# Assignment-2 FIT3142 Report

***Student ID – 26356104***
***Name – PAARTH BHASIN***

**(a) Summarise and discuss the differences between the Alluxio, Spark and Hadoop schemes.**

**HADOOP VS SPARK SCHEMES**

Hadoop is used for many more things that what Spark is used for. Spark is simply used for high speed processing of data in clusters. Whereas Hadoop does that with MapReduce but also provides its own distributed file system, HDFS, for accessing and creating distributed environments which Spark does not.

Spark is not entirely comparable to the Hadoop ecosystem but to a subset of it, namely the MapReduce module of Hadoop which is a distributed computing processing engine like Spark.

Spark is a cluster computing framework/engine used for extremely fast and real-time processing of large amounts of data. On the other hand, Hadoop is a framework for distributed processing of large data sets across clusters and contains many modules working together like MapReduce, Yarn, HDFS and Common.

The main difference between Spark and the processing engine of Hadoop, i.e. MapReduce is that of speed/performance, ease of use and cost.

**Performance/Speed**

Spark is many times faster than MapReduce and processes enormous amounts of data in real-time, i.e. virtually instantly. Both exploit the use of parallel and distributed computing to process quickly, but still Spark performs much faster. This is because in Hadoop MapReduce, each phase causes disk read/write, drastically increasing the time. But Spark performs IO (Input/Output) operation for read and write data on HDD once at the times of shuffles and does all the processing in main memory causing it to work much faster, though it can also use disk memory space if main memory space is insufficient.

Hadoop MapReduce performs batch-processing of instructions, i.e. sequentially. Whereas Spark performs them at once, which again contributes to the delay.

**Machine Learning Libraries and Ease of Use:**

Spark has user-friendly APIs in a range of programming languages R, Java, Python, Scala. Whereas Hadoop is limited to only Java. Spark in addition also provides developer libraries for machine learning, graph programming and streaming, making it very easier for developers to develop applications for this on Spark. Hadoop MapReduce on the other hand does not provide any such libraries for developer use.

**Cost and Requirements:**

Spark is more expensive because it has a higher RAM requirement in order to process quicker but requires less number of systems than MapReduce. Hadoop MapReduce requires lots of fast disk space which can also be cost issue.

**ALLUXIO SCHEME:**

The Alluxio scheme on the other hand is a bridge/middleware between the computation engines like Spark, Hadoop MapReduce etc. and the actual physical storage systems like HDFS, Amazon S3 etc.

Unlike the Hadoop and Spark schemes which are used for different ways of processing large amounts of data, Alluxio is used to provide a mechanism for them to interface with the storage, making data accesses much more faster. It gives a single point of access to data, which Hadoop and Spark don't have. It is compatible with both MapReduce and Spark.

**(b) Summarise and discuss the differences between the Amdahl and LogP performance models.**

**Amdahl** performance model is based on speedup of a distributed system whereas the **LogP** performance model is mainly based on latency in a distributed system and avg. performance of its processors.

**LogP** model does not have distinct barriers between computation and communication phases and analyses performance including both of them. This makes it more general than **Amdahl** model which has clear distinction between communication (network delay) and computation (serial delay).

**Amdahl** performance model describes a simple relationship between the **speedup** that can be achieved by a distributed system application with a certain number of processors, using a certain algorithm and the percentage of **serial** and **parallel** instructions in the algorithm that is employed by this application. So **Amdahl** performance model assesses the distributed system in 3 main areas: **S** (serial component of workload) and hence **P** (parallel component of workload), and **N** (the number of processors).

**LogP** performance model on the other hand specifies the performance characteristics of interconnection/working of the network used by the algorithm of the distributed system application in consideration, without describing the structure of the network itself. It assesses this performance in four main criteria/areas:

1. **L** (latency or delay) for communicating message from source to target,
2. **o** (overhead) that processor needs for transmission and reception of each message
3. **g** (gap) which is the minimum time delay between consecutive message transmission and receptions by a processor
4. **P** (processors) which is the number of processors available for use.

The **LogP** performance has some assumptions and rigid requirements that needs to be fulfilled in order to be able to apply this performance model. These requirements are not necessary for the usage and applicability of Amdahl performance model:

1. The network should have a finite capacity so that at one time only $\lceil L/g \rceil$ (ceiling(L/g)) messages can be at transit between any two processors at a time. On the other hand there is no such capacity requirement in **Amdahl** performance model. It can be used regardless of the network congestion at any time.

2. The processors in the distributed system need to work asynchronously to be able to apply and evaluate the **LogP** performance model. This is so that we can correctly determine L, o and g. If they work synchronously, then L and g vary depending on how many processors a processor is interacting with. This is because latency changes, depending on synchronous dependency on other processors causing network delay. Gap changes depending on if the processors are waiting on any network delay due to other processors using it. On the other hand, there is no such requirement for processors to work asynchronously in **Amdahl** performance model.

3. Lastly, the message size of a transmission between two processors needs to be small for the applicability of the **LogP** performance model (even though LogGP model, which is an extension of **LogP** handles large message sizes as well). The **Amdahl** performance model is completely independent of the message size of transmission and does not require it.

To sum up, **Amdahl** performance model assesses the parallelisation ability of an application/algorithm. Whereas the **LogP** performance model focusses more on the average behaviour of the processors in terms of their processing speed in a distributed computing system and performance of the internetworking and communication between them.

These two performance models are highly independent and illuminate different areas of performance of a distributed computing system.

**(c) Summarise and discuss the differences between the Gustafson and MRM performance models.**

## INTRODUCTION

Both **MRM (Machine Repairman)** performance model and **Gustafson** performance model relate to **Amdahl** performance model. But the key differentiating factor between the two is in the way they relate to the Amdahl performance model and resemble it in different ways, extending and supporting its use in different situations and areas.

**MRM** performance model considers machines or workers (the number of processors) being run in the distributed system as having a limited life time after which they don't work well and need to be repaired by one single repairman. This puts them in a waiting queue of repairs causing unavoidable serial delay.

The **Gustafson** performance model on the other hand considers a processor's lifetime as infinite which would keep functioning as expected. It works independent of a processor's lifetime without taking that into consideration.

## APPLICATION SPEED-UP VIEW

The **MRM** performance model has a different view of how a distributed system application speeds up in comparison to Gustafson performance model.

In the **Gustafson** performance model, more serial component of the task reduces speed up. Whereas longer repairing queues and shorter machine lifetime increase the serial delay in the Machine repairman model and hence decrease the speed up.

## SERIAL AND PARALLEL WORK VIEW

The **MRM** performance model mentions that despite the existence of unavoidable serial delay (S) for repairing a machine, the time it takes for an individual task to be completed is inversely proportional to the machine workers available at a time. This means that it assumes that serial delay is only caused when a machine needs repair and not because of the serial component of the task itself. The parallelisation and hence speed up of a task increases only with more number of processors available for use and not by the amount parallel component existing in the task.

The **Gustafson** performance model on the other hand has a different view of what is considered serial delay and what is considered parallel workload. It states that the amount of parallelizable work that can be done in a certain time increases as the processors or machines become more advanced and robust. This causes serial delay to become smaller in comparison to the entire work load.

Unlike the **MRM** model which considers the entire task as parallelizable affected only by the number of workers available, the **Gustafson** performance model takes into consideration the parallelizable component of the task itself along and not the number of processors to calculate the speedup

## IN GENERAL:

The **Gustafson** performance model focusses on how with improving quality of processors the speedup improves linearly. Whereas the **MRM** performance model focuses on how lifetime of processors and the number of processors available affect the speed up achievable. They are both extensions of **Amdahl** performance model, making it more broadly applicable for various other factors that it does not consider while computing the speedup achievable.

The **MRM** interpretation of **Gustafson** is that the improvement/enhancement or scalability of processors decreases synchronous queuing delay or the mean execution time/lifetime of processors.

**(d) Summarise and discuss the differences between the harmonic speedup and Erlang performance models**.

## INTRODUCTION

Both the **harmonic** performance model and the **Erlang** performance model work on having a more realistic achievable speedup, which is greatly over claimed by all the previous performance models. The difference between **Erlang** and **harmonic** models, however, lies in the different ways they take to bring more realistic speedup computations in the MRM model.

While the **Erlang** performance model tries to realistically match the theoretical maximum speed-up available in the MRM model, the **harmonic** performance model tries to encapsulate a lower bound on MRM model, to see how it would work in the worst case scenarios.

## ERLANG PERFORMANCE MODEL VS HARMONIC PERFORMANCE MODEL

The **Erlang** performance model (Erlang-B) was devised to get a more accurate overview of the achievable speedup in the practice. It builds upon the **Machine Repairman** (MRM) model assuming *asynchronous* messaging between processors and repairman, to find out the speedup by using the Erlang-B function.

It is this asynchronous message passing assumption that tries to match the performance achievable in the MRM model. The Erlang model also ensures that the up times (or the life time) of every processor does not have a lot of variation as the MRM model assumes from the start (they have constant life times). Using this model, it is possible to know how many machines are alive at any one time, making the speedup more realistic and achievable.

While the **Erlang** performance model tries to meet the upper bound of MRM mode by asynchronous messaging and small processor lifetime variation, the **harmonic** model tries to set a lower bound on it. It does so by taking into consideration that the workload may not use all the available processors for concurrent execution even though it could've if they were available or in working as expected. This brings an even closer lower bound on the MRM performance model.

With **harmonic** speedup model, the workload is equally likely to be distributed among any number of processors, ranging from 1 to all processors, in any phase of execution in the distributed system application.

On the other hand, **Erlang** performance model assumes that a workload will take all the available processors for use if it can. Using the information in **harmonic** performance model, the workloads can be intelligently distributed across the various processors available to get the best possible performance out of using the right amount of processors for a task. This cannot be achieved in **Erlang** performance model since this information is not provided.

The lifetime of any processor remains constant in the **harmonic** process model unlike the **Erlang** model which considers them to vary in each phase, as shown above.

## IN GENERAL:

Both the models provide speedup which is built around the same concern, processors are not available all the time and require repair. But despite this, the difference between the two is that the **harmonic** performance model just gives a speed up including all different cases when all processors might not be available for usage on a parallel task. This is a more inaccurate worse-case scenario because it doesn't include possible downtime ranges.

The **Erlang** model goes a step further and gives a speedup which is based on the fact that processors may not always be available for use and have a range of different possible downtimes. This makes the speedup calculation much more accurate because it includes downtime possibilities and not just say they are not available for usage at all times. This makes the **Erlang** model a lot more accurate than the **harmonic** model in terms of the possible speed-up that an application can get.

**(e) State which of these six performance models is most suitable for Big Data Storage and Clouds, and explain exactly why this is so.**

The most suitable performance model has to also be the most accurate, realistic and hence achievable one. While **Amdahl**, **Gustafson**, **MRM** and **LogP** models are very good to get a start and theoretical understanding of the requirements, they are not very practical and almost impossible to achieve in real life.

The **harmonic** and **Erlang** models are more realistic. But even then the **harmonic** model gives the worst case speedup without as much accuracy as the **Erlang** performance model. **Harmonic** performance model says that there could potentially be a case that very few processors are being used to distribute the parallel workload and not causing much speed up as a result. This can happen, but is included in the **Erlang** performance model in a more accurate form and closer to achievable speedup in practice. Hence **Erlang** model is most suitable. There are many reasons for this claim.

**Big Data** storage and **Clouds** have many complexities which need to be addressed in order to make their performance better. One of them is the *lack of* robustness in terms of their ability to recover from disasters like machine downtime and faults that might occur unexpectedly. Robustness here means that the Big Data storage and Clouds should the ability to recover and work in various disasters (worst case situations) with ease without causing any loss in service. Using the **Erlang** performance model, this can be looked into because it provides insight into how different machines will perform and what can be the expected lifetime ranges for them before they hit downtime and require repair. The **harmonic** model fails to address this since it assumes a fixed failure time for every machine. With the **harmonic** performance model, the speedup we get is not of much use. This is because it simply gives a speedup if processors are not working, and does not include details which take into consideration various possibilities of downtime ranges of a processor, which gives a more accurate speedup calculation.

With increase in the number of machines, the number of faults arising also increase. To be able to handle this correctly, it needs to very clear from the start that how much a machine lifetime can vary in any phase. This is handled by the **Erlang** model but not the **harmonic** model.

Big Data storage and cloud computing requires precision in terms of the amount of time it takes to complete a task. **Erlang** models gives a surety on this from its small processors lifetime variation and asynchronous message passing, which make the time needed determinable and not uncertain. Using this, Big Data Storage and Clouds can be designed around the fixed set of possible outage times of processors to keep working and recover because it is predictable. They can't recover from outage times they can't predict.

Big Data storage and Clouds require to calculate massive amounts of information in short time, and thus it requires consistent performance model of its machines, using which the processing speed attainable can be determined. **Erlang** model a simple way to achieve this and is hence very suitable.

Very large amounts of message passing occur within the application of Big Data Storage which is deployed to Virtualized Environments, i.e. Clouds. To handle this message passing in an efficient way which does not cause unnecessary access latencies, **Erlang** model has asynchronous message passing which greatly improves the message passing delay and internetworking between machines. Thus Erlang model can be used to build Big Data storage and Cloud models which assume asynchronous processing and can see the speedup the respective distributed system can hence achieve.

*References on next page*

**REFERENCES:**

**Question 1:**


**[1] J.P. Verma and A. Patel, "Comparison of MapReduce and Spark Programming Frameworks for Big Data Analytics on HDFS", (IJCSC) International Journal of Computer Science & Communication, pages 80-84, Jun 2016**

**[2] S. Gopalani and R. Arora, "Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means", International Journal of Computer Applications, vol. 113, no. 1, pp. 8-11, 2015.**

**[3]"Hadoop vs. Spark: The New Age of Big Data - Datamation", Datamation.com, 2017. [Online]. Available: https://www.datamation.com/data-center/hadoop-vs.-spark-the-new-age-of-big-data.html. [Accessed: 12- Oct- 2017].**

**[4] "Brave New World: Hadoop vs. Spark", October 2015. [Online] Available: https://home.zhaw.ch/~dueo/bbs/files/20151007_Hadoop_Spark_Stockinger_DatalabSeminar.pdf [Accessed: 12- Oct- 2017].**

**[5] "Introduction to Alluxio – Beyond the lines", Beyondthelines.net, 2017. [Online]. Available: https://www.beyondthelines.net/computing/introduction-to-alluxio/. [Accessed: 12- Oct- 2017].**


**Question 2:**


**[1] N. J. Gunther. "Unification of Amdahl's law, LogP and other performance models for Message Passing architectures". In PDCS 2005, International Conference on Parallel and Distributed Computing Systems, pages 569–576, November 14–16, 2005, Phoenix, AZ, 2005.**

**[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian and T. von Eicken, "LogP: towards a realistic model of parallel computation", ACM SIGPLAN Notices, vol. 28, no. 7, pp. 1-12, 1993.**

**[3] D. Culler, L. Liu, R. Martin, and C. Yoshikawa, "LogP Performance Assessment of Fast Network Interfaces," IEEE Micro, 1996.**

**[4] Al-Tawil, K. and Moritz, C.A. "Performance modeling and evaluation of MPI". Journal of Parallel and Distributed Computing, 61: 202–223, 2001**


**Question 3:**


**[1] M. Gillespie. Amdahl's law, Gustafson's trend, and the performance limits of parallel applications. [Online]: https://software.intel.com/sites/default/files/m/d/4/1/d/8/Gillespie-0053-AAD_Gustafson-Amdahl_v1__2_.rh.final.pdf, 2008. [Accessed: 12-Oct-2017].**

**[2] Juby Mathew, Dr. R Vijayakumar, "The Performance of Parallel Algorithms by Amdahl's Law, Gustafson's Trend", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (6) pages 2796-2799, 2011.**

**[3] N. J. Gunther. "Unification of Amdahl's law, LogP and other performance models for Message Passing architectures". In PDCS 2005, International Conference on Parallel and Distributed Computing Systems, pages 569–576, November 14–16, 2005, Phoenix, AZ, 2005.**

**[4] N.J. Gunther. A general theory of computational scalability based on rational functions. Technical report, arXiv:0808.1431, 2008.**

**[5] "Amdahl's Law vs. Gustafson-Barsis' Law", Dr. Dobb's, 2017. [Online]. Available: http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2. [Accessed: 12- Oct- 2017].**

**Question 4:**

**[1] N. J. Gunther. "Unification of Amdahl's law, LogP and other performance models for Message Passing architectures". In PDCS 2005, International Conference on Parallel and Distributed Computing Systems, pages 569–576, November 14–16, 2005, Phoenix, AZ, 2005.**

**[2] L. Subramanian et al. MISE: "Providing performance predictability and Improving fairness in shared main memory systems". In HPCA, 2013**

**[3] N.J. Gunther, "Analyzing Computer System Performance with Perl", PDQ. 85-160. 10.1007/978-3-642-22583-3_4 (2011).**

**Question 5:**

**[1] "Big Data in the Cloud: Converging Technologies" - Intel (2014) [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/big-data-cloud-technologies-brief.pdf [Accessed: 12- Oct- 2017].**

**[2] R. Ranjan, "Modeling and Simulation in Performance Optimization of Big Data Processing Frameworks," in IEEE Cloud Computing, vol. 1, no. 4, pp. 14-19, Nov. 2014.**

**[3] I. Mytilinis, D. Tsoumakos, V. Kantere, A. Nanos and N. Koziris, "I/O Performance Modeling for Big Data Applications over Cloud Infrastructures," 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, 2015, pp. 201-206.**

**[4] Jewell D, Barros R D, Diederichs S, Duijvestijn L M, Hammersley M, Hazra A, and Zolotow C P"erformance and Capacity Implications for Big Data". IBM Redbooks, 2014.**