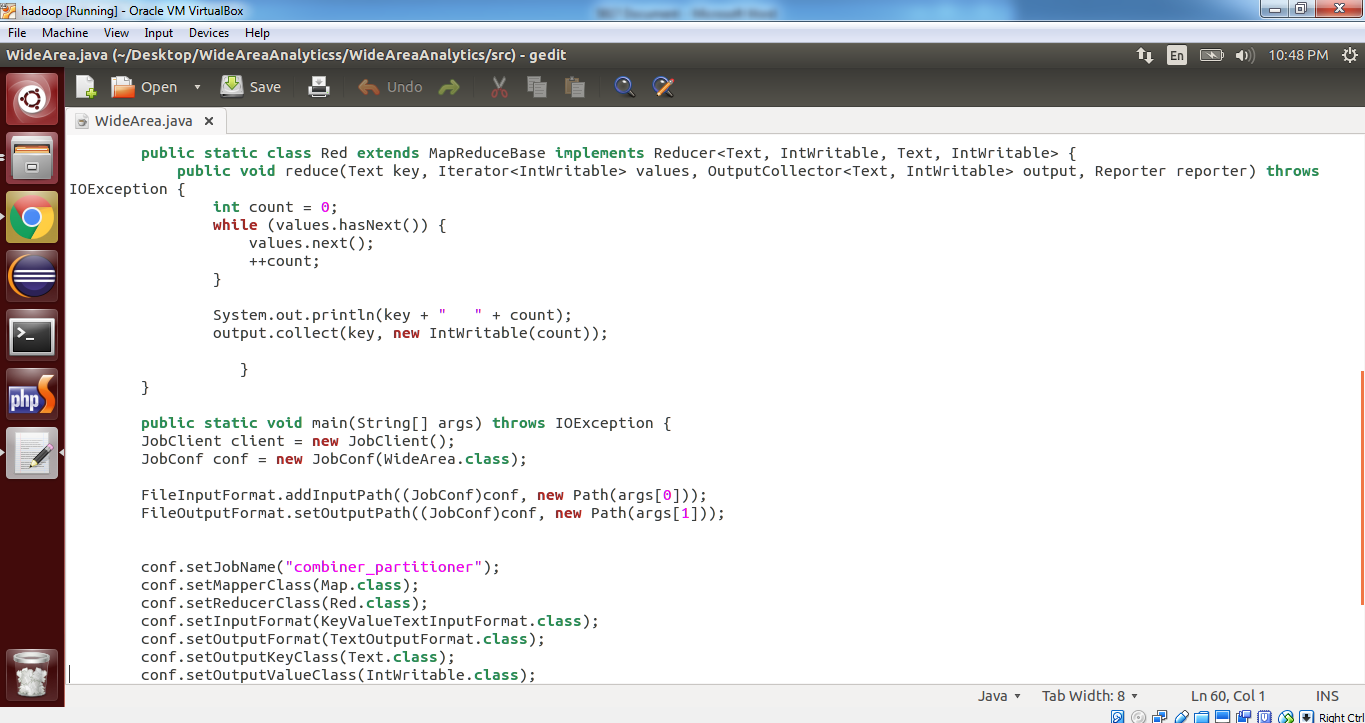


Figure 4.2: Source java file wideArea.java Page2

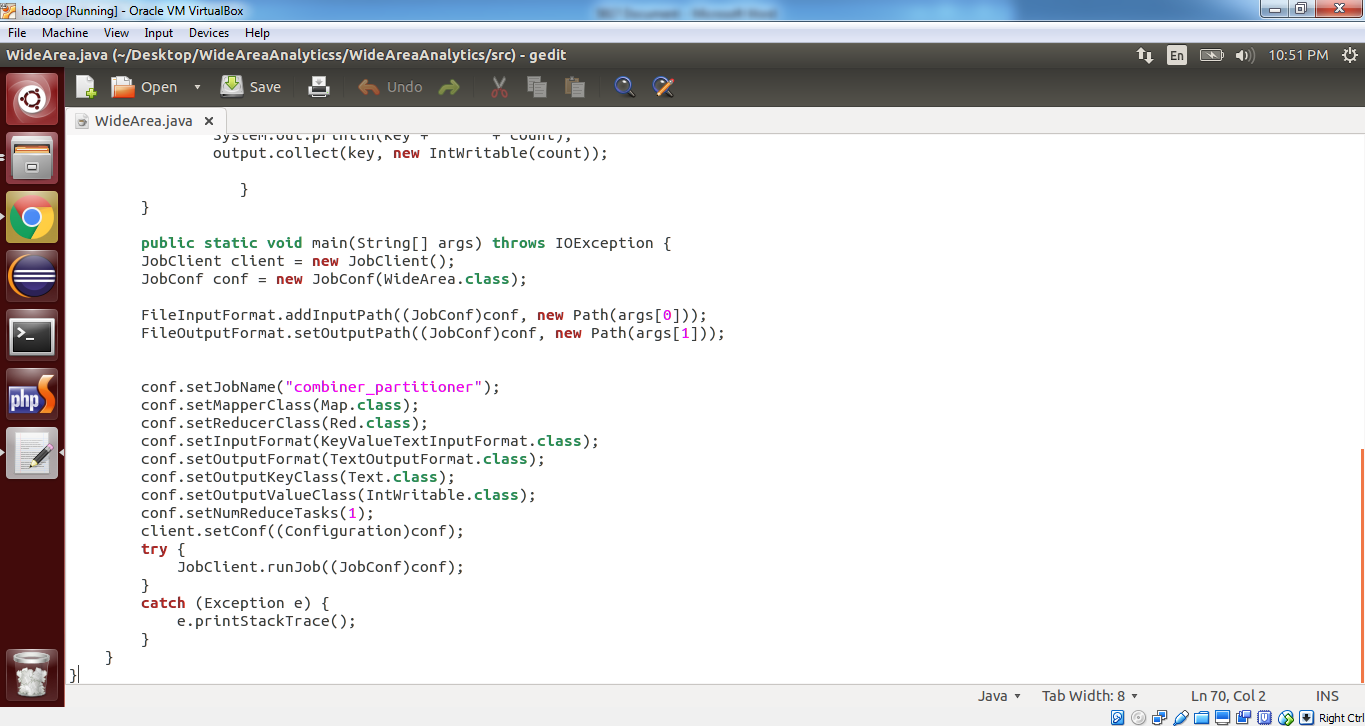


Figure 4.3: Source java file wideArea.java Page3

**4.2 Forming the Pseudo Distributed Mode**

1. **Installing Default Version of Java**

First update the source list by using following command and install the default version of java

user@node:$sudo apt-get update

user@node: $sudo apt-get install default-jdk

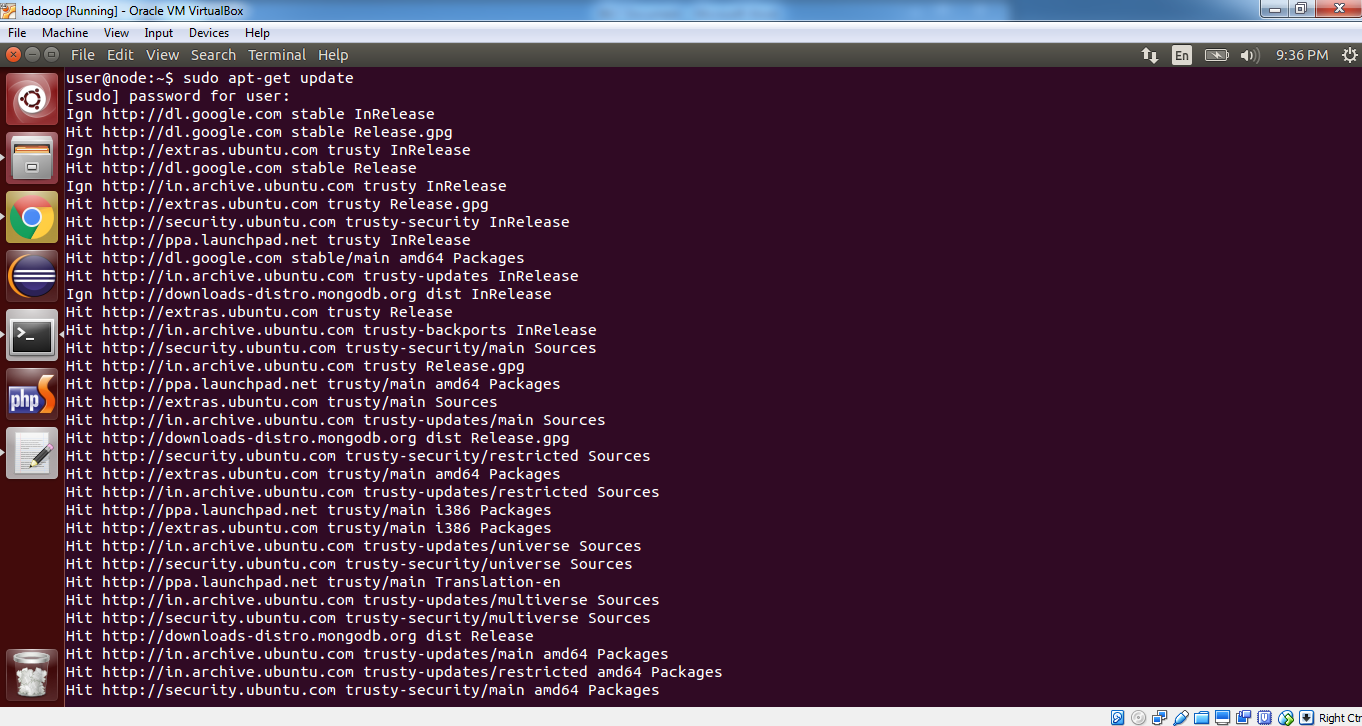


Figure 4.4: Updating the Source list

**b) Adding a Dedicated User**

user@node~$ sudo adduser hadoop

[sudo] password for user:

Adding user `hadoop’…

Creating home directory `/home/hadoop’…

Copying files from `/etc/skel’…

Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully

Changing the user information for hadoop

Enter the new value, or press ENTER for the default

Full Name []:

Room Number []:

Work Phone []:

Home Phone []:

Other []:

Is the information correct? [Y/n] y

1. **Installing ssh**

ssh has 2 main components :

1. ssh : The command we use to connect to remote machines - the client.

2. sshd : The daemon that is running on the server and allows clients to connect to the server.

The ssh is pre-enabled on Linux, but in order to start sshd daemon, we need to install

ssh first. Use this command to do that :

user@node~$ sudo apt-get install ssh

This will install ssh on our machine. If we get something similar to the following,

We can think it is setup properly:

user@node:~$ which ssh

/usr/bin/ssh

user@node:~$ which sshd

/usr/sbin/sshd

**CREATE AND SETUP SSH CERTIFICATES**

Hadoop requires SSH access to manage its nodes, that is remote machines plus our local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to local host.

So, we need to have SSH up and running on our machine and configured it to allow SSH public key authentication.

Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands. If asked for a filename just leave it blank and press the enter key to continue.

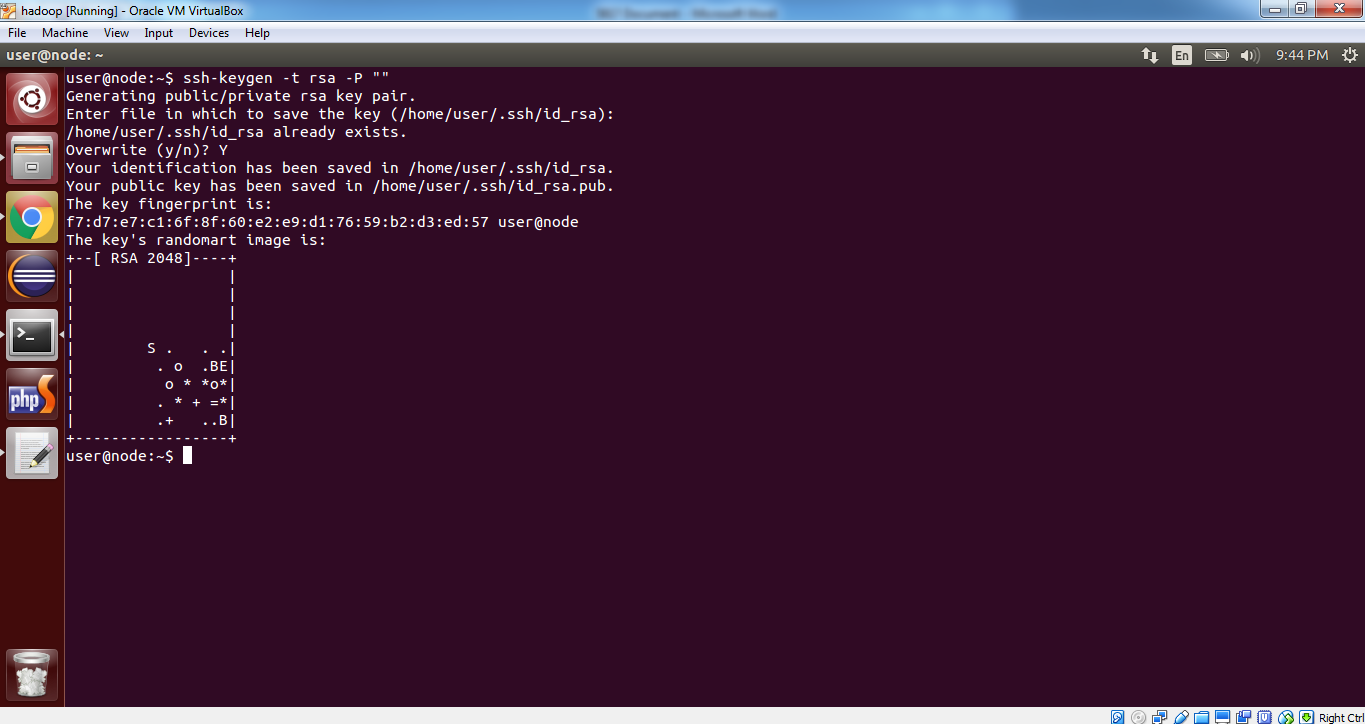


Figure 4.5: generating a rsa key pair

user@node:/home/hadoop$ cat$HOME/.ssh/id\_rsa.pub >> $HOME/.ssh/authorized keys

This command adds the newly created key to the list of authorized keys so that,

Hadoop can use ssh without prompting for a password.

We can check if ssh works:

user@node:/home/hadoop$ ssh localhost

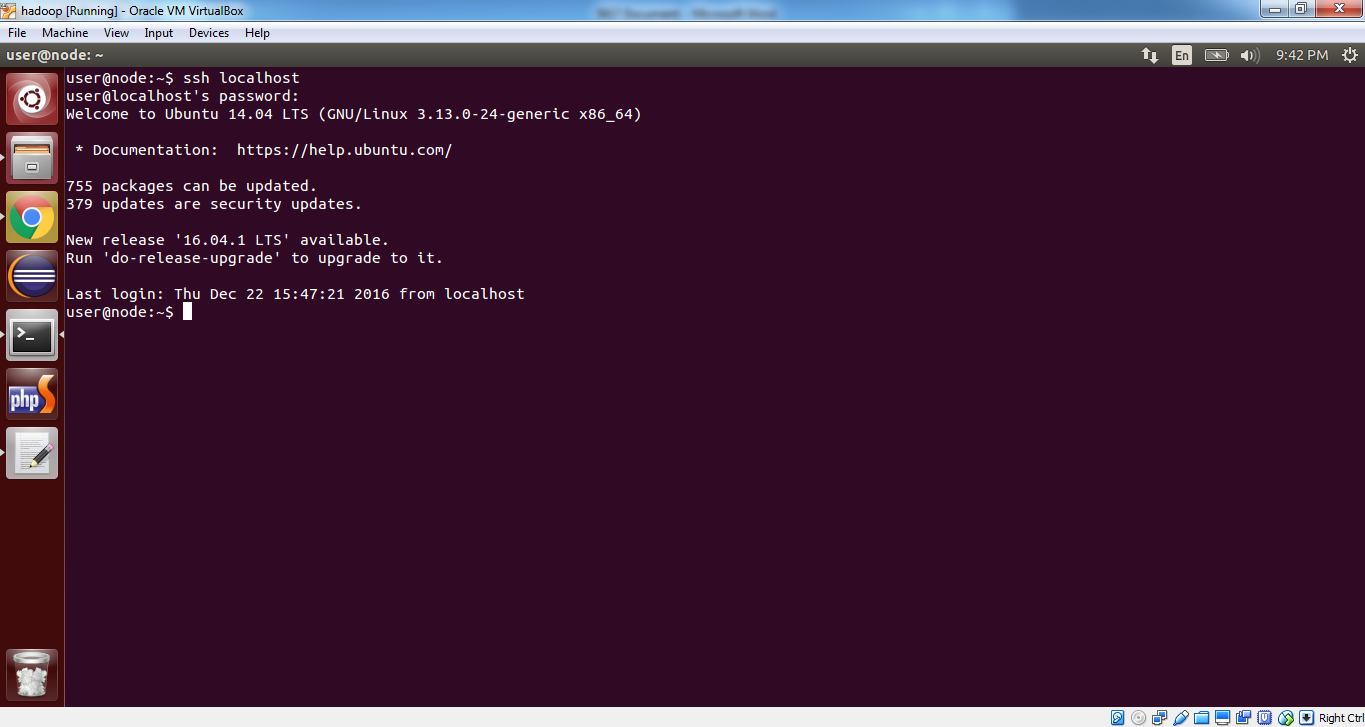


Figure 4.6: Localhost

1. **INSTALLING HADOOP:**

Download the file from the following command

user@node:~$wget <http://mirrors.sonic.net/apache/Hadoop/common/Hadoop-2.6.0/Hadoop-2.6.0.tar.gz>

extract file from the following command

user@node: tar xvzf Hadoop-2.6.0.tar.gz

We want to move the Hadoop file into the /usr/local/Hadoop directory using the following command:

user@node:~$ sudo mv Hadoop-2.6.0 /home/balu/work

Giving permissions to the Hadoop directory

user@node:~$ sudo chown -R hadoop:Hadoop Hadoop-2.60

present working director

user@node:~$ cd /home/hadoop/work/Hadoop-2.6.0

user@node:~$ ls

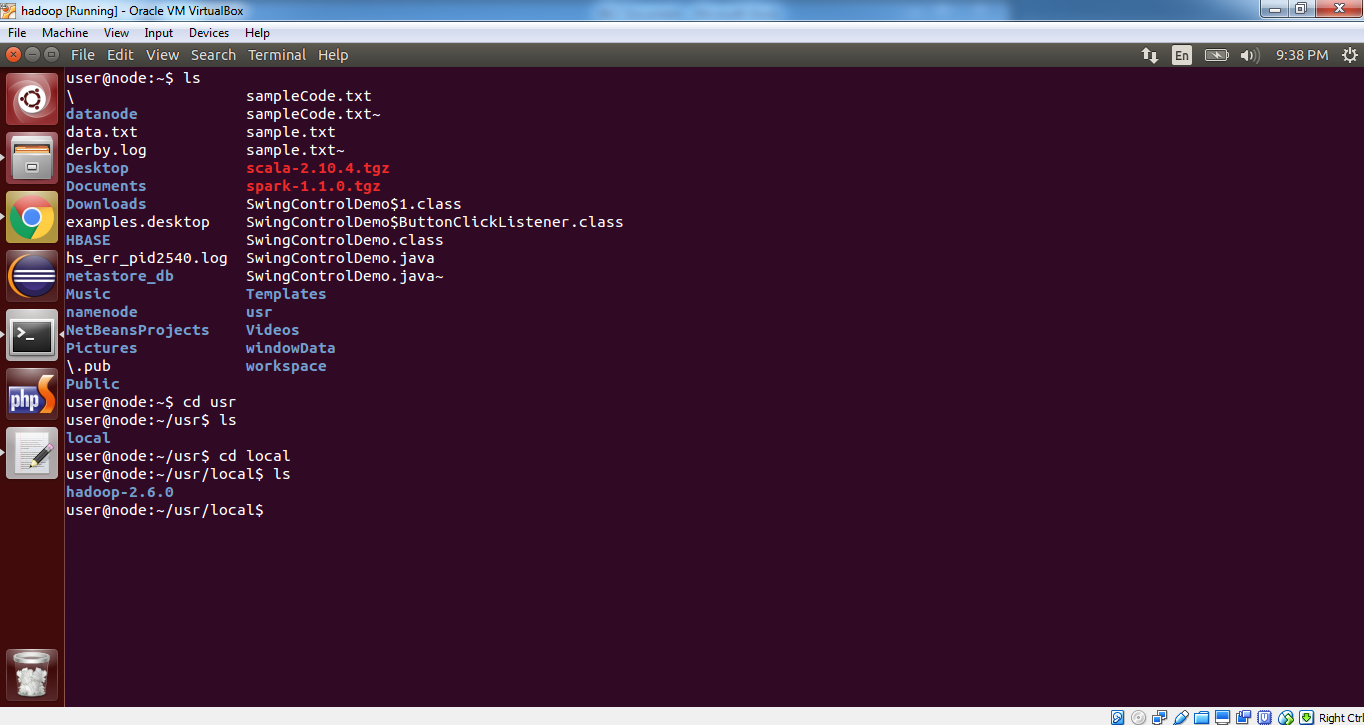


Figure 4.7: Listing the Files in Hadoop Directory

1. **SETUP CONFIGURATION FILES**

The following files will have to be modified to complete the Hadoop setup:

1. ~/.bashrc

2. /home/hadoop/work/hadoop-2.6.0/conf/Hadoop-env.sh

3. /home/hadoop/work/hadoop-2.6.0/conf/core-site.xml

4. /home/hadoop/work/hadoop-2.6.0/conf/mapred-site.xml.template

5. /home/hadoop/work/hadoop-2.6.0/conf/hdfs-site.xml

**1. ~/.bashrc:**

Before editing the .bashrc file in our home directory, we need to find the path where Java has been installed to set the JAVA\_HOME environment variable using the following

user@node: $ update-alternatives --config java

There is only one alternative in link group java (providing /usr/bin/java):

Nothing to configure.

Now we can append the following to the end of ~/.bashrc

#HADOOP VARIABLES START

export JAVA\_HOME=/usr/lib/jvm/java-1.7.0-openjdk-i386

export HADOOP\_HOME=/home/hadoop/work/hadoop-2.6.0

export PATH=$HADOOP\_HOME/bin:$JAVA\_HOME/bin:$PATH

#HADOOP VARIABLES END

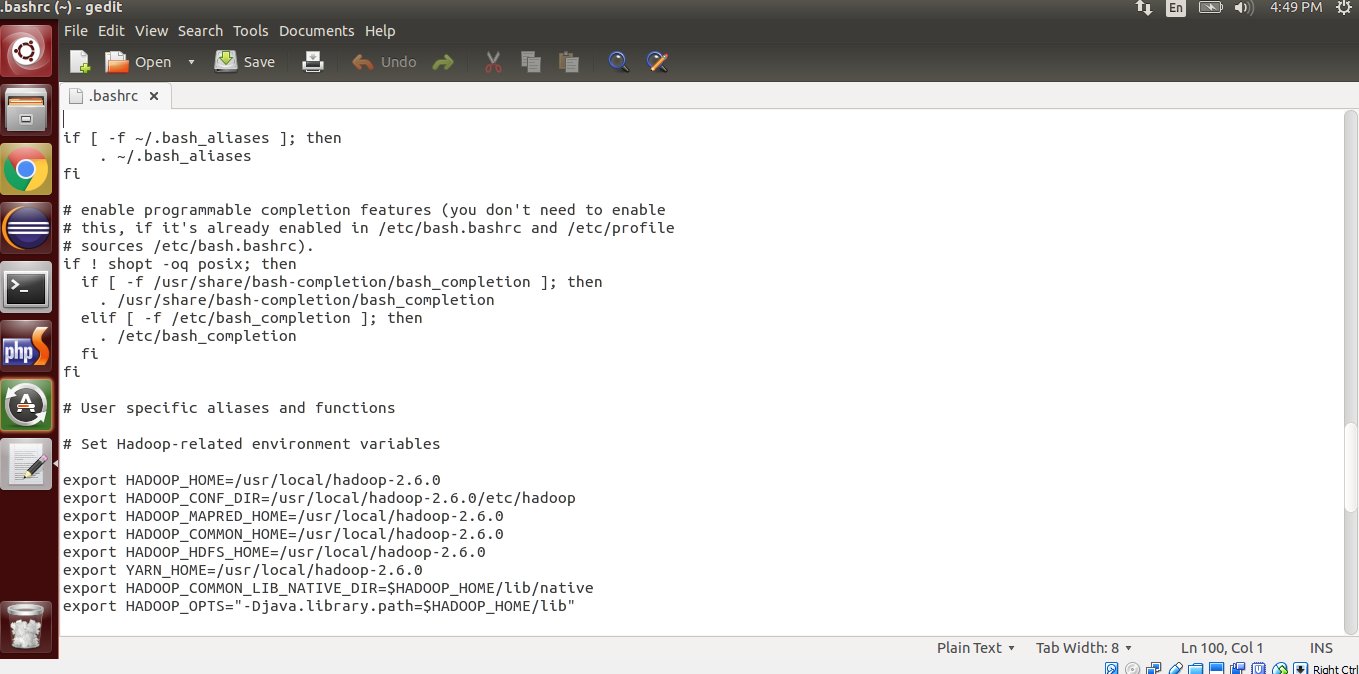


Figure4.8: configuring the .bashrc file

**2. /home/hadoop/work/hadoop-2.6.0/conf/Hadoop-env.sh**

We need to set JAVA\_HOME by modifying Hadoop-env.sh file

export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-i386

Adding the above statement in the Hadoop-env.sh file ensures that the value of

JAVA\_HOME variable will be available to Hadoop whenever it is started up.

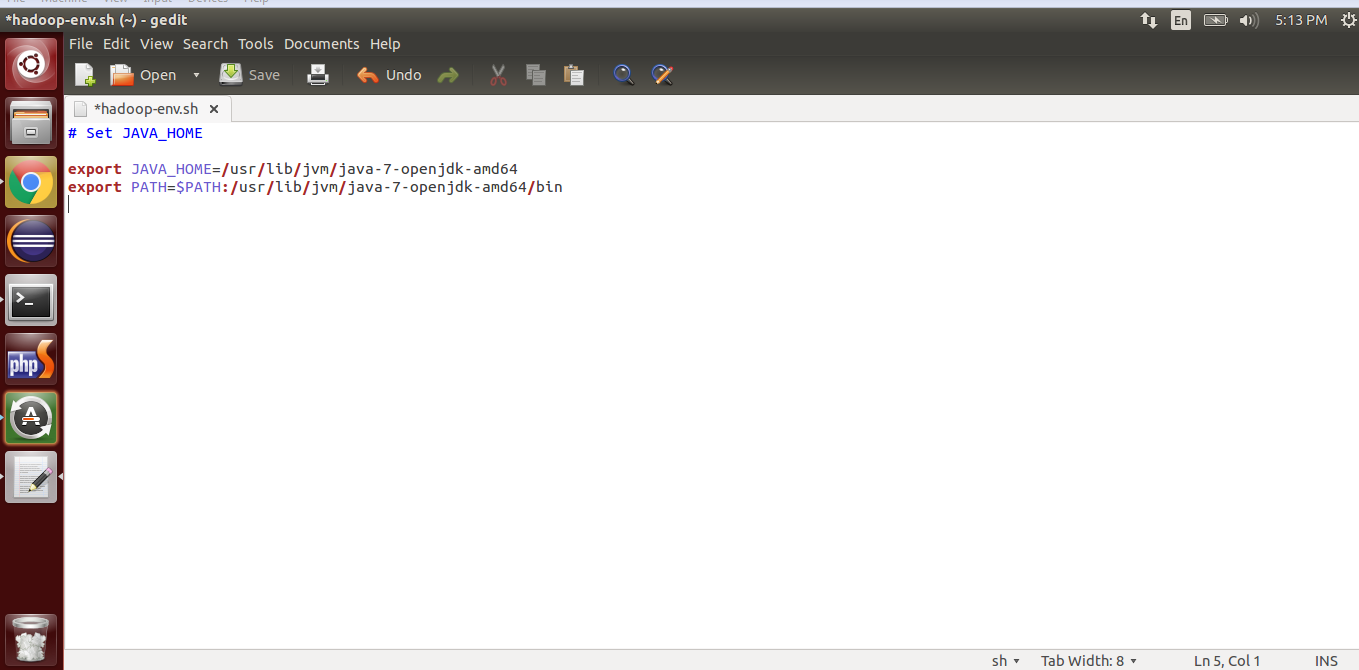


Figure 4.9: Configuring the Hadoop-env.sh

**3. /home/hadoop/work/hadoop-2.6.0/conf/core-site.xml:**

The /home/hadoop/work/hadoop-2.6.0/core-site.xml file contains configuration

properties that Hadoop uses when starting up. This file can be used to override the

default settings that Hadoop starts with.

Open the file and enter the following in between the <configuration></configuration> tag:

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hadoop/work/hadoopdata/tmp</value>

</property>

</configuration>

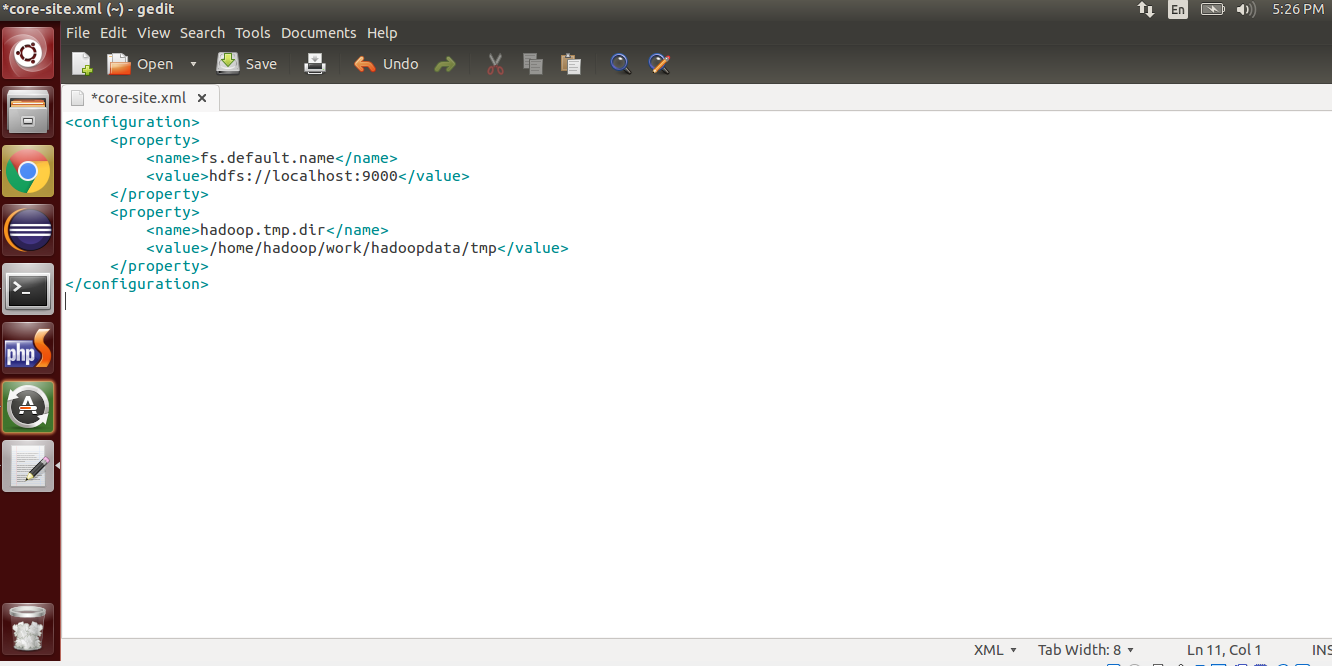


Figure 4.10configuring the core-site.xml

**4. /home/hadoop/work/hadoop-2.6.0/conf/mapred-site.xml**

By default, the /home/hadoop/work/hadoop-1.1.2/ folder contains the /home/balu/work/hadoop-1.1.2/conf/mapred-site.xml.template file which has to be renamed/copied with the name mapred-site.xml:

user@node:~$cp/home/hadoop/work/hadoop-2.6.0/conf/mapred-site.xml.template /home/hadoop/work/hadoop-2.6.0/conf/mapr ed-site.xml

The mapred-site.xml file is used to specify which framework is being used for

MapReduce. We need to enter the following content in between the <configuration>

</configuration> tag:

<configuration>

<property>

<name>mapred.job.tracker</name>

<value>localhost:9001</value>

</property>

<property>

<name>mapred.local.dir</name>

<value>/home/balu/work/hadoopdata/mapred/local</value>

</property>

<property>

<name>mapred.system.dir</name>

<value>/mapred/system</value>

</property>

</configuration>

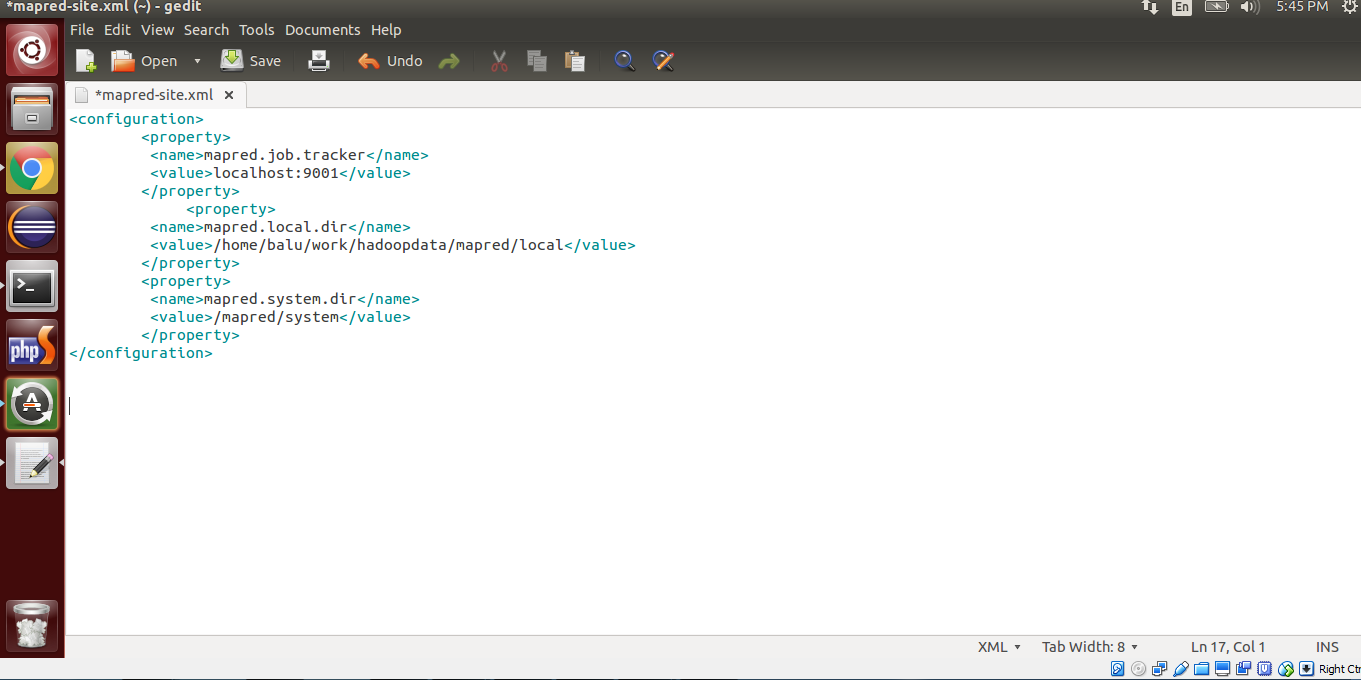


Figure 4.11: configuring the mapred-site.xml

**5. /home/hadoop/work/hadoop-2.6.0/conf/hdfs-site.xml**

The /home/hadoop/work/hadoop-2.6.0/hdfs-site.xml file needs to be configured for each host in the cluster that is being used. It is used to specify the directories which will be used as the namenode and the datanode on that host.

Before editing this file, we need to create two directories which will contain the

namenode and the datanode for this Hadoop installation. This can be done using the

following commands:

user@node~$ sudo mkdir -p /home/hadoop/work/hadoopdata/dfs

user@node:~$ sudo mkdir -p /home/hadoop/work/hadoopdata/tmp

user@node:~$ sudo chown -R Hadoop:Hadoop /home/hadoop/work/hadoopdata

Open the file and enter the following content in between the <configuration>

</configuration> tag:

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>/home/hadoop/work/hadoopdata/dfs/name</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>/home/hadoop/work/hadoopdata/dfs/data</value>

</property>

</configuration>

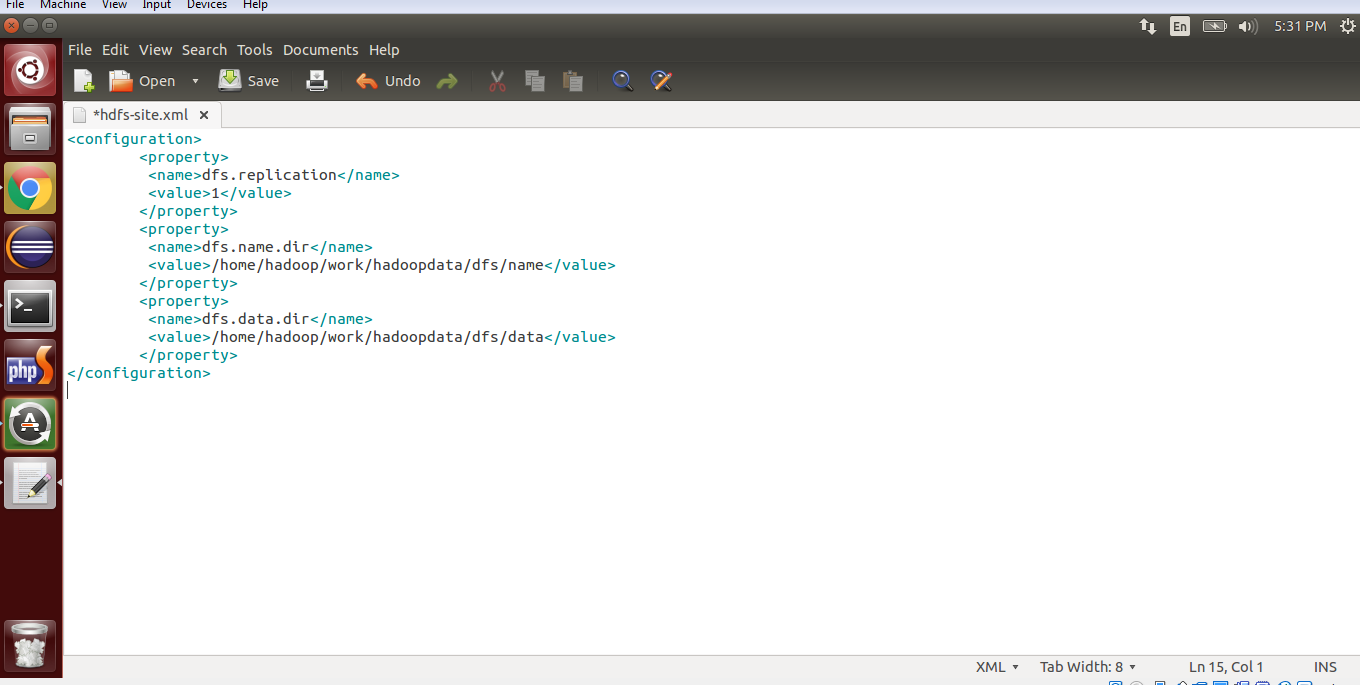


Figure 4.12: configuring the hdfs.site.xml

1. **FORMAT THE NEW HADOOP FILESYSTEM**

Now, the Hadoop filesystem needs to be formatted so that we can start to use it. The

format command should be issued with write permission since it creates current

directory under /usr/local/Hadoop\_store/hdfs/namenode folder:

user@node:~$ Hadoop namenode –format

**4.3GENERATING THE OUTPUT DATA**

**STARTING HADOOP**

Some of the important commands to execute the project let see

user@node:~$ start-all.sh

user@node:~$ jps

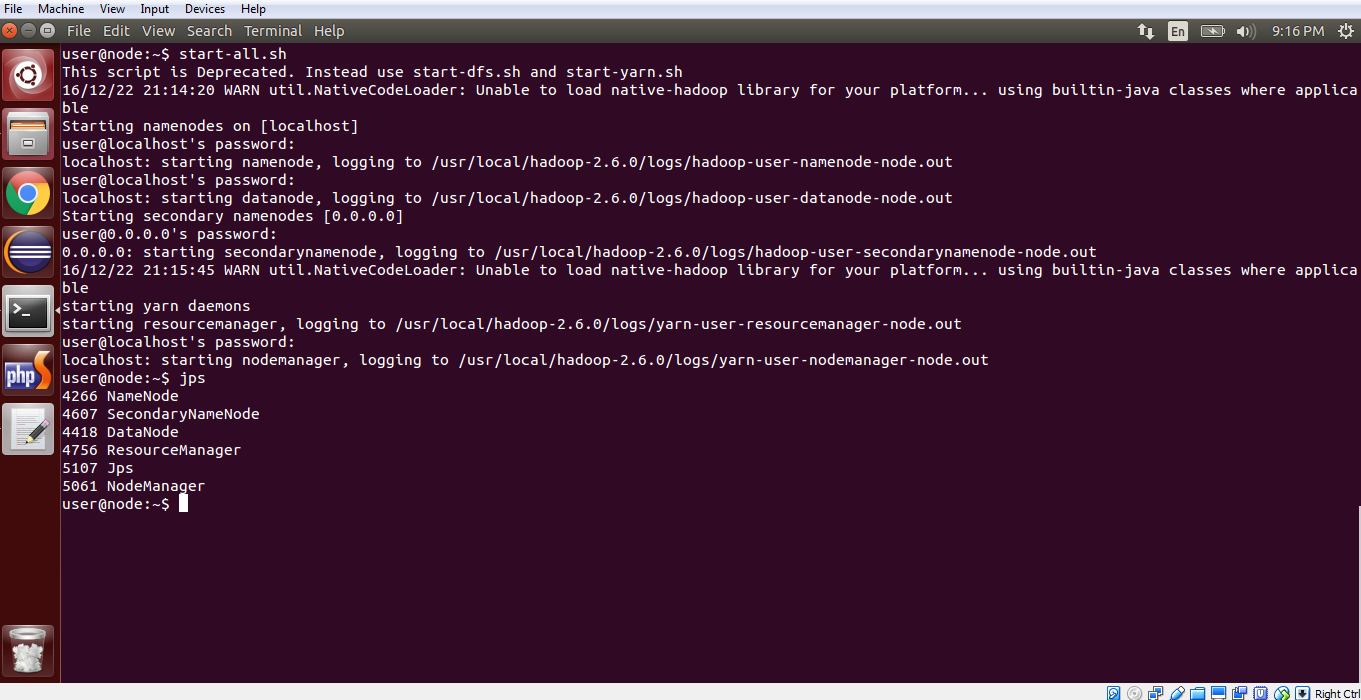


Figure 4.13: Starting Hadoop Demons

user@node:~$ hadoop fs -mkdir /WideArea15

user@node:~$hadoop fs -put /home/user/Desktop/WideAreaAnalyticss/dataset.csv /WideArea15

user@node:~$cd /home/user/Desktop/WideAreaAnalyticss/WideAreaAnalytics/

user@node:~$ /home/user/Desktop/WideAreaAnalyticss/WideAreaAnalytics/$hadoop jar WideArea.jar WideArea /WideArea15/dataset.csv /WideArea15/output

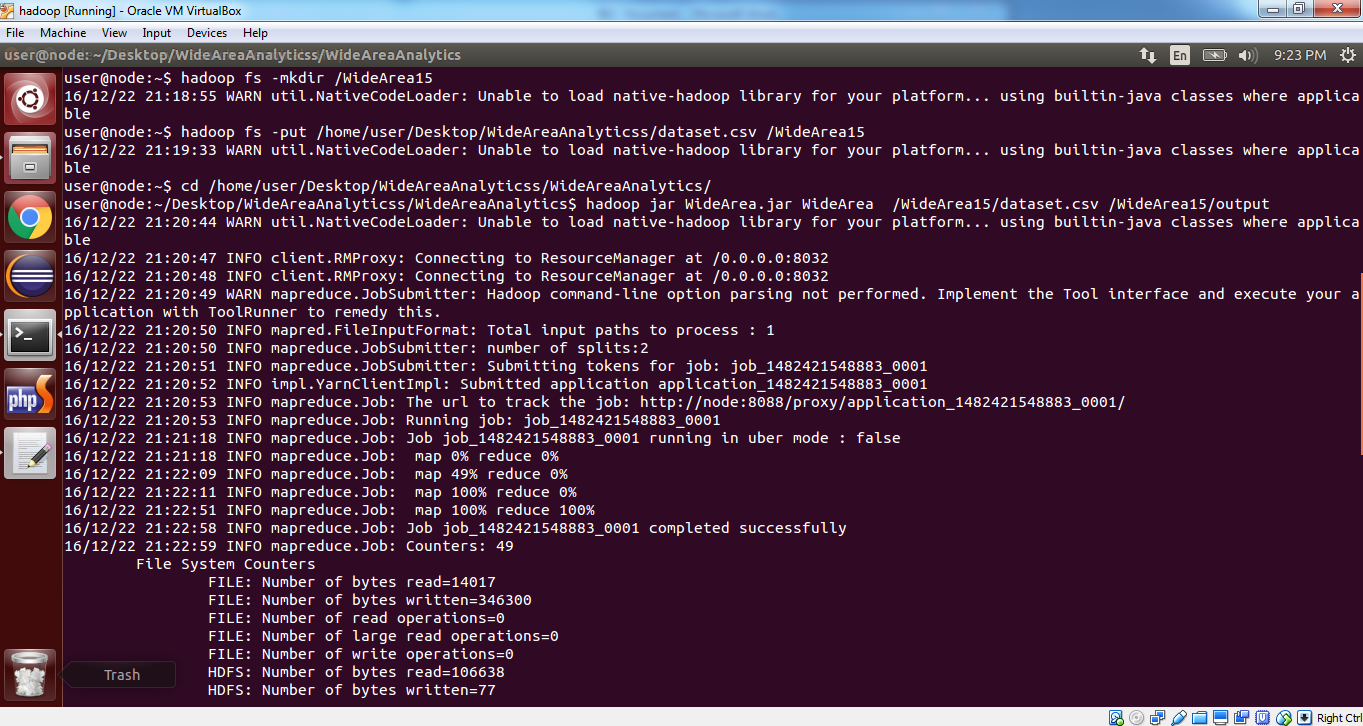


Figure 4.14: Creating Directory in Hadoop File Sytem and executing the hadoop jar file

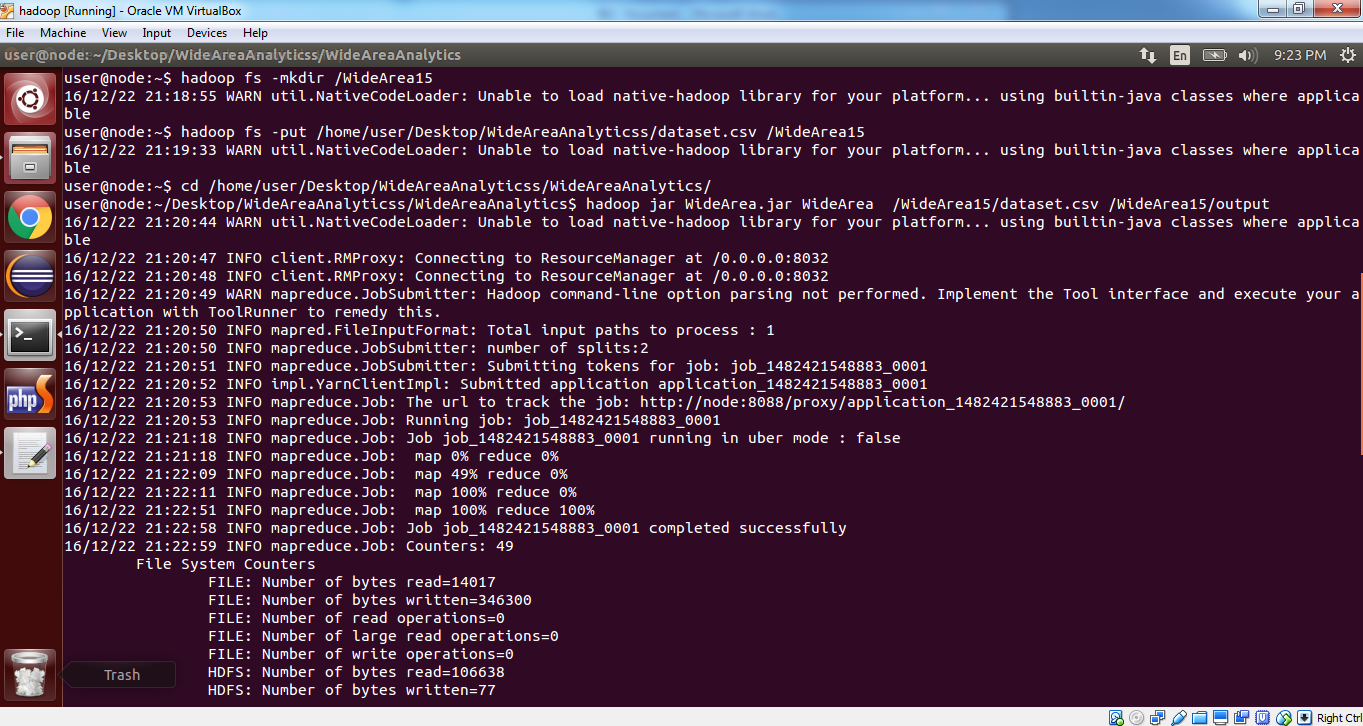


Figure 4.15: Hadoop process Map and Reduce generate result

Result:

Viewed in localhost

Goto chrome

localhost: 50070

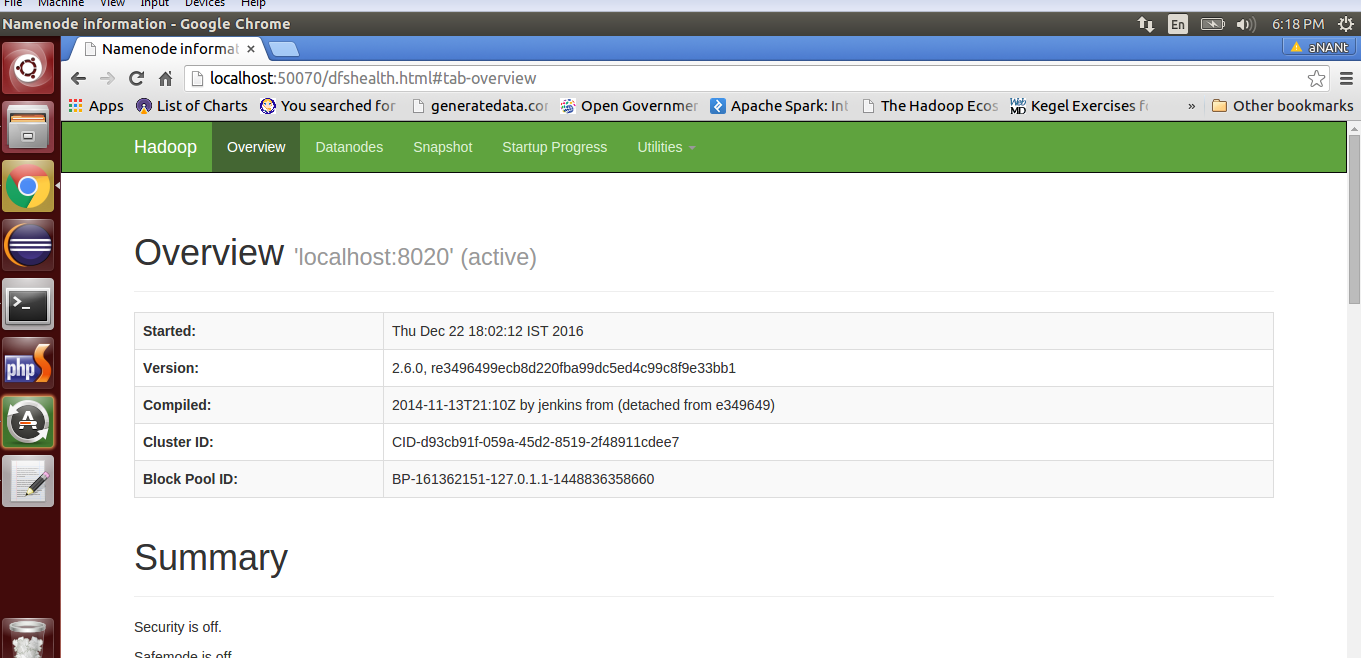


Figure 4.16: Running Hadoop on Local host

Check wideArea then click wideArea u will get output as success then in other tab the type

localhost/WideArea ----- The graph result will be viewed

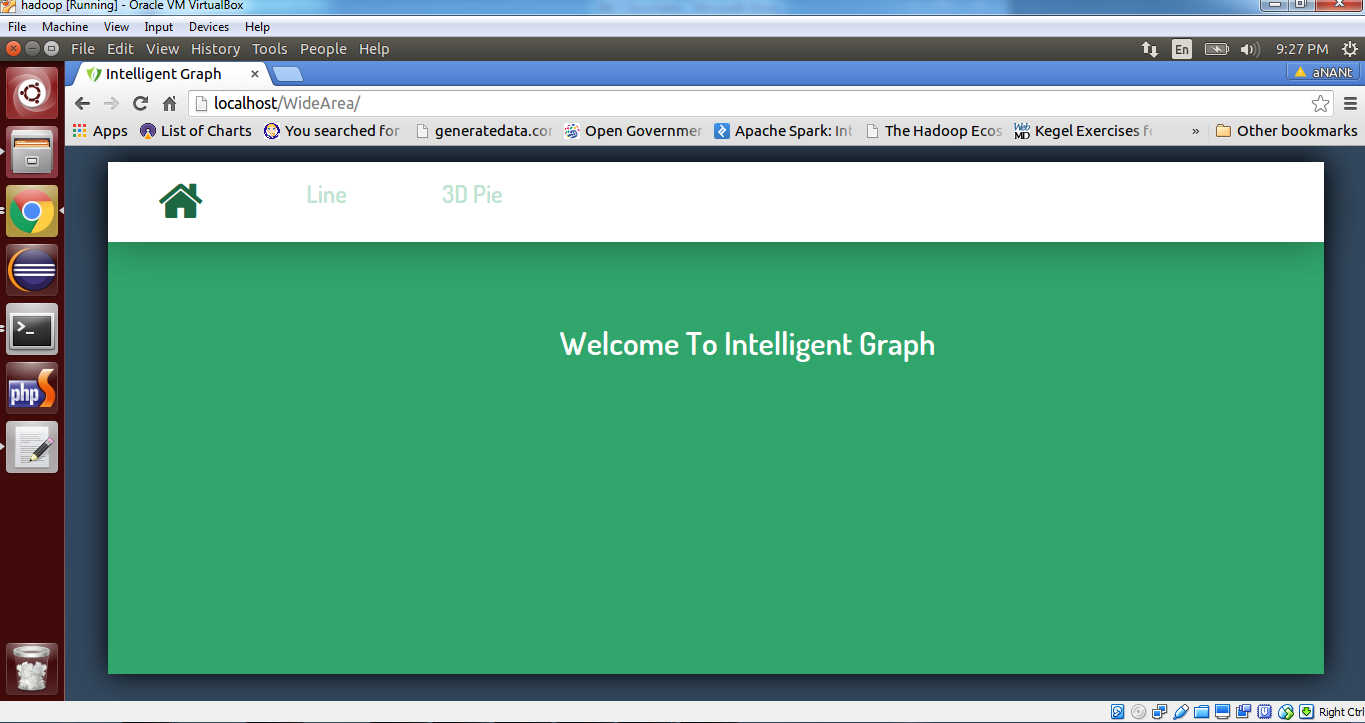


Figure 4.17: Home page Intelligent Graph

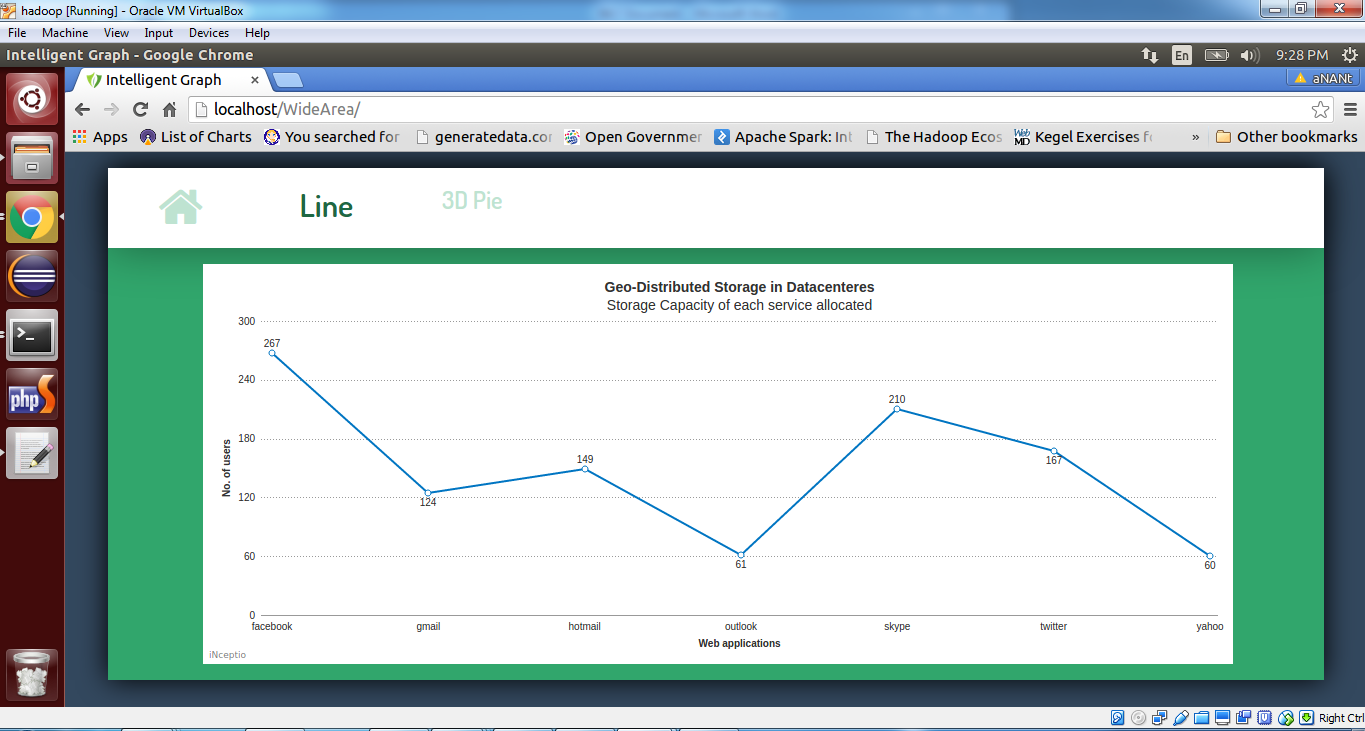


Figure 4.18: Represent the line Graph number of user available at each social network sites

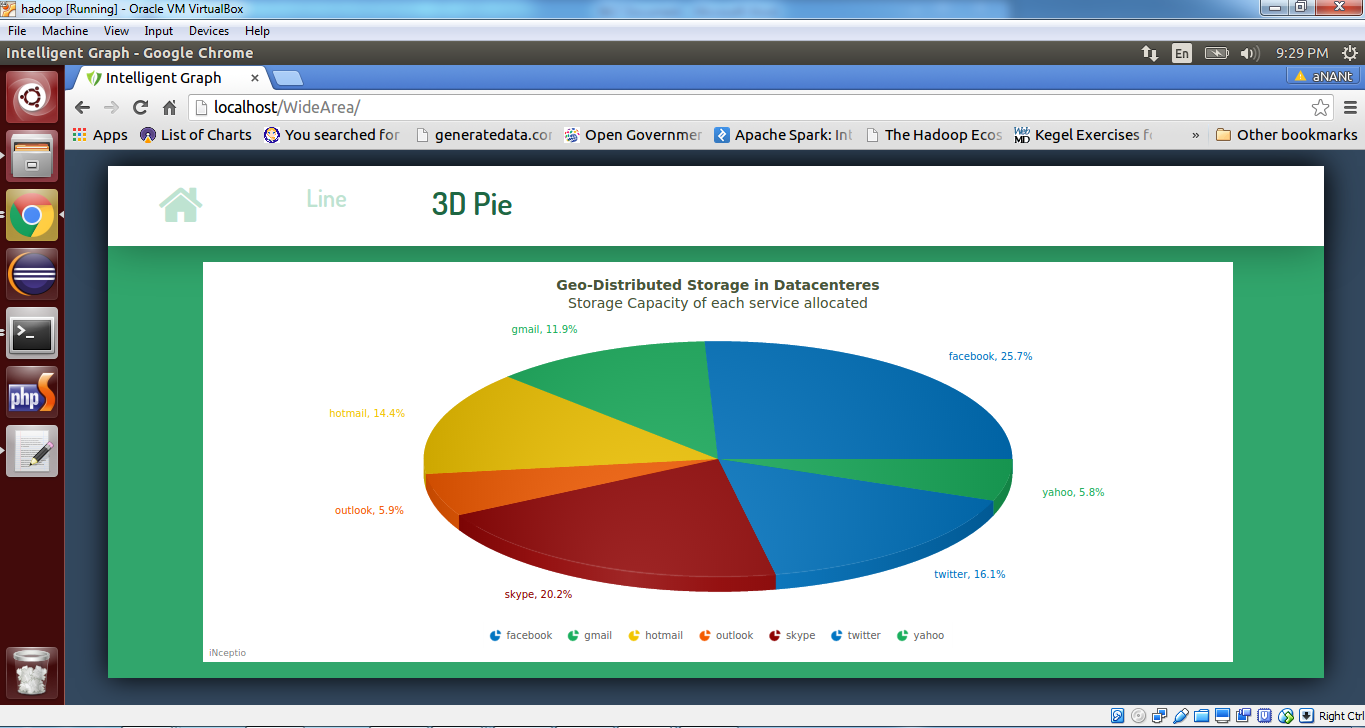


Figure 4.19: To analyse storage capacity of each service allocated

Then goto terminal end the hadoop process using

user@node:~$ stop-all.sh

### 5. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**TYPES OF TESTS**

**Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centred on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

**CONCLUSION&FUTURE SCOPE**

As data grows at a tremendous rate, achieving optimal performance in the wide area analytics becomes more and more challengeable. Compared with the local network in a data centre, the WAN covers a relatively broad geographical area, which is more complicated and unstable. Moreover, processing a substantial amount of data within a very small time interval is a great challenge for those low latency cloud applications. In this paper, we present a number of typical mechanisms in the wide area analytics, discuss high-level ideas, and give a comparison of these mechanisms. Although with some limitations, more effective solutions may be inspired by these mechanisms and applied in the real world in the near future.

**FUTURE SCOPE**

To extend my project to social network sites to ecommerce website. To analyse the storage capacity of various ecommerce website located distributed. Compare the various ecommerce website. To use efficient and most relevant website. To achieve optimal performance on various ecommerce website located distributed. Processing a substantial amount of data within a very small time interval is a great challenge for those low latency cloud applications.

**REFERENCES**

[1] K. Kloudas, M. Mamede, N. Preguic¸a, and R. Rodrigues, Pixida: Optimizing data parallel jobs in bandwidth-skewed environments, VLDB Endowment, vol. 9, no. 2, pp. 72–83, 2015.

[2] A. Vulimiri, C. Curino, P. Godfrey, T. Jungblut,J. Padhye, and G. Varghese, Global analytics in the face of bandwidth and regulatory constraints, in Proc. OfUSENIX Symposium on Networked Systems Design and Implementation (NSDI), 2015.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The Hadoop distributed file system, in Proc. of IEEE on MassStorage Systems and Technologies (MSST), 2010.

[4] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012.

[6] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, Discretized streams: Fault-tolerant streaming computation at scale, in Proc. of the 24th ACM Symposium on Operating Systems Principles (SOSP), 2013, pp. 423–438.

[7] R. Couto, S. Secci, M. Campista, and L. Costa, Latencyversus survivability in geo-distributed datacenter design, in Proc. of IEEE Global Communications Conference

(GLOBECOM), 2014, pp. 1102–1107.

[8] Q. Zhang, L. Liu, K. Lee, Y. Zhou, A. Singh, N. Mandagere, S. Gopisetty, and G. Alatorre, Improving hadoop service provisioning in a geographically distributed cloud, in Proc. of the 7th IEEE InternationalConference on Cloud Computing, 2014.

[9] A. Munir, I. A. Qazi, and B. Qaisar, On achieving lowlatency in data centers, in Proc. of IEEE InternationalConference on Communications (ICC), 2013, pp. 3721–3725.

[10] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, Aggregation and degradation in JetStream: Streaming analytics in the wide area, in Proc. of USENIX NSDI, 2014.

[11] P. Upadhyaya, Y. Kwon, and M. Balazinska, A latency and fault-tolerance optimizer for online parallel query plans, in Proc. of ACM SIGMOD International Conference on

Management of Data, 2011, pp. 241–252.

[12] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, WANalytics: Analytics for a geo-distributed data-intensive world, in Proc. of Conference on Innovative

Data Systems Research (CIDR), 2015.

[13] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, Low latency geodistributed data analytics, in Proc. of ACM SIGCOMM,

2015.

[14] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, Inter-datacenter bulk transfers with netstitcher, in Proc. of ACM SIGCOMM, 2011.

[15] L. Gu, D. Zeng, P. Li, and S. Guo, Cost minimization for big data processing in geo-distributed data centers, IEEE Trans. on Emerging Topics in Computing, vol. 2, no. 3, pp. 314–323, 2014

[16] C.-C. Hung, L. Golubchik, and M. Yu, Scheduling jobs across geo-distributed datacenter, in Proc. of the 6th ACM Symposium on Cloud Computing (SoCC), 2015.