

國立清華大學國際專業管理碩士班碩士論文

檢視使用 R 為統計服務核心之平行計算策略

**Review of Parallel Computation Strategies for Statistical
Service Engines Using R**



指導教授：雷松亞 博士-**Assistant Prof. Soumya Ray**

研究生：陶瑞 **Musa Touray**

學號：**100077427**

中華民國一〇二年七月

Review of Parallel Computation Strategies for Statistical Service Engines Using R

檢視使用 R 為統計服務核心之平行計算策略

Submitted to the School of Technology Management

Department of International Masters in Business Administration

Partial Fulfillment of the requirements for the degree of

Masters of Business Administration



Advisor: Assistant Prof. Soumya Ray (雷松亞)
College of Technology Management
Institute of Service Science

Second Reader: Prof Fu-Ren Lin (林福仁)
College of Technology Management
Institute of Service Science

Third Reader: Prof Jyun-Cheng Wang (王俊程)
College of Technology Management
Institute of Service Science

National Tsing Hua University - NTHU
College of Technology Management

July 2nd, 2013



ABSTRACT

In enterprise environment, the source data are stored in various forms such as files, database, and streaming data. Currently, analysts conduct data analysis in offline mode using statistical software [5]. In a conventional sequential computer, processing is channeled through one physical location. In a parallel machine, processing can occur simultaneously at many locations and consequently many more computational operations per second should be achievable. Due to the rapidly decreasing cost of processing, memory, and communication, it has appeared inevitable for at least two decades that parallel machines will eventually displace sequential ones in computationally demanding fields [9]. Many modern enterprises are collecting data at the most detailed level possible, creating data repositories ranging from terabytes to petabytes in size. The ability to apply sophisticated statistical analysis methods to this data is becoming essential for marketplace competitiveness. This need to perform deep analysis over huge data repositories creates a significant challenge to existing statistical software and data management systems. On the one hand, statistical software provides rich functionality for data analysis and modeling, but can handle only limited amounts of data; e.g., popular packages like R and SPSS operate entirely in main memory. On the other hand, data intensive management systems—such as MapReduce-based systems—can scale to petabytes of data, but provide insufficient analytical functionality. [1]

We are reviewing the statistical model in Lee's paper [5] which runs in sequential mode executing data given to it by the Application Server. We use Hadoop/MapReduce model as our statistical engine at the back end of our architecture data analysis algorithms (statistical service engine solution) which include two parts; one half is the R statistical analysis system

and the other half is the implementation of the Hadoop data management system. This model consists of three components: an R driver process operated by the data analyst, a Hadoop cluster that hosts the data and runs Jaql (and possibly also some R sub-processes), and an R-Jaql bridge that connects these two components [1]. This is to improve the performance of the scalability and the functionality of the statistical jobs sent to it in a cluster or distributed environment. Also, we use the approach of Message Passing Interface (MPI) and Parallel DBMS Computation to support our model of parallel computation. Thus the new system architecture of statistical service engine solution of Lee's paper is built.



ACKNOWLEDGEMENT

Ya-Allah, I want to take a minute, not to ask for anything from you, but simply to say “Thank You!” for all I have. Am thankful for giving me the strength to go through the accomplishment of completion of this vital challenge in my education career; my deepest gratitude to my lovely parents Mr.Bambo Touray and Mrs.Tida Bayo, siblings, family and friends for the infinite amount of support and guidance they have given me throughout these two years in Taiwan.

I would also like to express my sincere gratitude to my supervisor Prof. Soumya Ray for all his advice, the useful comments, remarks, engagement and help throughout this master thesis. He has tirelessly helped me with everything from setting and modification of the systems or approaches used in my paper as well as revising my project. This project would not have been possible without his guidance and persistent help. I would further extend my gratitude to Prof. Fu-Ren Lin and Prof Jyun-Cheng Wang for being my second and third reader respectively and help revising this paper. Not forgetting Rich C.Lee, senior IT Specialist, System & Technology Group, at IBM Taiwan Corporation for sharing his taught on his paper and giving me some guidance to follow.

I thank the Taiwan ICDF for giving me this opportunity, a full-time scholarship to pursue my master’s degree in a well reputable university, NTHU. Being student/alumni at National Tsing Hua University gives me a great joy to share my knowledge with different students from different backgrounds, the cultural difference and fun shared and so on which forever will stick with me. Moreover, deep appreciations to NTHU for providing world class infrastructure and a favorable healthy environment for a better study and research, the lovely and cooperative staff

of IMBA department as well as to all my professors who enlighten me with their advices and experiences, Thank You!

Finally, I would like to thank my loved ones who have supported me throughout the entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love.

"No duty is more urgent than that of returning thanks." - St. Ambrose



Table of Contents

CHAPTER 1: INTRODUCTION	11
CHAPTER 2: Literature Review	15
2.1: Lee's Statistical service engine solution.....	15
2.2: R-Statistical Engine.....	16
2.3: motivation	20
CHAPTER 3: PARALLEL DBMS Approach of Parallel Computation	22
3.1: Different Architectures of Parallel DBMS	22
1. Shared - memory system:	22
2. Shared - disk system:.....	24
3. Shared - nothing system:	25
3.2: Types of Parallel DBMS.....	26
a) Pipeline Parallelism	26
b) Partition Parallelism	27
3.3: Some Major Terminology of Parallel DBMS	28
• Linear Speed-UP	29
• Linear Scale-Up	29
3.4: Advantages and Disadvantages of Parallel DBMS plus SQL Sample Code	31
a) Advantages:.....	31
b) Disadvantages	32
Sample Codes and Example of Parallel DBMS	33
• Selection / projection / aggregation	33
• Sorting.....	33
3.5: The Architecture Overview of Parallel DBMS on Statistical Service Engine Solution.....	33
CHAPTER 4: The Approach of Hadoop-Bridge with A High Query Language	36
4.1: Jaql query language	36
4.2: Hadoop Data Management Systems.....	38
4.2.1 some advantages of Hadoop/map-reduce.....	39
4.2.2 some disadvantages or limitations of Hadoop/map-reduce	41
4.2.3 an example of Hadoop/map-reduce plus sample code.....	41
4.3: The Architecture Overview of Hadoop-Bridge Statistical Service Engine Solution	43
CHAPTER 5: Message Passing Interface (MPI) Approach of Parallel Computation	46
5.1: Programming Model	47
a) Shared-memory system	48
b) Distributed-memory system	49

c) Hybrid Distributed-shared memory system	50
5.2: <i>Operations for Communications</i>	51
a) Point-to-Point Operations	51
b) Collective Communication.....	53
5.3: <i>Sample design and Examples</i>	53
a) Example	53
b) The simple Architecture of MPI	54
5.4: <i>Advantages and Disadvantages of Message Passing Interface (MPI)</i>	55
a) Advantages of MPI	55
b) Disadvantages of MPI.....	56
5.5: <i>The Architecture Overview of MPI on Statistical Service Engine Solution</i>	56
CHAPTER 6: COMPARISON OF THE THREE MODELS	59
6.1: <i>Scalability:</i>	60
6.2: <i>Ease of Writing code and easy understanding- programming model</i>	61
6.3: <i>Flexibility</i>	63
6.4: <i>Performance and Efficiency</i>	64
6.5: <i>Cost</i>	66
6.6: <i>Fault Tolerance</i>	66
6.7: <i>Brief Comparison showing in Tabular Form</i>	68
CHAPTER 7: CONCLUSION	71
REFERENCES.....	74

Lists of Tables

Table 1: basic functions and descriptions of ODBC [16]	17
Table 2: shows how R gets access to database on ODBC [16].....	18
Table 3: Sample name of people and their DOB.....	19
Table 4: R accessing data from the local disk	19
Table 5: R accessing data on the web	20



Lists of Figures

Figure 1: Source:- statistical service engine solution [5]	16
Figure 2:- shared memory Architecture.....	23
Figure 3: parallel database systems	24
Figure 4:- shared-disk Architecture	25
Figure 5:- shared nothing Architecture.....	26
Figure 6:- Diagram of Pipeline and Partitioned Parallelism	28
Figure 7:- Diagram of Linear Speed-up and Scale-up.....	Error! Bookmark not defined.
Figure 8: Parallel DBMS Approach to the statistical Service Engine Solution.....	34
Figure 9: Map-Reduce Data Flow [19].....	38
Figure 10: Hadoop/Map-Reduce Approach to the statistical Service Engine Solution	44
Figure 11: A shared-memory system	48
Figure 12: A distributed-memory system	49
Figure 13: A hybrid distributed-shared memory system	50
Figure 14: shows the simple send and received	51
Figure 15: Path of a message buffered at the receiving process.....	52
Figure 16: shows processes within a communicator	53
Figure 17: Master/Slave MPI Architecture Diagram	54
Figure 18: MPI Approach to the statistical Service Engine Solution	57

"I know for sure that what we dwell on is who we become."

Author: Oprah Winfrey

Interpretation: Our thoughts determine our reality.

CHAPTER 1: INTRODUCTION

Making good decision is a major aspect of any good industry or company to make an outstanding performance and strive forward in a competitive market in today's world. Decision making and analyzing data is not an easy task to do for any organizations which really needs a lot of concern. However, in an enterprise, decision has to be taken to improve the aim and objective of the firm which is generally the responsibility of any unit manager in an organization. A good statistician or data analyst is needed to do such an outstanding job. Doing this is quite easy nowadays but serious knowledge is required to grasp the techniques and there implementation; there are some Business Intelligence (BI) tools or software that can help in making good decision making, for instances Data warehousing and on-line analytical processing (OLAP) and so on.

As companies are competing in a well distributed computing environment, there tend to be more and more data that the companies collect to be analyzed. Lee's [5] identified architecture of how Statistical Service engine solution can be more useful in executing and analyzing data in a particular industries using R-Engine as the statistical package at the back-end of the model. This model [5] runs in sequential mode executing data given to it by the Application Server. Though, the model can be applied in any industries or company that wants to use it to make business decision analysis. Thus, the motivation of reviewing Lee's paper [5] is essential wondering if managers in big company's wants to implement this model in their various unit, it might take them a large amount of time for execution of programs; since is running in serial mode with R-engine which only execute structural data. Thinking about the scalability (a big

data set with the company), performance wise, Fault-tolerance, dealing with data of any format (structured or unstructured), time saving and so on for the company's which the model [5] is not capable of, this might/will be a huge drawback for industries with big data set; then we suggested that applying this [5] Statistical model but changing the back end engine to Parallel computation model could be an easy way for achieving the company's objectives. Since parallel computation approach enabled us to execute a huge data set in parallel in a distributed environment, then we also suggested applying three kind of approaches or strategies to execute our data in parallel using Parallel DBMS, Hadoop/MapReduce and Message Passing Interface for the reason that they are broadly methods use in most of the industries and computing network like Google, Facebook, etc. However, in any good industries, the managers main objectives is to take a good decision that will burst up the company's goals and target mission as well as performance issues.

Many of today's enterprises collect data at the most detailed level possible, thereby creating data repositories ranging from terabytes to petabytes in size. Today's good decisions are driven by reliable data. Business managers and professionals are increasingly required to justify decisions on the basis of data. They need statistical model-based decision support systems. [8]

This paper is about the further proposal of the expansion or extension of "A Novel Data Analysis Service Oriented Architecture Using R (2012)" [5]. The paper demonstrates how service-oriented statistics engine was developed and how parallel computation can be applied in the R-statistical engine at the back end of the architectural design. Thus, Lee's paper shows the sequential computation and approach on the statistical service engine solution with R-engine at the back of the program which operates on a single server and basically sit on the

main memory executing statistical jobs sent to it through Enterprise Service Bus which serves as Message Oriented Middleware between the statistical job portal and the R-engine blades; The limitation here is that R-engine alone is not capable of handling/executing a huge data-set (e.g petabytes), it only works in serial mode and in addition can only be able to execute the structural data; the model is not fully suitable for a large distributed network which involves a large data-set of any format to be executed without failure of the work, performance as well as operational speed. Dealing with the execution of a large data set and handling the data in any format, be structured or unstructured as well as executing them in parallel; the purpose of this study is to review and compare the literature on three known architectures used to do statistical calculations in parallel. These three methods will be analyzed in parallel computation mode whereby statistical jobs will be executed simultaneously at the same time in a well distributed network or computer clusters. In the case whereby a user or manager of an organization is concern about handling the big data of their organization, let say the data becomes too large (terabytes, petabytes, etc) which includes any data format, the performance of the computation but most importantly he/she is concerned about the continual execution of the operation even if one or two machines in the distributed network fails i.e. the Fault Tolerance is among the organization top priority list; then one of the main approaches about this paper in statistical engine model ***“The Approach of Hadoop-Bridge with A High Query Language ”*** will be an appropriate approach for the manager and his organization to implement; thus the model provides superior performance, scalability, elasticity and fault-tolerance properties on large clusters of service machines. In scenario, where the manager or the company goals are to hold and execute their big data-set (structured or unstructured) in a distributed computer network that required a lot

of inter-process communication, the need for low latency reductions, performance and for short task processing but rather fault-tolerance is not their main concern, then the approach of **“Message Passing Interface (MPI) approach to the statistical model”** can be recommended to them; thus MPI is good for task parallelism. The last approach of this paper which is **“The approach of Parallel DBMS Architecture”** is basically good for an organization where the manager’s main concern is the scalability, performance, and does require scheme support/define which required data to fit into the relational paradigm of rows and columns as well as proper indexing for the acceleration of the speed to search through the data-set in a distributed computing.

Thus, this will ease the computational curves when having huge dataset to analyze with any format of being structured or unstructured and looking for different dimensions using R-package to improve the performance and functionality of the systems.

In parallel to the development of statistical software packages, the database community has developed a variety of large-scale data management systems (DMSs) that can handle huge amounts of data. Examples include traditional enterprise data warehouses and newer systems based on Map-Reduce [11], such as Hadoop. The data is queried using high-level declarative languages such as SQL, Jaql, Pig, or Hive [12]

The remaining section of the paper, is divided into following parts; in chapter (2) Literature Review (3) Parallel DBMS approaches to the Architecture of statistical service engine solution (4) The Approach of Hadoop with a High Query Language (5) MPI approach to the statistical model (6) Comparison of the three approaches (7) Future Work and Conclusion

CHAPTER 2: Literature Review

In this chapter, we look at the brief introduction of the model we developing on and the statistical engine using with it which we divide into three sections as described below: a) Lee's statistical service engine model in brief b) the R-statistical service and c) Motivation

2.1: Lee's Statistical service engine solution

The following model executes the statically jobs in a sequential mode. The Statistical Job Portal server provides a number of predefined statistical procedures and ad-hoc analysis to users. The Enterprise Service Bus server which serves as an intermediary between the Statistical Job Portal and the R-engine, receives messages (statistical job requests) sent to it by the Statistical Job Portal server and transmits them consequently through the Message Queue (the subscribers) which prioritizes the messages it receives and forward it to the R-engine. GNU-R Engine is a set of Blade servers executing GNU-R scripts designated by the messages. The script may retrieve data from the Database servers or the File Repository server. The Application servers are user process engines populating source data on the Database or the File Repository server. The File Repository server also stores the statistical results expecting users to retrieve later. The statistical job request is in XML format which is well structured for the R-engine to run on visualization. [5]

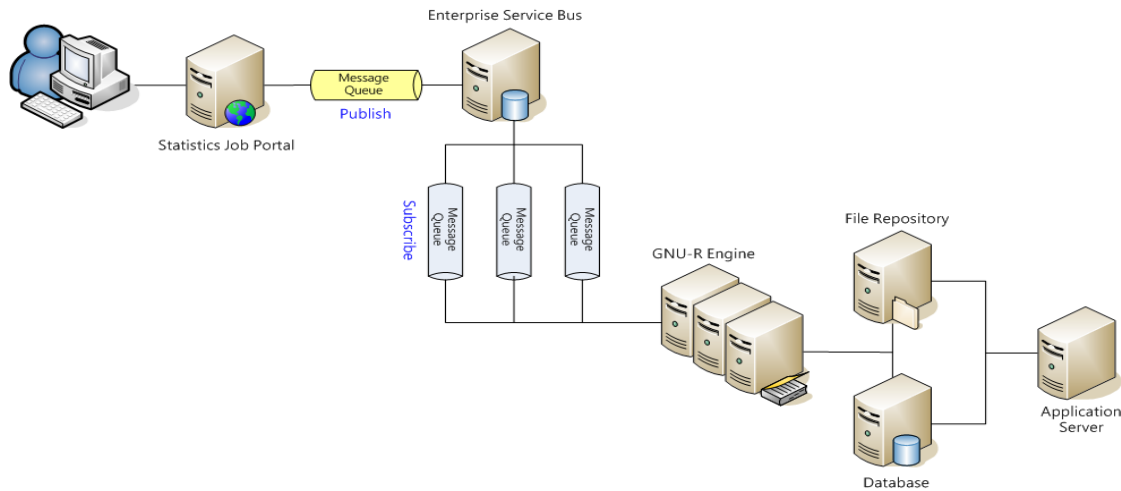


Figure 1: Source-: statistical service engine solution [5]

2.2: R-Statistical Engine

R environment for complete statistical computing and visualization is an open-source language and data analysis tool of the S statistical computing language with lot of packages. As open source software, R is completely free and can be freely distributed and it has the capability to run on most or all of the computing platforms which contains a lot of contributions from top or most of the computational statisticians or analyst. R is well recognized both by the users and the developers and is regularly updated with good user-friendly statistical computing texts [R.Ihaka & R.Gentleman [13]].

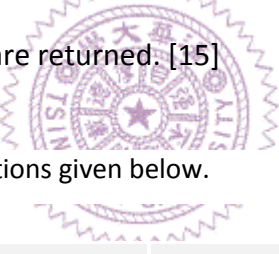
R was originally developed by Ross Ihaka and Robert Gentleman in 1993, who were at that time working at the statistics department of the University of Auckland, New Zealand. R provides both an open-source language and an interactive environment for statistical computation and graphics. The core of R is still maintained by a relatively small set of individuals, but R's gigantic popularity derives from the thousands of sophisticated add-on packages developed by hundreds of statistical experts and available through CRAN. Large enterprises such as AT&T and

Google have been supporting R, and companies such as REvolution Computing sell versions of R for commercial environments. [2]

“The r-base-core is a universe/math package described as a GNU R core of statistical computation and graphics system. GNU R script can access databases by r-cran-rodbc package through Open Database Connectivity (ODBC). The package should be platform independent and provide access to any database for which a driver exists” [5].

ODBC is a standard application programming interface (API) that allows an application to access data on a relational or non-relational database. Through the use of ODBC drivers, requests are submitted from the application to the database and results are returned to the application. The driver accepts queries written in Structured Query Language (SQL) and translates them to the database so that the desired results are returned. [15]

The primary functions of RODBC connections given below.



Function	Description
<code>odbcConnect(dsn, uid="", pwd="")</code>	Open a connection to an ODBC database
<code>sqlFetch(channel, sqtable)</code>	Read a table from an ODBC database into a data frame
<code>sqlQuery(channel, query)</code>	Submit a query to an ODBC database and return the results
<code>sqlSave(channel, mydf, tablename = sqtable, append = FALSE)</code>	Write or update (append=True) a data frame to a table in the ODBC database
<code>sqlDrop(channel, sqtable)</code>	Remove a table from the ODBC database
<code>close(channel)</code>	Close the connection

Table 1: basic functions and descriptions of ODBC [16]

```
# RODBExample
```

```
# import 2 tables (Crime and Punishment) from a DBMS
```

```
# into R data frames (and call them crimedat and pundat)
```

```
library(RODBC)
```

```
myconn <- odbcConnect("mydsn", uid="Rob", pwd="aardvark")
```

```
crimedat <- sqlFetch(myconn, Crime)
```

```
pundat <- sqlQuery(myconn, "select * from Punishment")
```

```
close(myconn)
```

Table 2: shows how R gets access to database on ODBC [16]

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- Graphical facilities for data analysis and display either directly at the computer or on hardcopy. [14]

Suppose, we have the following data set given below, which is saved as “students.txt” in the local **disk D:**

Table 3: Sample name of people and their DOB

Name	Bday	Sex
Beckham	19-Aug	Male
Alice	8-Oct	Female
Muhammad	12-Dec	Male
Jackie	24-Mar	Female
Solskjaer	6-Mar	Male

The above Table 3: shows how R reads in the data from the file students.txt on a disk in drive d:

Table 4: R accessing data from the local disk

```
> results<- read.table("d:/students.txt", header=TRUE)
```

The <- is a left diamond bracket (<) followed by a minus sign (-). It means "is assigned to". Use of header=TRUE causes R to use the first line to get header information for the columns. If column headings are not included in the file, the argument can be omitted.

Now type in results at the command line prompt, displaying the object on the screen:

```
> results
```

Name	Bday	Sex
Beckham	19-Aug	Male
Alice	8-Oct	Female
Muhammad	12-Dec	Male
Jackie	24-Mar	Female
Solskjaer	6-Mar	Male

However, accessing data from the web should be in a simple format for R statistical computation to read it from.

Data store in the web or the HTTP output format should be in a text/plain format with tab delimiter for data columns...

Table 5: R accessing data on the web

```
results <- read.table("http://data.princeton.edu/wws509/datasets/effort.dat")
```

2.3: motivation

The motivation of reviewing Lee's paper [5] is essential wondering if managers in big company's wants to implement this model in their various unit, it might take them a large amount of time for execution of programs; since is running in serial mode with R-engine which only execute structural data. Thinking about the scalability (a big data set with the company), performance wise, Fault-tolerance, dealing with data of any format (structured or unstructured), time saving and so on for the company's which [5] is not capable of, it might/will be a huge drawback for industries with big data set; then we suggested that applying this [5] Statistical model but changing the back end engine to Parallel computation model could be an easy way for achieving this objectives. Since parallel computation approach enabled us to execute a huge data set in parallel in a distributed environment, and then we also suggested applying three kinds of approaches or strategies to execute our data in parallel for couple of reasons:

Parallel DBMS – for the reason of high performance and high availability; DBMS seeks to improve performance through parallelization of various operations, such as data loading, index building and query evaluating. Even though data may be stored in a distributed manner in such a system, performance considerations is the sole purpose.

The second approach being the Hadoop/MapReduce- which is an open source, also good for large problems and handling a large data set (scalability) and fault-tolerant, parallel computing is much more cost effective and/or feasible in a distributed computing network. E.g. a big enterprise, Facebook has implemented a large dataware house system using MapReduce system.

Third approach is the Message Passing Interface- the need for inter-process communication, performance, task parallelism and for short task processing; also for a portability (the ability of the same source code to be compiled and run on different parallel machines), efficient, and flexibility.

The reason of using these approaches is that they are broadly methods use in most of the industries and computing network like Google, Facebook, etc. shows some flexibility that the users using the approaches on the statistical service engine solution discussed on this paper might not have trouble executing the program since the approaches are already familiarized.

In the subsequent sections, we look at the following point: Sec.3) Parallel DBMS model, Sec.4) Hadoop Statistical Service Engine Model and Sec.5) Message Passing Interface approaches.

CHAPTER 3: PARALLEL DBMS Approach of Parallel Computation

A parallel database management system is one that tries to find, and purposely to improve performance through parallel implementation of various operations such as loading data, building indexes, and evaluating queries.

The conceptual idea or thought about parallel DBMS is that the data might/are stored in different machines which are then kept physically in different locations but closed enough, i.e. the physical machines that hold the data are physically close to each other for instances in the same server room but its communication and accessibility of the machines is through the connection of high speed switches or local area network which is transparent to the user. A typical example can be a server room in banks or telecommunication industries that have different server machines that kept the companies data and executing them in parallel. Data may be distributed but the sole purpose is for performance reasons to shorten the response time and provide good loading of data among the processors when data is queried. [26]

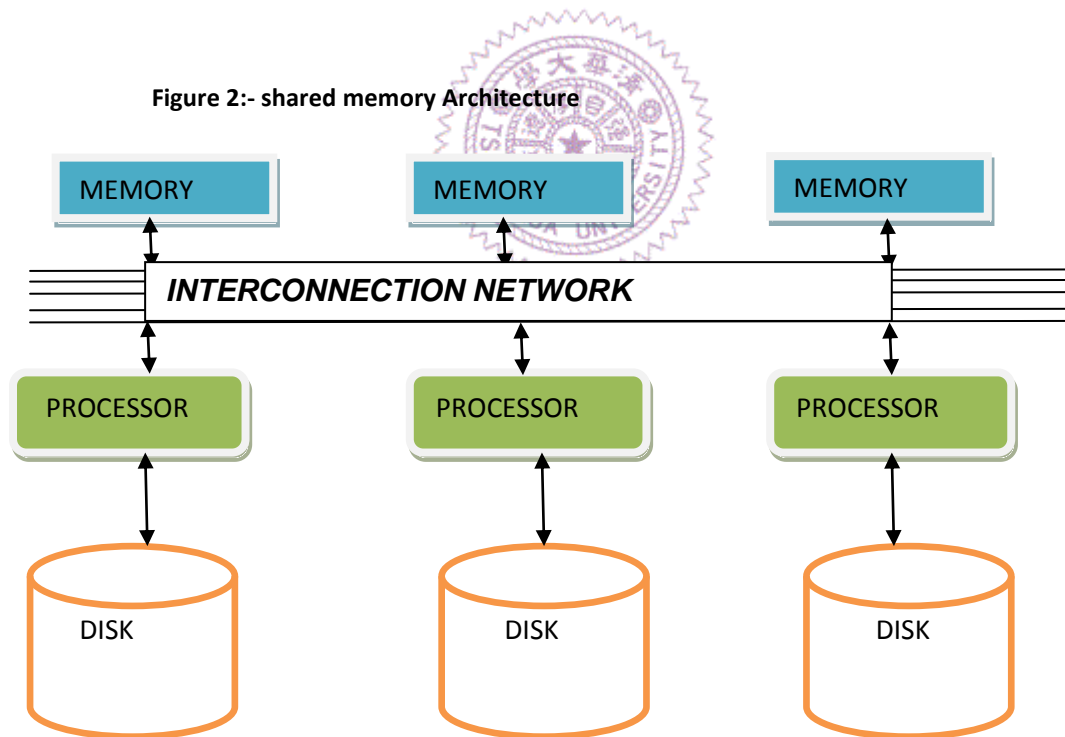
3.1: Different Architectures of Parallel DBMS

Parallel DBMS has three main architectures that are planned for building parallel databases systems in a distributed computing network.

1. **Shared - memory system:** this is one of the main systems of parallel DBMS where multiple processors/CPU's are attached to an interconnection network and can access a common region of main memory. One of the main advantages of this type of system is load balancing of data and its simplicity of processors sharing the same main memory;

where information can be passed from one processor to another easily. Likewise, this type of system is costly to implement that because reading or writing to far memory can be slightly more expensive and has low availability of handling failures of hardware components since each CPU shared the memory (fault tolerance). [26]

Examples of shared-memory parallel database systems include XPRS, DBS3, and Volcano, as well as portings of major RDBMSs on shared memory multiprocessors. In a sense, the implementation of DB2 on an IBM3090 with 6 processors was the first example. All the shared-memory commercial products (e.g., INGRES and ORACLE) today exploit inter-query parallelism only (i.e., no intra-query parallelism) [27].



Here, every processor has its own disk and sharing the main memory through the interconnection network. Memories can be one or more, not necessary the same amount of disk in the architecture.

Also, all processors shared common global memory and access to all disks, that is any CPU has the access to any memory component or disk unit as in the case of figure below: more processors and one main memory.

Figure 3: parallel database systems

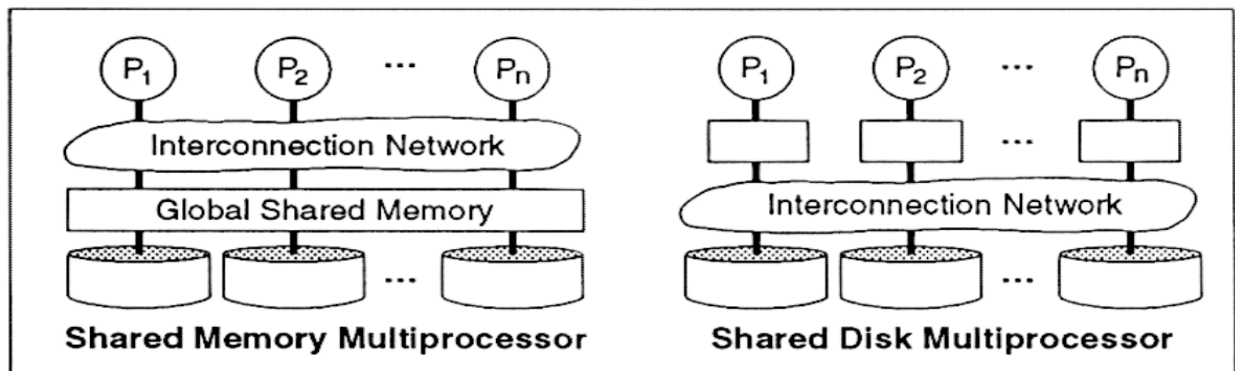


Figure 3: Source: David Dewitt and Jim Gray, "Parallel Database Systems: The Future of High Performance Database Systems"

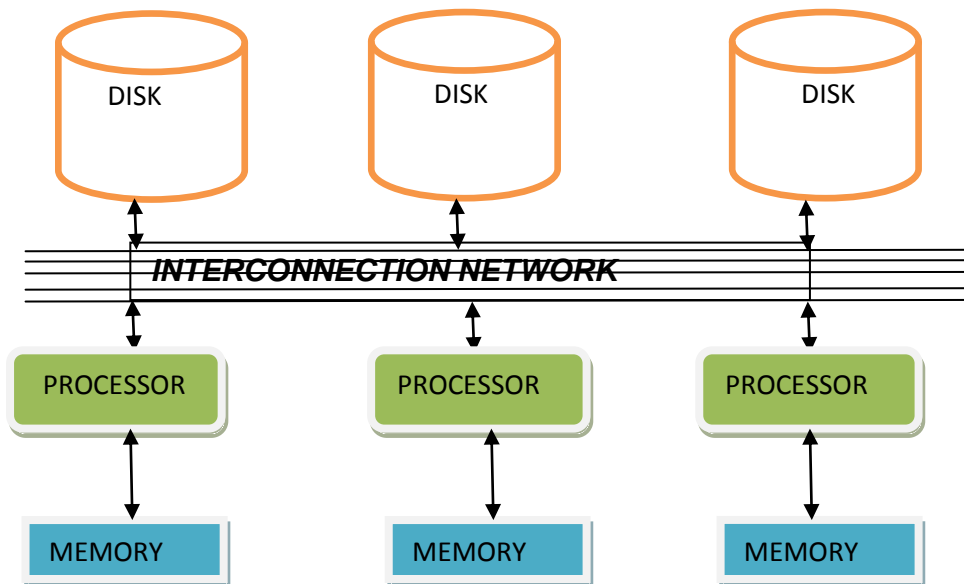
2. Shared - disk system:

This is the system where each CPU has a private memory and direct access to all disks through an interconnection network. As well the number of disk doesn't necessary have to match with the number of memory. Some advantages of this system is having the potential of adding more processing and storage power as well as ability to fault tolerance. [[Özsun,Valduries[2011]]]

"Examples of shared-disk parallel DBMS include IBM's IMS/VS Data Sharing product and DEC's VAX DBMS and Rdb products. The implementation of ORACLE on DEC's VAXcluster and NCUBE computers also uses the shared-disk approach since it requires minimal extensions of the RDBMS kernel. Note that all these systems exploit inter-query parallelism only" [27]

The diagram below shows the description

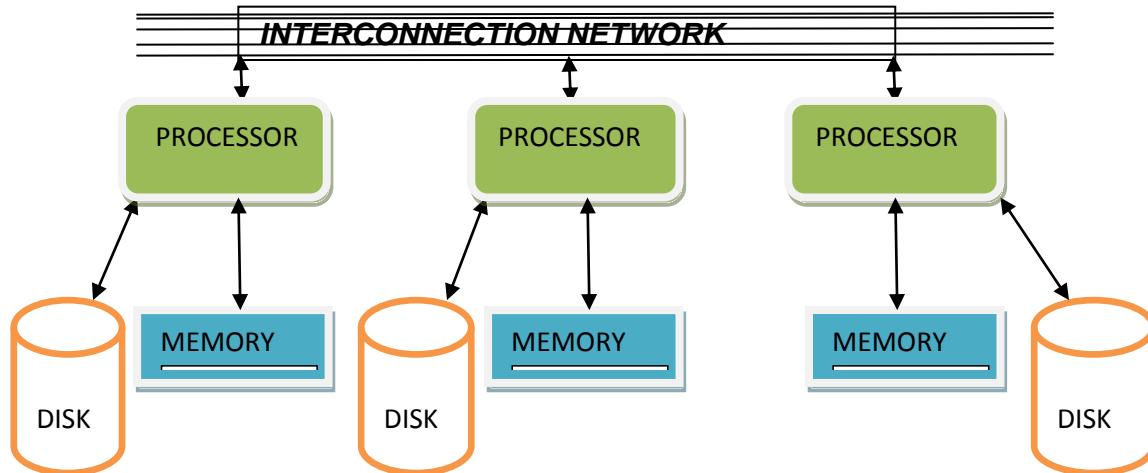
Figure 4:- shared-disk Architecture



3. **Shared - nothing system:** this type of parallel DBMS doesn't involve any sharing of disk or memory. Is a situation where each CPU has local main memory and disk space, but no two CPUs can access the same storage area; all communication between CPUs is through a network connection. Thus, the communication of processors is done through high speed network and switches. [Özsun,Valduries[2011]]

Examples of shared-nothing parallel database systems include the Teradata's DBC and Tandem's Non- StopSQL products as well as a number of prototypes such as BUBBA , EDS , GAMMA , GRACE ,PRISMA , and ARBRE [27].

Figure 5:- shared nothing Architecture



Scaling the system is an issue with shared memory and shared disk architectures because as more CPUs are added, existing CPUs are slowed down because of the increased conflict for memory accesses and network bandwidth.

3.2: Types of Parallel DBMS

“Relational queries are ideally suited to parallel execution; they consist of uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graph” (*Dewitt and Gray*) [26]. However, there are two major types of parallel DBMS that handle these operational activities.

a) Pipeline Parallelism

A graph of relational algebra operators (e.g. select operator), and the operators in a graph can be executed in parallel. If an operator consumes the output of a second operator, we have

pipelined parallelism. In the case of having two operators, let say A and B; streaming the output of one operator A into the input of another operator B, the two operators can work in series giving pipeline parallelism. Here, we have many machines each doing one step in a multi-step process. For instance, the ordered task is distributed to different machines each performing different tasks. E.g. if we have ordered task to be executed and five different machine to use, then each of the 5 different machines will be performing different task in serial fashion. The first task has to be complete before the second task can start to process the data and so on [26],[27],[28].

b) Partition Parallelism

This is another approach of parallel DBMS where each individual operator can also be executed in parallel by partitioning the input data and then working on each partition in parallel and then combining the result of each partition. Here, many machines are doing the same thing to different pieces of data. So, in this case the task is divided over all machines to run in parallel. E.g. let's say, after an exam, student's scripts are divided to different lecturers to mark and access their grades simultaneously. Thus, Pipeline parallelism is when multiple steps depend on each other, but the execution can overlap and the output of one step is streamed as input to the next step. The purpose of pipeline parallelism is to increase the speed of your program and decrease your I/O operations Thus, task parallelism....[26][27][28]

a) Pipeline Parallelism



b) Partition Parallelism



Figure 6:- Diagram of Pipeline and Partitioned Parallelism

(Source: from Joe Hellerstein; and Jim Gray, Microsoft Research ppt)

3.3: Some Major Terminology of Parallel DBMS

Improving consistency is one of the main factors that Parallel DBMSs are intended to solve i.e. fault tolerance. The breakdown of a single processor(s), or the communication link failure between different processors which makes one or more processors inaccessible, is not enough to bring down the entire system. This means that although some of the data may be difficult to get to, with proper system design users may be permitted to access other parts of the database in a distributed computing network. In Parallel DBMS, there is ability for user to increase the sizes of the database or to increase the performance demands. The two main terminologies: [27][26].

Linear Speed-Up

This is the situation where the user increases twice the hardware/processors in the database and keeps the main memory/disk size constant to perform the same task given to it in half the normal time. For example, if 10petabyte task or data takes 3minutes to be executed in 5 nodes; increasing processors can leads to linear increase in the processing and storage power which will enabled the hardware to perform the execution of 10petabyte task within 1.5minutes. “Linear speedup refers to a linear increase in performance for a constant database size, and a linear increase in processing and storage power”[27].

Linear Scale-Up

This is the condition where the user increases the main memory/disk size for a given task size to perform twice the usual time. For example, if 10petabyte or data takes 3minutes to be executed in 5 nodes; increasing the disk size can leads to linear increase in the processing and storage power which will enabled the hardware to perform the execution of twice 10petabyte task within 3minutes.”Linear scale-up refers to a continual performance for a linear increase in both database size, processing and storage power”. [27]

In detail, the speedup is calculated as:

“(Speedup = small_system_elapsed_time/big_system_elapsed_time) if for example a fixed job runs on a small syatem, and then run on a larger system. Moreover, scaleup is the ability of an N-times larger system to perform an N-times larger job in the same elapsed time as the original system.

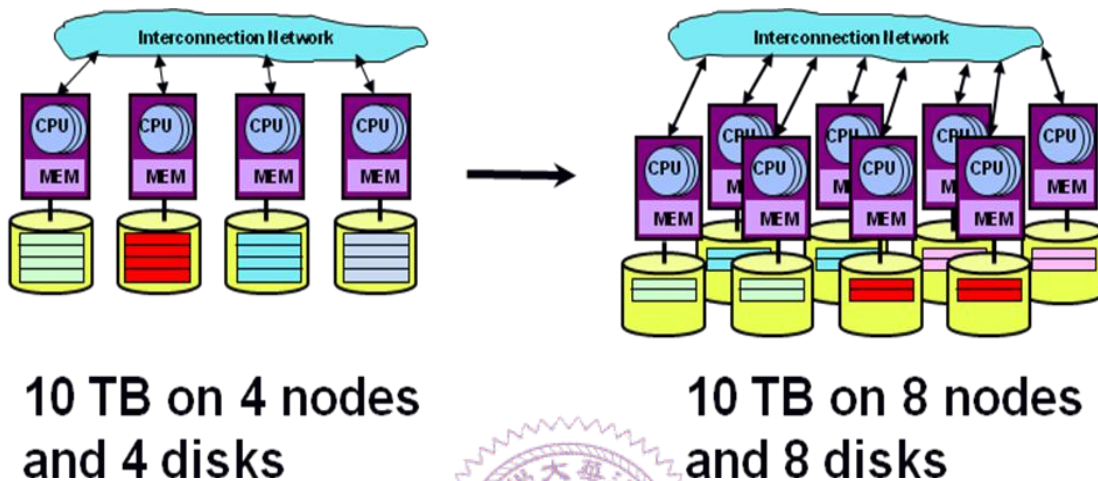
The scaleup is of the form

(Scaledup=

small_system_elapsed_time_on_small_problem/big_system_elapsed_time_on_big_problem)”

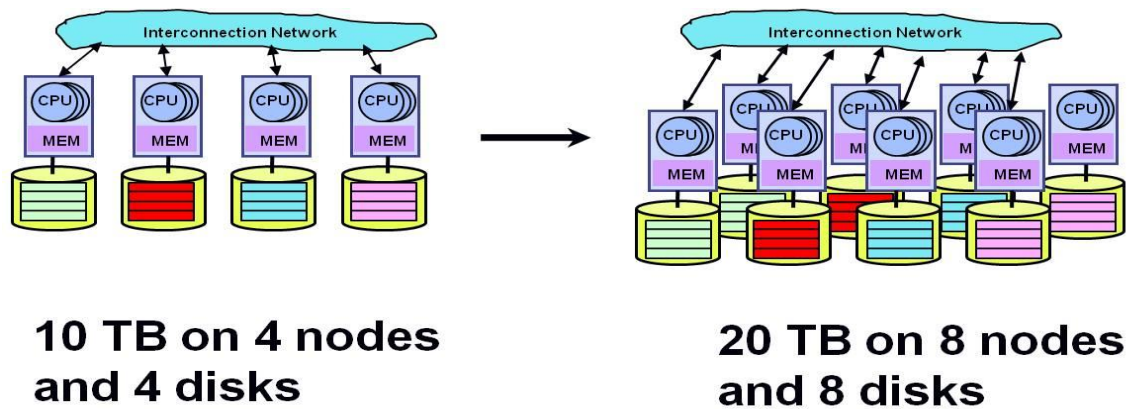
... [26]

a) Linear Speed-up



Here, keeping the workload constant execute $\frac{1}{2}$ the response time

b) Linear Scale-Up



Here, keeping the response time constant

Figure 7:- Diagram of Linear Speed-up and Scale-up

(Source: pages.cs.wisc.edu/~dewitt/includes/passtalks/pass2008.pptx - by Dewett and Gray)

“The generic barriers to linear speedup and linear scaleup are the triple threats of:

Startup: the time needed to start a parallel operation. If thousands of processes must be started, this can easily dominate the actual computation time.

Interference: the slowdown each new process imposes on all others when accessing shared resources.

Skew: as the number of parallel steps increases, the average size of each step decreases, but the variance will exceed the mean. The service time of a job is the service time of the slowest step of the job. When the variance dominates the mean, increased parallelism improves elapsed time only slightly.”[26]

Intra-operator parallelism, Inter-operator parallelism and Inter-query parallelism are other types of DBMS parallelism which can be available in details on (Dewitt and Gray [26]) and (Ozsu, Valduriez [27])

3.4: Advantages and Disadvantages of Parallel DBMS plus SQL Sample Code

Below are some of the pros and cons of ||DBMS

a) Advantages:

- Parallel DBMSs has the scheme defined using the relational data model. It requires data to fit into the relational paradigm of rows and columns which is good for querying data in the database. [Pavlo , Paulson, Rasin[2009]]

- All modern DBMSs use hash or B-tree indexes to accelerate access to data.

Most database systems also support multiple indexes per table. Thus, the query optimizer can decide which index to use for each query or whether to simply perform a brute-force sequential search. [Pavlo , Paulson, Rasin[2009]]

- Parallel DBMS improves response time since is fast in querying data in the database table by scheme define and built-in indexing techniques which speed up access to data inside of their application.
- It also improves the average rate of successful message delivery over an interconnection network between processors i.e. throughput
- It tolerate fault-tolerance- A parallel database, properly configured, can continue to work despite the failure of any computer in the cluster. The database server senses that a specific computer is not responding and reroutes its work to the remaining computers.

b) Disadvantages

- There is interference – contention/disagreement for memory access and bandwidth when each new process imposes on all others when accessing shared resources.
- Parallel DBMS may be costly to manage which involves multiple processors and servers in a well distributed system and managing such system simultaneously might become difficult.
- Since is not an open source, cost wise it is expensive

Sample Codes and Example of Parallel DBMS

Below are two examples showing the execution of SQL command

➤ Selection / projection / aggregation

SQL Query:

```
SELECT year, SUM(price)
FROM sales
WHERE area_code = "US"
GROUP BY year
```

➤ Sorting

```
SQL Query:
SELECT *
FROM sales
ORDER BY year
```



SOURCE: Data management in Cloud – Advance Topics in Databases (2011) by Saake/Schallehn (FIN/ITI)

3.5: The Architecture Overview of Parallel DBMS on Statistical Service Engine

Solution

Parallel DBMS architecture design showing in the diagram below is running in parallel whose server replaced the “database” in [5] architecture of the service engine solution. This architecture shows the parallel computation of the statistical jobs sent by the user and executed in parallel with the approach of Parallel DBMS. The model runs more or less the same

as Lee's [5] model of the architecture which runs in serial mode with R-engine retrieving data from database or file repository at its back-end; the extended version of [5] replaced the "database" to "Parallel DBMS server" executing the jobs in parallel.

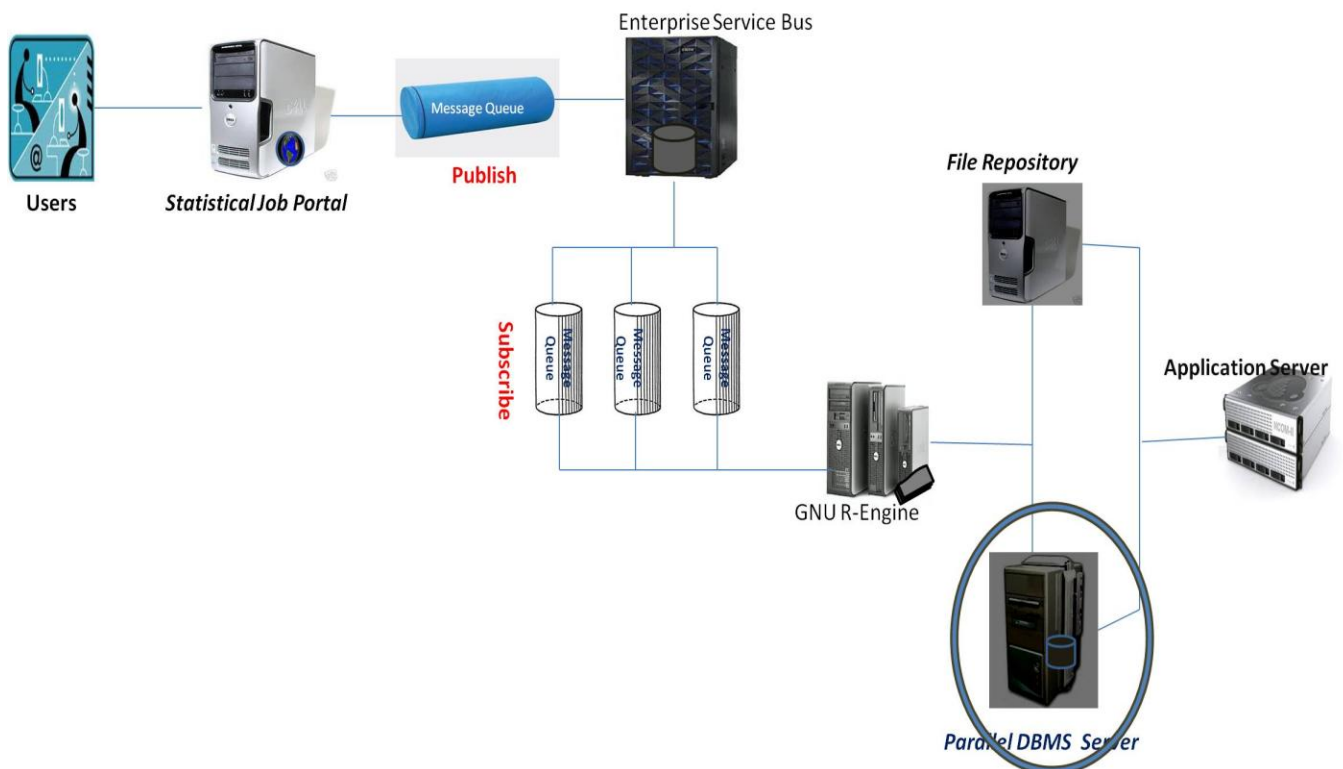


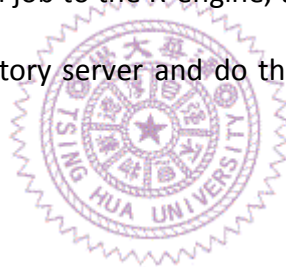
Figure 8: Parallel DBMS Approach to the statistical Service Engine Solution

Referring to the "Chapter 2.1" explained [5] the serial mode of the architecture of how user sends statistical job to the program.

The data stored in Parallel DBMS server is populated by the application server. Upon receiving the larger data from Application Server, the data is partitioned across the cluster. The master node (query optimizer) then assigns the work to the slave nodes for processing. The underlying

hardware or node have multiple processors, each with its own local memory and disk (mostly uses shared nothing architecture) where each of the processor has direct access only to the instructions and data stored in its local memory but can also communicate to other processors within the server by the support of interconnection network which enable the flow of information among each of the processors.

This enabled processors to communicate with each other in a distributed memory systems. When each of the worker nodes finishes the task given to it, it will then be collected by the master node. The master node will then have the final parallel computational data from the worker nodes and keep/stored in the server which then serves as a data source to the GNU-R engine. Upon user sending statistical job to the R-engine, the R-script will reclaim data from the Database servers or the File Repository server and do the final visualization for the user, e.g. running regression.



CHAPTER 4: The Approach of Hadoop-Bridge with A High Query Language

In this section, a brief discussion of the following point are categorized as follows: a) the components of the model (Hadoop Statistical Service Engine Model) b) Architecture Overview of the model with detail explanation of how the flow of the program execution takes place from the user to the back end of the statistical service engine solution.

4.1: Jaql query language

The programming interface of the Hadoop DBMS is quite too low-level for most of its users which was claimed to be the main defect of the program. The JSON high query language like jaql, pig and so on is used on top of Hadoop to meet the needs/satisfaction and the requirements of the users [2]. Jaql is an open-source dataflow and functional query language that provides users with a simple, declarative syntax with rich data processing features such as filtering, joins processing, transformation, grouping and aggregation JSON data.[2] [17]

MapReduce was incorporated into the JAQL engine, and then query jobs are able to be processed by a Hadoop cluster. Jaql will have access to the same cluster nodes, and will compute the same data on the Hadoop File System, to produce fair and reliable results. Although Jaql operates directly on user data files; it makes use of Java Script Object Notation (JSON) views to facilitate data processing. [2]

Considering for an instance, an application log which is an example of JSON:

```
%cat log.json
[
  { from: 101,
    to: [102],
    ts: 1243361567,
    msg: "Hello, world!"
  }, .....
]
```

```
.....  
]
```

```
% cat user.json
```

```
[  
  { id: 101,  
    name: "Alicia Fox",  
    zip: 95008  
  },-----  
-----  
]
```

```
[17]
```

The file consists of an array, delimited by brackets '[]', of JSON objects (or records), delimited by braces '{ }'.

Jaql is primarily used to analyze large-scale semi-structured data. Core features include user extensibility and parallelism. The following Jaql example is using the log and user data from JSON example. Our first task is to load the locally stored JSON data into Hadoop file system, HDFS (We store data in HDFS since it can be processed in parallel using map-reduce).

```
// load log data  
read(file("log.json")) -> write(hdfs("log"));  
  
// load user data  
read(file("user.json")) -> write(hdfs("user"));
```

“read is an example of a source whereas write is an example of a sink. In the example, we've constructed a simple pipe to read from a local file and write to an HDFS file.”[17]. Then it further queries the JSON data and transforms it to the language that the R engine can understand to further do the visualization analysis, for instance regression.

In addition to Jaql, it has the advantage to flexibly read and write data from different types of data stores and formats. Its core IO functions are read and write functions that are parameterized for specific data stores (e.g., file system, database, or a web service) and formats (e.g., JSON, XML, or CSV). [12]

4.2: Hadoop Data Management Systems

Initial enterprise data storage has been done in dataware houses which handle a huge datasets. Though, dataware house is a central repository for a subset of data that an enterprise's various business systems collect. Historically, data warehouses were most often used as a central repository to integrate, cleanse and reconcile data used for business intelligence (BI) and analysis. However, it is costly and its flexibility of only handling the structured data might be a great disadvantage in the analyst point of view in recent years.

This is because, analyst wants to deal with data that they can be able to work with being structured, semi-structured or even unstructured data to get rid of the lengthy cleansing process needed for data warehousing. For these reason, Map-Reduce and its open-source implementation Apache Hadoop have become popular and widespread solutions. [2]

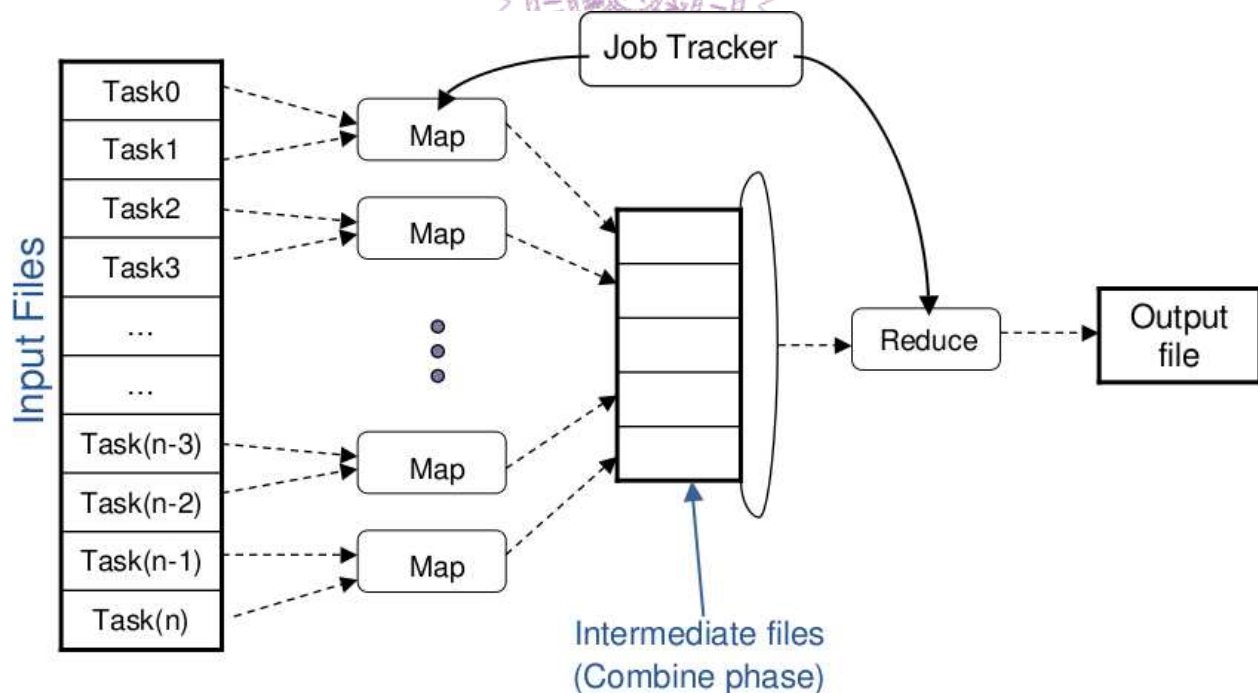


Figure 9: Map-Reduce Data Flow [19]

The Hadoop implementation can be categorized into two major parts: parallel data processing system called *MapReduce* and the file store system call *Hadoop Distributed File System (HDFS)*.

Map Reduce computation takes a set of input key/value pairs (tuples), and produces a set of output key/value pairs. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values connected with the same intermediate key and passes them to the reduce function. The reduce function, also written by the user, accepts an intermediate key and a set of values for that key. It merges these values together to form a possibly smaller set of values. [18].

The job tracker is running multiple daemons to ensure task resiliency in the cluster, so Hadoop recommends that this also be its own dedicated server

“HDFS stores files across a collection of servers in a cluster. Files are decomposed into blocks, and each block is written to more than one of the servers. This replication provides both fault-tolerance (loss of a single disk or server does not destroy a file) and performance (any given block can be read from one of several servers, improving system throughput. Because HDFS expects failure, organizations can spend less on servers and let software compensate for hardware issues.”[18]

4:2.1 some advantages of Hadoop/map-reduce

Map-Reduce is attractive because it provides a simple model through which users can express relatively sophisticated distributed programs, leading to significant interest in the educational community. It doesn't require any scheme support/define and is free to structure there data in

any manner or even to have no structure at all, thus that make it flexible to handle any big data in any format. [20]

In addition, below are the lists of some advantages in a distributed network:

- Simple programming model. The end-user programmer only writes map-reduce tasks.
 - Fault tolerance by detecting faults and applying quick, automatic recovery. Tasks are independent so, is easy to handle partial failure. Here the entire nodes can fail and restart.
 - it provides superior scalability, elasticity and fault-tolerance properties on large clusters of commodity machines [2]
 - HDFS is scalable and fast access to this information and it also possible to serve large number of clients by simply adding more machines to the cluster.
 - Hadoop uses MapReduce framework which is a *batch-based*, distributed computing framework, it allows paralleled work over a large amount of data.
 - It has the ability to write MapReduce programs in Java, a language which even many non-computer scientists can learn with sufficient capability to meet powerful data-processing needs
 - HDFS simplicity and robust model leads it to store large amount of data or information
 - HDFS is integrated well with Hadoop Map Reduce, allowing data to be read and computed upon locally when possible.
- [21]

4:2.2 some disadvantages or limitations of Hadoop/map-reduce

MR frameworks do not provide built-in indexes, this make Hadoop slow in querying through data. The programmer must implement any indexes that they may desire to speed up access to the data inside of their application. Also, due to its increase start up costs, as more nodes are added to clusters, Hadoop become slower in query time [20].

In addition to the above, below are the lists of some of its disadvantages in a distributed network

- Map Reduce is a *batch-based* architecture that means it does not lend itself to use cases which needs real-time data access.
- Cluster management is hard:- In the cluster, operations like debugging, distributing software, collection logs etc are too hard
- Joins of multiple datasets are tricky and slow:- No indices! Often entire dataset gets copied in the process.
- It's single master requires care and may limit scaling

[21]

4:2.3 an example of Hadoop/map-reduce plus sample code

A typical real example, suppose that I am an examiner in *Ole's Institute of Technology* (OIT). At the end of the academic year, am supposed to mark the student's script on "Introduction to Data Analysis". Suppose I have 500 papers to mark and as a professor, I was assigned 4 teaching assistants to help mark the papers within a short period of time. Let's say the exam consist of 5 compulsory questions for students to answer. . In order to grade the exam, I had decided to divide the 500 exams script into five loads of hundred exams for each (me and my four TA's),

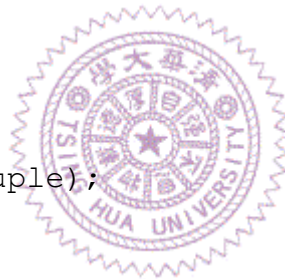
and each of us graded all the papers in one of the loads we had. So the work have been shared into 5 equal parts, I marked the first bunch of 100 scripts, TA1 marked the next 100 scripts, TA2 the next and up to the last TA. In conclusion, the students' papers are considered to be the "data" here which are equally divided into five parts (between me and my TA's) whom we are regarded as the nodes/slaves and each slave worked on the data he's being assigned to and equally applies more or less the same grading instructions to each paper. Thus the approach here is considered to be an example of data parallelism in Hadoop clusters.

A simple example showing map-reduce jobs:

Sorting

– Map/Reduce job:

```
map(key, tuple) {  
  emit(YEAR(tuple.date), tuple);  
}  
  
reduce(key, tuples) {  
  emit(key, sort(tuples));  
}
```



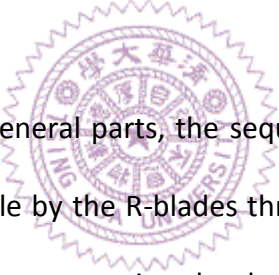
The example above is selecting a key from a table and sorts the output data by year.

SOURCE: Data management in Cloud – Advance Topics in Databases (2011) by Saake/Schallehn (FIN/ITI)

4.3: The Architecture Overview of Hadoop-Bridge Statistical Service

Engine Solution

In our architecture design, we shows the integration of the two systems, direction A showing the former [5] model of the architecture which runs in sequential mode with R-engine at its back-end; and also the extended version of [5] which shows the parallel computation of the statistical jobs sent by the user and executed in parallel with new component of Hadoop clusters and R-jaql bridge serving as the middleware between the R-engine and the HDFS. Message Oriented Middleware is mainly the middleware that facilitates communication between distributed application systems.



Our model is divide into two main general parts, the sequential part which handled data that are well structured and can be handle by the R-blades through the file repository or database and second; the approach of parallel computation that handle a big data set of all kinds, being structured or not- The parallel computation part consists of three main mechanism; an R-engine which is operated by the data analyst, the R-jaql bridge which sits in the middle between R-engine and Hadoop Clusters, and thirdly the Hadoop clusters that hosts the data and runs Jaql (and possible also some R sub-processes). [2]

The given data to be executed can be divided into two main parts; small-data part which is executed in R and the large-data part that is executed in the Hadoop/Jaql DMS. This is for the reason that the amount of data that need to be executed will be sufficiently small and will be

well distributed among the two components in the model. In that, each of the mechanism will be able to execute the task that's well suit it.

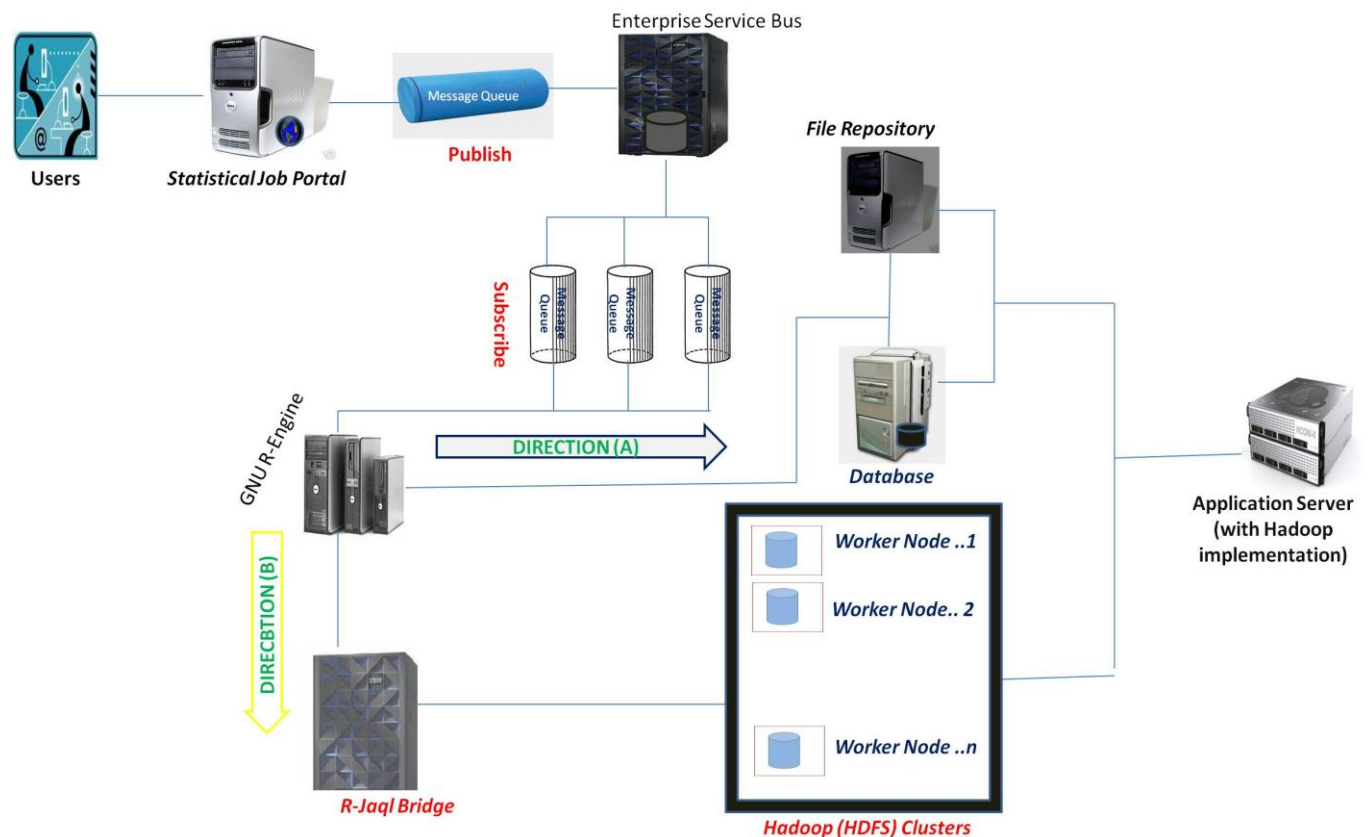


Figure 10: Hadoop/Map-Reduce Approach to the statistical Service Engine Solution

In the diagram above, '**Direction A**' shows the sequential path. Likewise, in the '**Direction B**' from the figure above, i.e. the parallel computation of the data which shows the trade between R and Hadoop. R sends an aggregation-processing queries to Hadoop (written in the high-level Jaql query language), and Hadoop sending aggregated data to R for advanced statistical processing or visualization. Since R cannot parallelize the data or execute unstructured data, it then waits the aggregation data from Hadoop which is transform by R-jaql bridge to language that R understood enabling R to analyze and visualize the data, for instances running

regression .Here, the base data is stored in a distributed file system within the Hadoop cluster, which comprises a set of worker nodes that both store data and perform computation. Hadoop and Jaql are responsible for data storage and large-scale aggregation. Jaql provides a high-level language to process and query the Hadoop data, compiling into a set of low-level Map-Reduce jobs. The R-Jaql Bridge serving as the communication layer between the two components (i.e. R and Hadoop), execute Jaql queries, transform and receive the results of such queries as R data frames and allows Jaql queries to produce R processes on Hadoop worker nodes.[2]

“The bridge comprises an R package that provides integration of Jaql into R and a Jaql module that integrates R into Jaql. The latter module allows worker nodes to execute R scripts in parallel and R functions to be used inside Jaql queries”[2]. The Application servers with Hadoop implementation built in it are user process engines populating source data on the Database servers or the File Repository server as well as the Hadoop Distributed File System (HDFS) clusters.

CHAPTER 5: *Message Passing Interface (MPI) Approach of Parallel Computation*

Message Passing Interface (MPI) is not a language or an implementation but rather an independent, standard or specification in message passing library that support portable parallel programming which is available on most of the commercial multicomputer. MPI is an industry standard (not an IEEE or ISO standard) for writing message passing programs on high performance computing (HPC) platforms. The main function of MPI is the message passing from one processor to another in a shared or distributed memory within a high computing platform. The MPI interface is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes in a language-independent way [Schmidberger, Morgan, & Eddelbuettel (2009)]. It is based on MPI Forum, which has over 40 high performance computing participants from different organizations, including researchers, users and so on. There are more than a few MPI implementations, but most of them consist of a set of routines that are callable from FORTRAN, C, and C++, as well as any language that can interface with their libraries. [21][23]

However, the advantages of developing message passing software using MPI is to “meet the needs of the majority of users in order to foster the use of a common interface on the ever-growing number of parallel machines. By separating the interface from the implementation, MPI provides a framework for Massive Parallel Processing (MPP) vendors to utilize in designing efficient commercial implementations”[22] and as well to create a *portable, efficient*, and

flexible standard for message passing that will be generally used for writing message passing programs in a distributed network.

Some of the goals of MPI Forum were defined: (*Gropp and Lusk (2002) [21]*)

- MPI would be a library for writing application programs, not a distributed operating system.
- MPI would support heterogeneous computing (the MPI Datatype object allows implementations to be heterogeneous), although it would not require that all implementations be heterogeneous
- MPI would be capable of delivering high performance on high-performance systems. Hence, no memory copies would be mandated by the design. Scalability, combined with correctness, for collective operations required that groups be “static”.
- MPI would require well-defined behavior (no race conditions or avoidable implementation-specific behavior).

5.1: Programming Model

MPI runs on virtually any hardware platform, for instance distributed Memory, shared Memory or hybrid .Earlier introduction of MPI were entirely designed to run on distributed memory architecture. [23].

There are two main types of parallel systems in message passing interface: shared-memory systems and distributed-memory systems. Pthreads and OpenMP were designed for programming shared-memory systems. They provide mechanisms for accessing shared-memory locations. OpenMP allows us to parallelize many programs with relative ease, while Pthreads

provides us with some constructs that make other programs easier to parallelize. PVM and MPI, on the other hand, were designed for programming distributed-memory systems. Thus, MPI provides mechanisms for sending messages. [21][24]

a) Shared-memory system

In a shared-memory system, the nodes can share access to the computer's memory; in standard, each node can read and write each memory location. In a shared-memory system, we can coordinate the nodes by having them examine and update shared-memory locations. [21][23][24]

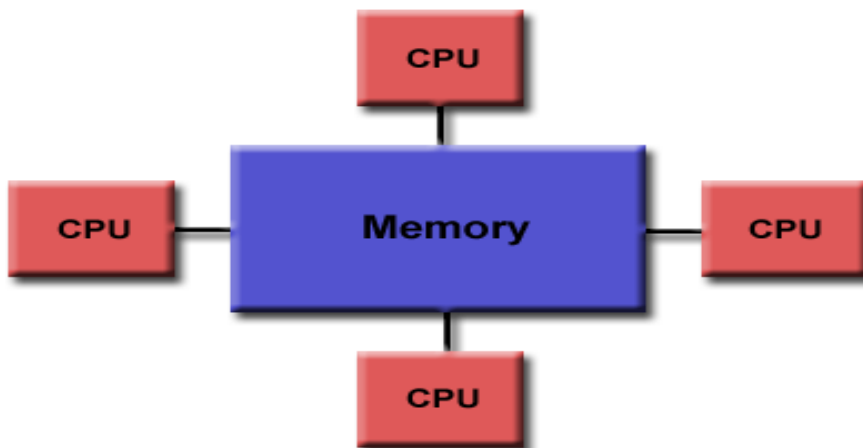


Figure 11: A shared-memory system [Source: *Barney, Livermore (2012)*]

Here, some of the advantages are that all processors can access the same memory and no internetwork is needed for the communications of processors to send messages. The tasks share a common address space, which all the processors can read and write asynchronously (meaning, each can get the message whenever it wants to). The limitations/disadvantages

might be the difficulty of handling/controlling the data locality and the management of the memory to avoid synchronization conflicts.

b) Distributed-memory system

In a distributed-memory system, on the other hand, each node has its own private memory, and the nodes must communicate explicitly by doing something like sending messages across an interconnection network [21][23][24]. In this model of communication one device or process (manager) controls one or more other devices or processes (workers). Once a manager-worker relationship between devices or processes is established (spawned), the direction of control is always from the manager to the workers [24].

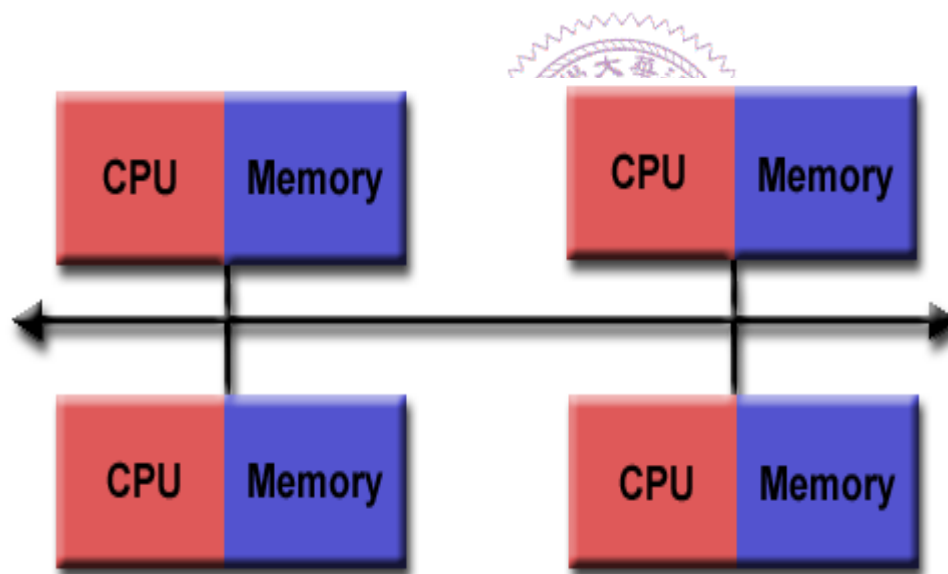


Figure 12: A distributed-memory system [Source: *Barney, Livermore (2012)*]

Here, Distributed memory systems require a communication network to connect inter-processor memory. Thus it has the advantage of no memory interference since each process controls its own memory; it is not possible for one processor to accidentally overwrite a variable controlled by another processor. It is cost effective since multi computers can

communicate and execute parallel programs in distributed computing network through the existence of interconnection network and the scalability of its memory with a particular number of processors. In the other hand it has the disadvantage of developing a programmer that will be responsible for data communication between processors. [25]

c) Hybrid Distributed-shared memory system

“As architecture trends changed, shared memory was combined over networks creating hybrid distributed memory / shared memory systems. MPI implementors adapted their libraries to handle both types of underlying memory architectures seamlessly”. [23]

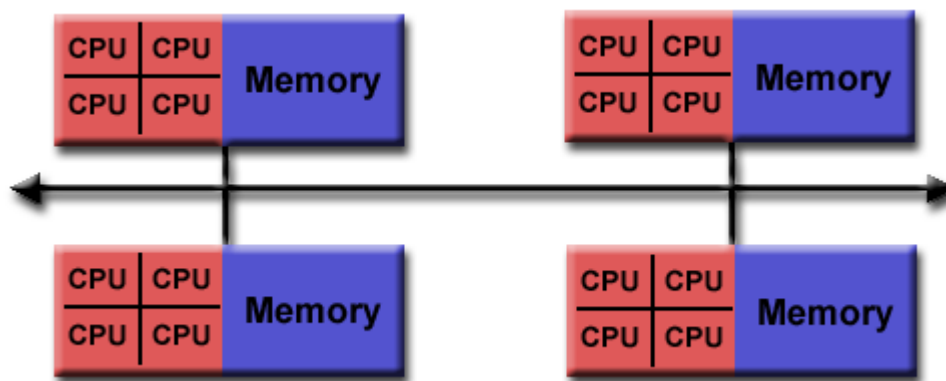


Figure 13: A hybrid distributed-shared memory system [Source: *Barney, Livermore (2012)*]

Thus, the hybrid distributed-shared memory system is the combination of both the shared and distributed memory computing system which is more applicable for the current fastest computers in a high performance computing environment. 21][23][24]

5.2: Operations for Communications

There are two major types, which are Point-to-Point Operations and collective operations:

In MPI, process passes messages both to communicate and to synchronize with each other.

When a message containing data passes from one process to another, it is serving as a communication function. A message has a synchronization function when for instances Process Z cannot receive a message from Process Y until after process Y sends it. Thus process Z receiving the message indicates something about process Y.

a) Point-to-Point Operations

MPI point-to-point operations typically involve message passing between two, and only two, different MPI tasks. Data is explicitly sent by one process and received by another processes within the network.

In point to point operations; different types of send and receive routines used for different purposes and MPI communication modes differ in what conditions on the receiving end are needed for completion for instance buffered send , blocking and non-blocking. [23][25] Different types of point to point operations is available on (Gropp,Lusk(2002)), (Barney and Livermore)

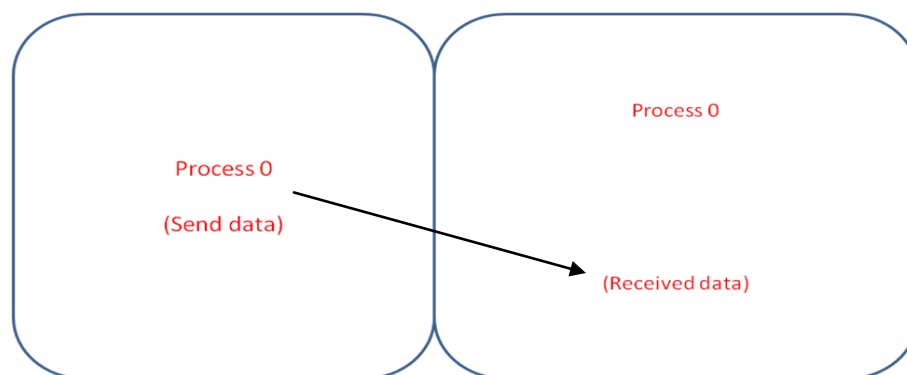


Figure 14: shows the simple send and received

Here, communications and synchronization is combine

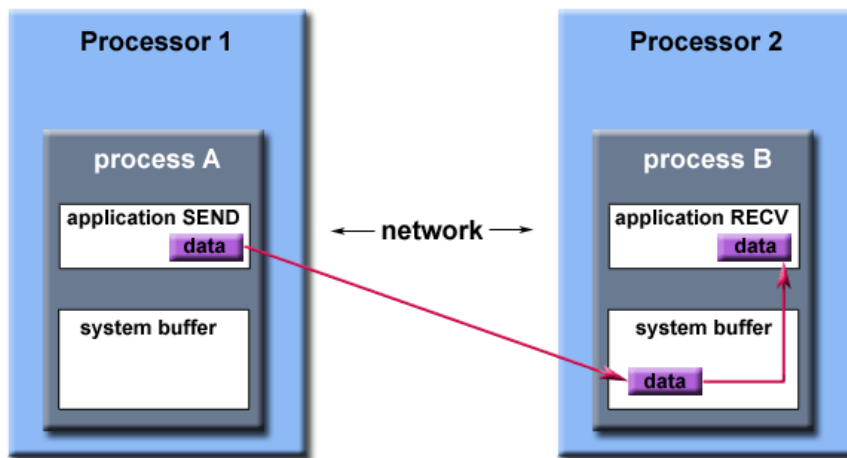
Buffering (Barney, Livermore (2012))

“The MPI implementation must be able to deal with storing data when the two tasks are out of sync.

Consider the following two cases:

1. A send operation occurs 5 seconds before the receive is ready - where is the message while the receive is pending?
2. Multiple sends arrive at the same receiving task which can only accept one send at a time - what happens to the messages that are "backing up"?

The MPI implementation decides what happens to data in these types of cases. Typically, a system buffer area is reserved to hold data in transit. “[23]



Path of a message buffered at the receiving process

Figure 15: Path of a message buffered at the receiving process [Source: Barney, Livermore (2012)]

b) Collective Communication

This type of communication is the situation where all the processes must involve in the scope of a communicator. Under normal circumstance, all processes are elements in the communicator `MPI_COMM_WORLD`. It is the programmer's responsibility to ensure that all processes within a communicator participate in any collective operations. [23]

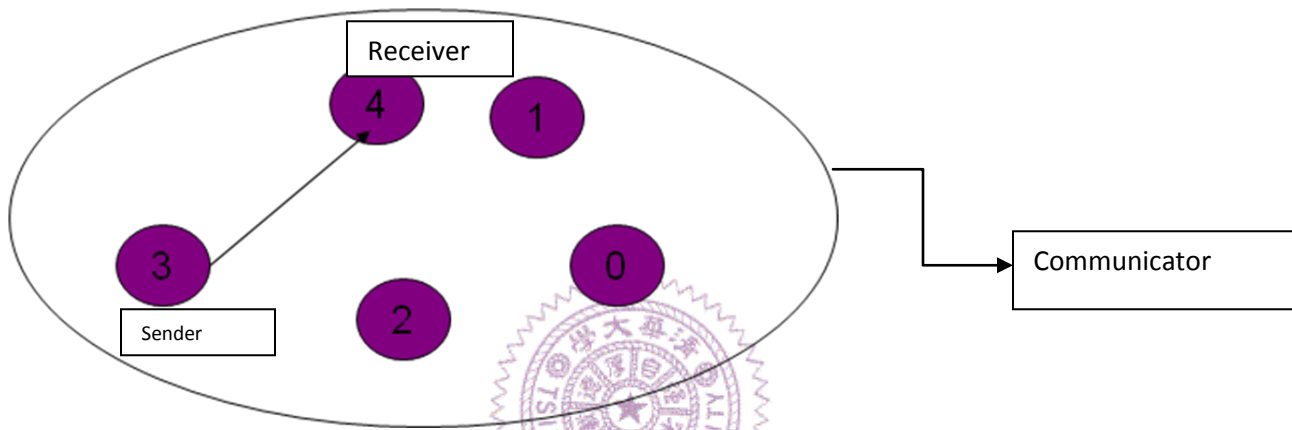


Figure 16: shows processes within a communicator

From the figure above, the Processor 3 (sender) process sends message to Processor 4(receiver) process within the scope of the communicator. The RECEIVER process is identified by its rank in the communicator.

5.3: Sample design and Examples

a) Example

Since Message Passing Interface deals with task parallelism; below is a simple example of the approach of data division among individuals in an organization. In task-parallelism, we partition the various tasks carried out in solving the problem among the processors or slaves.

A typical real example, suppose that I am an examiner in Ole's Institute of Technology (OIT). At the end of the academic year, am supposed to mark the student's script on "Introduction to Data Analysis". Suppose I have 500 papers to mark and as a professor, I divide the work among 4 other examiners to help mark the papers within a short period of time. Let's say the exam consist of 5 compulsory questions for students to answer. . In order to grade the exam, I had decided to divide the 500 exams script for each of us to grade all five hundred responses to one of the questions; for instance I graded question 1's, Examiner 2 graded question 2's, Examiner 3 graded question 3's, and so on. There are five tasks to be carried out: grading the first question, grading the second question, and so on; and then in this case, I and the other 4 examiners are the "processors/cores "and the student examination papers are the "data". Thus, the task is equally divided, each worker can execute the data given to it in parallel. The approach here is considered to be an example of task parallelism in MPI.

b) The simple Architecture of MPI

Example: 1 master, 3 slaves. The master sends information (data) to the slaves to work on.

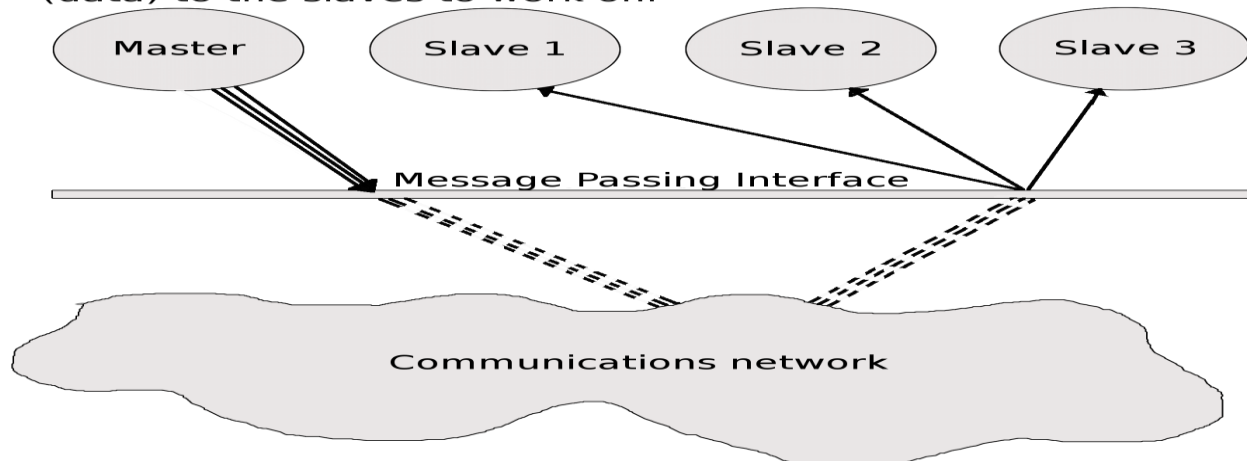


Figure 17: Master/Slave MPI Architecture Diagram [Source: http://www.hpc2n.umu.se/support/beginners_guide]

Above is the simple scenario of Message Passing Interface where the jobs are divided into workers/slaves by the master to speed up the performance of the computation over the distributed network.

5.4: Advantages and Disadvantages of Message Passing Interface (MPI)

Some of the pros and cons of MPI are described below:

a) Advantages of MPI

- MPI is a language-independent communications protocol used to program parallel computers
- MPI is very portable and it is generally optimized for the hardware it runs on, so will be reasonably fast.
- MPI has a convenient interface that provides:
 - mechanisms for performing synchronization and
 - syntax for data movement
- MPI is good for the execution of task parallelism in a high performance computing network
- It enables the communication to work on both synchronization and asynchronization by offering buffer to each of the processor
- Programs that are parallelizable should be reasonably easy to convert to MPI programs by adding MPI routines to i.e. point to point communication routines, Environment Management routines, Virtual topology routines and so on.

b) Disadvantages of MPI

- However, the MPI processes are not reliable; if one compute node fails the entire application needs to be restarted. There are solutions for a fault tolerant MPI implementation; however, they are applicable only to a specific set of problems [19].
- In a distributed memory system, MPI required a programmer that will be responsible for data communication between processors
- In a shared memory system, MPI is difficult to handle data locality and the management of the memory to avoid synchronization conflicts.
- It is harder to learn and difficult to program compared to other parallel programming languages like Hadoop



5.5: The Architecture Overview of MPI on Statistical Service Engine Solution

The following architecture design showing the approach of Message Passing Interface running in parallel at the back end of the service engine solution. The extended version of [5] showing below is the parallel computation of the statistical jobs sent by the user and executed in parallel within our third model of the paper, known as the Message Passing Interface (MPI).

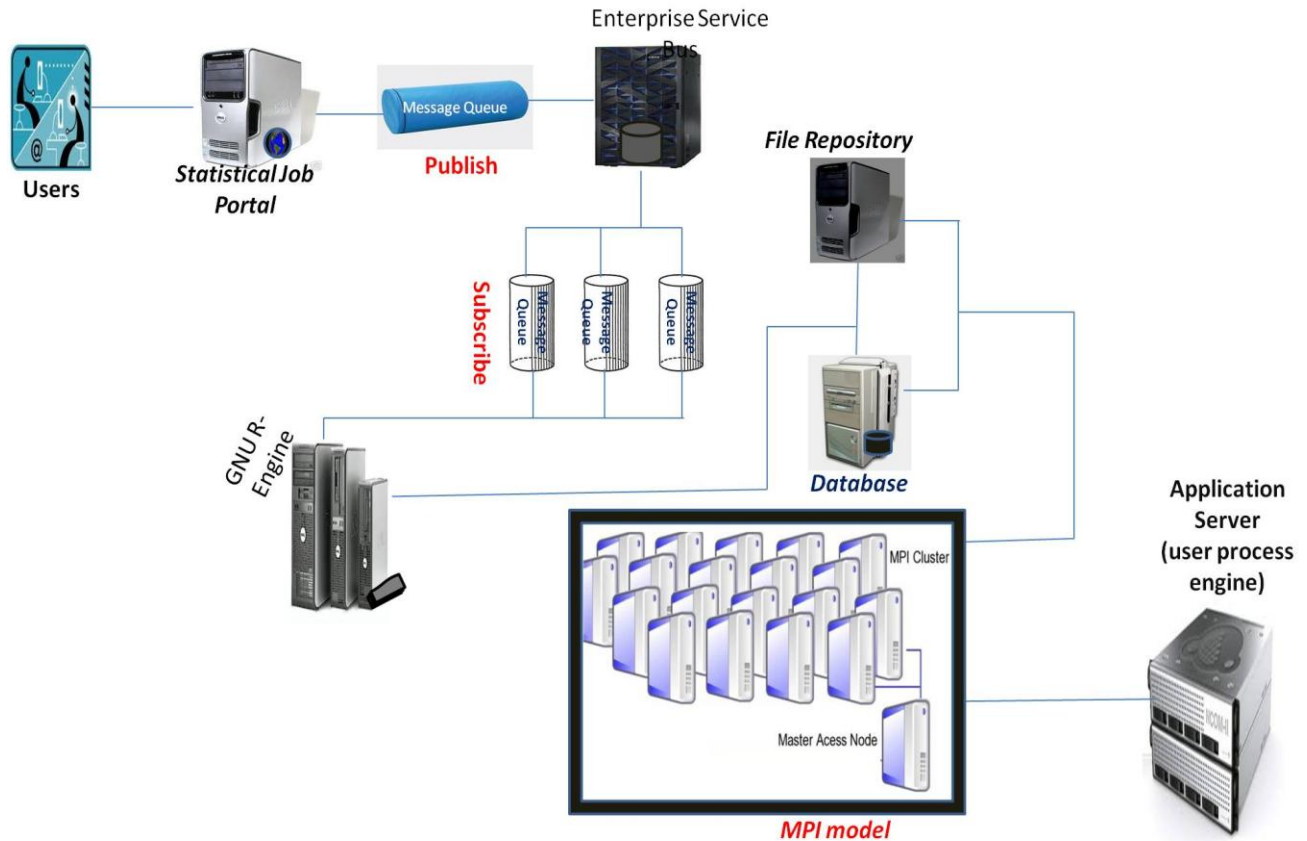


Figure 18: MPI Approach to the statistical Service Engine Solution

Referring to the “Chapter 2.1” explained [5] the serial mode of the architecture of how user sends statistical job to the program. In addition to that shows the approach of parallel computation that handles a big data-set of any format. From the figure above, the Message Passing Interface model sits between the application server and the database or file repository where R-blade is connected to. The computation of data to MPI clusters is populated by the application server to the master node of the MPI cluster. Upon receiving the larger data, the master node in the MPI clusters then distributed the data to the workers/slaves to process the task given to them. The underlying hardware or node is a collection of processors, each with its

own local memory which each of the processor has direct access only to the instructions and data stored in its local memory but can also communicate to other processors within the clusters by the support of interconnection network which enable message passing among each of the cores/processors. The existence of the interconnection network opens a door for every processor to communicate with every other processor in a distributed memory systems. Processors here, pass message both to communicate as well as to synchronize. When each of the worker nodes finishes the task given to it, it will then be collected by the master node. The master node will then have the end product of parallel computational data from the worker node clusters and put it in a readable format that the R-engine can understand (since R-engine can only execute structural data) and finally populate either to the file repository or the database which then serve as a data source to the GNU-R engine. Upon user sending statistical job to the R-engine, the R-script will repossess data from the Database or the File Repository server and run the data clear visualization, e.g. running regression.

CHAPTER 6: COMPARISON OF THE THREE MODELS

Computer clusters in a distributed computing environment are built up of many connected nodes/computers, with a limited number of CPU's and memory each of them is having. In order to get a better performance and speed-up the working load, the problem must be parallelizable.

We've done this in three main approaches by looking at the Hadoop/Map-reduce case, the Message Passing Interface and the Parallel DBMS. A simple way is to just divide the big set of data into smaller chunks, and then run the program many times for difference of parameters on your input data. The tasks are each serial programs, running on one node each, but many instances are running at the same time. Another option is to use a more complex parallelization that involves changes done to the program, in order to allow it to be split between several processors/nodes that communicate over the distributed network for example MPI.

Thus, this section, we looked at some of the relationship among the three types (mostly MR and DBMS) of approaches and there significant differences in data parallelization in a distributed computing environment. The most common argument among the programmers, users or experts lies in the usage of parallel DBMS over Hadoop/Mapreduce execution or vice versa mostly in a typically arranged in a shared-nothing Massive Parallel Processing (MPP) architecture in a computing network as we've discussed in the previous section. This is simply because they are the two widely approach use in most computing network for data processing. The comparison of MPI plays a little role in this section, thus MPI provides a framework for MPP vendors to utilize in designing efficient commercial implementations [22] and as well to create a portable (the ability of the same source code to be compiled and run on different parallel

machines), efficient, and flexible standard for message passing that will be generally used for writing message passing programs in a distributed network.

Below are the points discussed in making a comparison?

6.1: Scalability:

How large the data, can a particular organization run and analyzed in an hour or a day? Most of the enterprises are facing the trouble of executing a load of data each day for the success of the company. The amount of data that needs to be stored and analyzed is blowing up every second with the demand of running them on computing clusters of tens, hundreds or even to thousands of machines to work with in parallel computation for a better analysis performance. Thus, the scalability became one of the key concerns to the manager and his organization. Shared nothing architecture where each processor has its own local memory and disk is strongly believed to scale the best.” Furthermore, data analysis workloads tend to consist of many large scan operations, multidimensional aggregations, and star schema joins, all of which are fairly easy to parallelize across nodes in a shared-nothing network. Analytical DBMS vendor leader, Teradata, uses a shared-nothing architecture. Oracle and Microsoft have recently announced shared-nothing analytical DBMS products in their Exadata and Madison projects, respectively”[30]. The widespread use of Hadoop implementation in a range of organizations like yahoo and facebook for example; Facebook, which as of 2009 had about 400 terabytes of data that were managed by the Hadoop/Mapreduce system and consume about 15 terabytes of data each day [Facebook, 2010 [31]]

Parallel DBMS does not scale well of hundreds of nodes and have been proven to scale really well into the tens of nodes (near linear scalability is not uncommon). Few of the parallel DBMS deployment was known to be more than hundreds nodes and there is no evidence that parallel DBMS published deployment with numbers more enough to thousands nodes. One of its main failures is the addition of more nodes to a network system in which parallel DBMS were purposely designed with the assumption that failures are a rare event and few of the parallel DBM application were designed to deployed on more than a tens of nodes for reasonable performance this is because DBMS are not specifically tested at a larger scales. Map reduce are best suited for performing analysis at the scale of hundreds of nodes or up to thousands since they were designed from the beginning to scale to thousands of nodes in a shared-nothing architecture, and have had proven success in Google's internal operations and on the TeraSort benchmark as well as handling Facebook's 2.5 petabyte data warehouse [Azza, 30] . Thus, we've seen that hadoop/mapreduce was good for scalability among the three approaches. Message Passing Interface also is a good approach for handling a larger amount of data-set and MPI is best suited for problems that require a lot of inter-process communication in a distributed computing environment [22]. Though, it can scale well to several of nodes in a computer clusters, but single computing node failure will tend to the restart of entire job.

6.2: Ease of Writing code and easy understanding- programming model

In general, the Parallel DBMS or the SQL DBMSs were significantly faster and required less code to implement each task, but took longer to tune and load the data. Programs in high-level languages, such as SQL, are easier to write, easier to modify, and easier for a new person to

understand (Pavlo, 2009 [20]). The rows of relations are distributed across nodes using techniques such as hash range, and once they are in separate nodes, the execution of queries happens in parallel.

As for the Hadoop implementation, the programming model requires users to program their tasks into mappers and reducers. The map function is applied to a set of key-value tuples and transforms each tuples into a set of tuples of a different type. Then, the reduce function aggregates the set of values with the same key. The Hadoop open source implementation, mappers and reducers are written in the java programming language and required more coding for a particular parallel computing program.

So to most users' point of view, SQL language is far way easier to comprehend and write as compare to java programming because in the SQL; the user defines what the query should return not exactly how the system should retrieve it as the case of java. In SQL, the user only needs to understand different operators that are available for data manipulation and their relationship to SQL schema and the relational model of how data is stored in it. The queries are highly optimized and the parallization is done automatically depending on the distribution of the data on the nodes. [Pavlo,2009[20],30]

Finally, for the Message Passing Interface, most of the users find it harder to learn and difficult to program compared to other parallel programming languages like Hadoop (Java) and parallel DBMS (SQL) [23].

Thus, parallel DBMS SQL was much easier to write than the Java Mapper and Reducer classes.

6.3: Flexibility

For quick and dirty analysis, Hadoop/Map-Reduce works is better because it does not need schema support. MapReduce handles any kind of data format being structured, semi-structured or unstructured data to be executed. A typical semi-structured data example is an internet company with a database that contains customer profiles with information such as address, name and gender. The company may wish to predict user behavior using large amounts of ad-hoc data such as click streams i.e. their behavior of using the internet by clicking let's say an apps or so (*Gruska and Martin, 2010[32]*). In this example, the Map Reduce will do a better of combining database information about the customer and the click stream data, which will probably get redundant afterwards.

Map Reduce systems are very flexible about the representation and storage of data. The HDFS storage system is independent of the programming model that requires the user to give an algorithm for accessing the data. This shows more flexibility of how Hadoop/Map Reduce can manipulate data. The drawback here is that the programming is done in Java, [20, 30] the accessibility is only limited to users who are experts in the language which is harder to learn than Parallel DBMS SQL.

Parallel DBMSs require data to fit into the relational paradigm of rows and columns (Pavlo, 2009[20]). Parallel DBMS are highly designed for different types of data and for highly structured relational tables with rows and columns as well as index and operator for data

manipulation. The operator's ¹ does not contain all analytical needs even though they are highly optimized and there uses of indexes that help them in sorting, grouping data and so on; some relational DBMS lets their users to define their own functions (user-define function) subject to some constraints. [20, 30, 33].

MPI has the flexibility to run on virtually any hardware platform, for instance distributed Memory, shared Memory or hybrid. Earlier introduction of MPI were entirely designed to run on distributed memory architecture [23].

6.4: Performance and Efficiency

When looking at performance advantages for various tasks, the Parallel DBMS outperformed the Map Reduce system significantly, especially for the Join task². From the users point of view, (Pavlo, 2009 [20]) also recommended that it would have been better if there were a higher-level interface (e.g. Pig, Hive) for Hadoop because even with the flexibility that Map Reduce offered, SQL was still much easier to write than the Java Mapper and Reducer classes. The Pig scripting language is not only a higher level data flow language but it also has operators similar to SQL such as FILTER and JOIN that are translated into a series of Map and Reduce functions.

¹ NB: An operator is a function that takes one or more relations as input and transforms it to produce an output.

² *The Join Task*: This required finding a URL that generates the most ad revenue for a particular data range. The most important aspect was forcing the systems to combine information from two different data sets. The parallel database outperformed Hadoop by a very significant amount, which revealed a weakness in Map Reduces ability to combine information from two different data sources. The DBMS are able to take advantage of the fact that the two tables involved a partition by the join key and therefore they are able to do the join locally without any overhead of repartitioning before the join.

Parallel DBMS has built-in indexes in their programming model and uses hash or B-tree indexes to speed up the querying of data in a database. The main advantage to parallel databases is speed. The server breaks up a user database request into parts and dispatches each part to a separate computer. They work on the parts simultaneously and merge the results, passing them back to the user. This speeds up most data requests, allowing faster access to very large databases. In the other hand, the Map Reduce frameworks do not support built-in indexes. To accelerate the performance in an application, the programmers have to implement any indexes in their program to do so (*Pavlo, 2009 [20]*).

In a scenario for example, “if data analysis software product A requires an order of scale more compute units than data analysis software product B to perform the same task, then product A will cost (approximately) an order of magnitude more than B” (*Azza, [30]*). Map Reduce lacks many of the features that have proven very useful for structured data analysis workloads, and their immediate satisfaction paradigms prevent some of the long term benefits of first modeling and loading data before processing. These shortcomings can cause an order of magnitude slower performance than parallel databases [*Azza [30]*].

For more on data analysis tasks examined in a comprehensive study of performance difference between the two systems, please looked at (*Pavlo, 2009 [20]*), who implemented the tasks on Hadoop and two other Parallel DBMS systems named Vertica and DBMSX.

In terms of performance, MPI is very portable and it is generally optimized for the hardware it runs on, so will be reasonably fast. It tries to be very light and fast on message passing and provides some options to deal with data arrays. MPI would be capable of delivering high

performance on high-performance systems. Hence, no memory copies would be authorized by the design. Scalability, combined with correctness, for collective operations required that groups be “static” [21]. MPI enables the communication to work on both synchronization and asynchronization by offering buffer to each of the processor [23], this makes the communication easier as well as maintain and increase the performance speed in an environment.

6.5: Cost

MapReduce and its open source implementation called Hadoop has a big advantage over Parallel DBMS in terms of cost. There is no cost attached to them, meaning they are free source. However, all of the parallel databases mentioned above have a nontrivial cost, often coming with price tags for large installations. Furthermore, since most of the high computing platforms have inbuilt MPI facility indicating that is also a free source for the users. [30]

6.6: Fault Tolerance

The process of recovering a failure without losing any information or data from an ongoing process or task in a distributed computing database where the failure of one worker nodes doesn't have to stop the entire work is a very significant issue in the abovementioned approaches. Map Reduce ability of independent storage system makes it suitable for production environments with a combination of storage systems, and its simplicity provides a fine-grained fault tolerance whereby only the task on nodes that failed has to be restarted i.e.

the Map Reduce scheduler automatically restarts the task on an alternating node for the work to continue (*Pavlo, (2009) [20]),[30]*).

A parallel DBMS, properly configured, can continue to work despite the failure of any computer in the cluster. The database server senses that a specific computer is not responding and reroutes its work to the remaining computers [30]. Due to the larger transaction of DBMS, the nodes restarted in terms of a failure for the reason mentioned (*Pavlo, 2009 [20]*) that DBMS keep away from saving intermediate results to disk whenever possible. So, in DBMS if there is a node failure during a long running query, the entire query in a DBMS must be completely restarted.

As there are more competitive in the computing network, more and more data is evolving to be queried by the computer clusters for better analytic performance and hence the more nodes required for computing those analytical processing. Thus scalability became concern; however the nodes failures sometimes occurred at the scale and its increases the probability of at least one node failing during query execution. “Google, for example, reports an average of 1.2 failures per analysis job. If a query must restart each time a node fails, then long, complex queries are difficult to complete”[30].

In contrast to Message Passing Interface, it does not support any failover nor fault tolerance; so if one process dies or some compute node fail, it usually brings down the whole MPI job [21][23].

6.7: Brief Comparison showing in Tabular Form

No		Parallel DBMS	Hadoop/MapReduce	MPI
1	Scalability	Parallel DBMS does not scale well of hundreds of nodes and have been proven to scale really well into the tens of nodes	It provides superior scalability , Ability to rapidly process large amounts of data in parallel. HDFS is scalable and fast access to this information and it also possible to serve s large number of clients by simply adding more machines to the cluster.	MPI can scale well to several of nodes in a computer clusters, but single computing node failure will tend to the restart of entire job.
2	Ease of Writing Code	In users perspective; DBMS SQL are easier to write, modify and much more easier for a new programmer	To users; Hadoop are written in the java programming language and required more coding for a particular parallel computing program	MPI- users find it harder to learn and difficult to program compared to other parallel programming languages like Hadoop
3	Flexibility	Scheme support - Parallel DBMS required data to be fit into the relational paradigm of rows and columns. Also, required less code to implement each task	Hadoop/MapReduce- does not need scheme support (handle any kind of data e.g. structured or unstructured)	MPI – runs on virtually any hardware platform , e.g. distributed Memory, shared Memory or hybrid

4	Performance	-easier to write , has built-in indexes to speed up the querying of the data	- difficult to write and required high level language/interface e.g. pig or Jaql to burst up the speed	-Very light, portable and fast on message passing on high performance computing platform. - MPI is very portable and it is generally optimized for the hardware it runs on, so will be reasonably fast.
5	Costs	Parallel DBMS- not an open source , required monetary value for the user to use it	Open source	Open source
6	Fault Tolerance	-If properly configured, can continue to work despite the failure of any computer in the cluster. - BUT, if there is a node failure during a long running query, the entire query in a DBMS must be completely restarted	Reliability by automatically maintaining multiple copies of data and automatically redeploying processing logic in the event of failures	it does not support failover nor fault tolerance - if one process dies or some compute node fails, this usually brings down the whole MPI job **

7	Task	Good for task parallelism	Hadoop is good for Data Parallelism	MPI is good for task parallelism
9	Main goals	Parallel DBMS outperformed on performance and efficiency	it provides superior scalability, elasticity and fault-tolerance properties on large clusters of commodity machines	MPI's goals are high performance, scalability, and portability



CHAPTER 7: CONCLUSION

In order for a problem to be parallelizable, it must be possible to split it into smaller sections that can be solved independently of each other and then combined. What happens in a parallel program is generally this: There is a 'master' processes that controls splitting out the data etc. and communicate it to one or more “slave” processes that does the calculations. The 'slave' processes then sends the results back to the “master”. It combines them and/or perhaps sends out further subsections of the problem to be solved.

We’ve looked at how data is evolving in a distributed computing enterprise systems and how organization or companies are set to react towards handling or executing this larger data set. Big quantity of data set of all format; being structured, semi-structured or unstructured data is increasing rapidly and data analyst experts have come to solution to process this bunch of data using the implementation of parallelizing the data by using parallel computation. We’ve also, shown how Lee’s paper on “A Novel Data Analysis Service Oriented Architecture Using R (2012)”[5] can be further extend from running in serial mode to parallel mode still using R-engine as the main statistical analytical computation solution. Thus, this parallelization model were designed and explained using some of the three widely approaches of parallel computing which are Hadoop/Map-Reduce, Message Passing Interface (MPI) and Parallel DBMS.

In the Map-Reduce programming model, users express their problem in terms of a map function and a reduce function. Message Passing Interface (MPI) which has been implemented almost for every distributed memory-architecture uses inter-processes communication to

transmit message across the distributed computing environment through the means of interconnection network. Parallel DBMS model organize data as a system of tables representing entities and relationships between them.

However, in handling the unstructured data, Parallel DBMS approach is more advantageous in performance wise while Map-Reduce and Message Passing Interface over performed in terms of flexibility. Also, both systems of Map-Reduce and Parallel DBMS offered a high query language for example; Jaql for Map-Reduce systems, and SQL for parallel DBMS. “There tend to be two schools of thought regarding what technology to use for data analysis in such an environment. Proponents of parallel databases argue that the strong emphasis on **performance** and **efficiency** of parallel databases makes them well suited to perform such analysis. On the other hand, others argue that MapReduce-based systems are better suited due to their superior **scalability**, **fault tolerance**, and **flexibility** to handle unstructured data”(Azza, [30]). MPI provides a framework for MPP vendors to utilize in designing efficient commercial implementations”[22] and as well to create a **portable**, **efficient**, and **flexible** standard for message passing that will be generally used for writing message passing programs in a distributed network.

On the final note, in case whereby a user or manager of an organization is concern about handling the big data of their organization, let say the data becomes too large (terabytes, petabytes, etc) which includes any data format, the performance of the computation but most importantly he/she is concerned about the continual execution of the operation even if one or two machines in the distributed network fails i.e. the Fault Tolerance is among the organization

top priority list; then one of the main approaches about this paper in statistical engine model ***“The Approach of Hadoop-Bridge with A High Query Language ”*** will be an appropriate approach for the manager and his organization to implement; thus the model provides superior performance, scalability, elasticity and fault-tolerance properties on large clusters of service machines. In scenario, where the manager or the company goals are to hold and execute their big data-set (structured or unstructured) in a distributed computer network that required a lot of inter-process communication, the need for low latency reductions, performance and for short task processing but rather fault-tolerance is not their main concerned, then the second approach of this paper ***“Message Passing Interface (MPI) approach to the statistical model”*** can be recommended to the them; thus MPI is good for task parallelism. The last approach of this paper which is ***“The approach of Parallel DBMS Architecture”*** is basically good for an organization where the manger’s main concerned is the scalability, performance, and does require scheme support/define which required data to fit into the relational paradigm of rows and columns as well as proper indexing for the acceleration of the speed to search through the data-set in a distributed computing.

REFERENCES

1. Sudipto Das, Yannis Sismanis, Kevin S. Beyer, University of California(Santa Barbara, CA, USA)- sudipto@cs.ucsb.edu,IBM Almaden Research Center (San Jose, CA, USA)-{syannis, kbeyer, rgemull, phaas, jmcphers}@us.ibm.com; “Ricardo: Integrating R and Hadoop”; pg1; 2010; ACM New York,NY,USA @2010
2. Sudipto Das, Yannis Sismanis, Kevin S. Beyer, University of California(Santa Barbara, CA, USA)- sudipto@cs.ucsb.edu,IBM Almaden Research Center (San Jose, CA, USA)-{syannis, kbeyer, rgemull, phaas, jmcphers}@us.ibm.com; “Ricardo: Integrating R and Hadoop”; pg1; 2010; ACM New York,NY,USA @2010
3. Surajit Chaudhur-, Umeshwar Dayal; “An Overview of Data Warehousing and OLAP Technology”; pg1; March 1997; Newsletter ACM SIGMOD, New York, NY, USA
4. Surajit Chaudhuri- Microsoft Research, Redmond, Umeshwar Dayal- Hewlett-Packard Labs, PaloAlto; “An Overview of Data Warehousing and OLAPTechnolog”; pg1; March 1997; Newsletter ACM SIGMOD, New York, NY, USA
5. Rich C. Lee; “A Novel Data Analysis Service Oriented Architecture Using R”; March 2012; Proceedings of The International Symposium on Grids and Clouds (ICGC 2012). 26 February-2 March. Taipei, Taiwan. Published online at <http://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=153>, id.8
6. Evelson Boris; “Topic Overview: Business Intelligence”; November 2008; Forrester Research Inc
7. B. P. Kumar, J. Selvam, V. Meenakshi, K. Kanthi, A. Suseela, and V. L. Kumar; “Business Decision Making, Management and Information Technolog”; Ubiquity, vol. 2007, p. 4, 2007
8. H. Arsham (2007); “Statistical Thinking for Managerial Decisions (9 ed.)”; retrieved July 14, 2007, from <http://home.ubalt.edu/ntsbarsh/Business-stat/opre504.htm>
9. Leslie G. Valiant; “A Bridging Model for Parallel Computation”; Vol.33, August 1990; Magazine communication of the ACM New York, NY, USA

10. Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, and Caleb Welton; "MAD skills: New analysis practices for big data"; PVLDB, 2(2):1481–1492; August 2009; Journal , Proceedings of the VLDB Endowment, USA
11. Jeffrey Dean and Sanjay Ghemawat; "MapReduce: simplified data processing on large clusters"; pg 137–150; October 2004; USENIX Associations, In OSDI'04 Proceedings of the 6th conference on symposium on Operating Systems Design and Implementation, USA
12. Robert J. Stewart, Phil W. Trinder, and Hans-Wolfgang Loidl; "JAQL: Query Language for JavaScript Object Notation (JSON)"; Retrieved 2009 from <http://code.google.com/p/jaql>
13. Ross Ihaka & Robert Gentleman _ Published online: 21 Feb 2012.; "R: A Language for Data Analysis and Graphics"; May 1995; Published by: American Statistical Association, Institute of Mathematical Statistics, and Interface Foundation of America
14. W.N. Venables & D.M. Smith, R.Gentleman & R.Ihaka; "An Introduction to R", Retrieved May 2013 from <http://cran.r-project.org/doc/manuals/R-intro.html>
15. Wendy K. Murphrey-SAS Institute, April 2005 , "Connections JMP and ODBC"; retrieved May 2013 from www.jmp.com/software/whitepapers/pdfs/102427_jmp_odbc.pdf
16. Robert I.Kabacoff; "Access to Database Management Systems (DBMS)";2012 , retrieved from www.statmethods.net/input/dbinterface.html
17. Powered by Google Project Hosting, "Jaql Query Language for JavaScript Object Notation"; Retrieved May 2013, from <http://code.google.com/p/jaql/wiki/JaqlOverview>
18. Mike Olson , "HADOOP: Scalable, Flexible Data Storage and Analysis" , Spring 2010, IQT Quarterly
19. Grant Mackey, Saba Sehrish, John Bent, Julio Lopez, Salman Habib, Jun Wang "Introducing Map-Reduce to High End Computing"; 2008, Petascale Data Storage Workshop, 2008. PDSW '08. 3rd
20. Andrew Pavlo , Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J.Dewitt, Samuel Madden; "A Comparison of Approaches to Large-Scale Data Analysis ", 2009, Proceedings of the 2009 ACM SIGMOD International Conference on Management of data ACM New York, NY, USA

21. William Gropp, Ewing Lusk ; *"Goals Guiding Design: PVM and MPI"*; August 2002, Conference Paper- 4th IEEE International Conference on Cluster Computing, San Diego, CA, USA
22. P. H. Carns, W. B. Ligon III, S. P. McMillan, and R. B. Ross; *"An Evaluation of Message Passing Implementations on Beowulf Workstations"*; March 1999, Proceedings of the 1999 IEEE Aerospace Conference (Volume:5)
23. Blaise Barney, Lawrence Livermore National Laboratory; *" Parallel computing Tutorial"* Retrieved from 2012 www.mcs.anl.gov/mpi/ (<https://computing.llnl.gov/tutorials/mpi/>)
24. Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, Ulrich Mansmann; *"State of the Art in Parallel Computing with R (2009)"*. August 2009, Journal of Statistical Software, No.1, Vol.31
25. Michael J.Quinn - CHP 4 (Message-Passing programming) of the text book : *"PARALLEL PROGRAMMING in C with MPI and OpenMP"*, June 2003, McGraw-Hill Science/Engineering/Math; 1 Edition
26. David Dewitt and Jim Gray, *"Parallel Database Systems: The Future of High Performance Database Systems"*, June 1992, Magazine-Communications of the ACM Volume 35. ACM New York, NY,USA
27. M. Tamer Özsu, Patrick Valduriez ; *"Distributed and Parallel Database Systems"*; 1996, ACM Computing Surveys
28. Michael I. Gordon, William Thies, and Saman Amarasinghe; *"Exploiting Coarse-Grained Task, Data, and Pipeline Parallelism in Stream Programs"*, December 2006; ASPLOS XII Proceedings of the 12th International Conference on Architectural support for programming languages and operating systems, pg 151-162, ACM New York, NY, USA
29. M. Tamer Özsu and Patrick Valduriez; *"Principles of Distributed Database Systems, Third Edition"*; 2011, Springer New York, USA
30. Azza Abouzeid, Kamil BajdaPawlikowski, Daniel Abadi, Avi Silberschatz, Alexander Rasin *"HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads"*; August 2009, Journal –Proceedings of the VLDB Endowment. Volume 2

- 31.** Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Xhakka, Ning Zhang, Suresh Antony, Hao Lin - Facebook.2010 "Hive-A petabyte scale data warehouse using Hadoop" retrieved June 2013,from http://www.facebook.com/note.php?note_id=89508453919
- 32.** Gruska, Natalie, Patrick Martin; "Integrating MapReduce and RDBMS"; pg 212-223, 2010, *Proceedings of the 2010 Conference of the Center for Advance Studies on Collaborative Research*. New York, NY, USA:ACM
- 33.** Fei Chen, Meichun Hsu, HP Labs "A Performance Comparison of Parallel DBMSs and MapReduce on Large-Scale Text Analytics", pg 613-624, 2013, *EDBT Proceedings of the 16th International Conference on Extending Database Technology*. ACM New York, NY, USA

