****

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

A Dissertation Report on

Application of Zipf’s Law for the Optimization of Hadoop Shuffle Phase and Comparison of Performance

of Hadoop Job Execution based on Shuffle Parameters

Submitted by

Aasia Afreen 1MS14CS002

Chayanika Bhandary 1MS14CS031

Jyothi Kumari R 1MS14CS049

*in partial fulfillment for the award of the degree of*

# *Bachelor of Engineering in Computer Science & Engineering*

Under the guidance of

Mrs. Geetha J

Assistant Professor,

Department of Computer Science and Engineering Ramaiah Institute of Technology, Bangalore

**M.S.RAMAIAH INSTITUTE OF TECHNOLOGY**

**(Autonomous Institute, Affiliated to VTU)**

**BANGALORE-560054**

2017-2018, [www.msrit.edu](http://www.msrit.edu),

****

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

A Dissertation Report on

Application of Zipf’s Law for the Optimization of Hadoop Shuffle Phase and Comparison of Performance

of Hadoop Job Execution based on Shuffle Parameters

Submitted by

Aasia Afreen 1MS14CS002

Chayanika Bhandary 1MS14CS031

Jyothi Kumari R 1MS14CS049

*in partial fulfillment for the award of the degree of*

# *Bachelor of Engineering in Computer Science & Engineering*

Under the guidance of

Mrs. Geetha J

Assistant Professor,

Department of Computer Science and Engineering Ramaiah Institute of Technology, Bangalore

**M.S.RAMAIAH INSTITUTE OF TECHNOLOGY**

**(Autonomous Institute, Affiliated to VTU)**

**BANGALORE-560054**

2017-2018, [www.msrit.edu](http://www.msrit.edu),

**Ramaiah Institute of Technology**

**(Autonomous Institute, Affiliated to VTU)**

**BANGALORE-560054**

# Department of Computer Science & Engineering

****

**CERTIFICATE**

This is to certify that the project work titled “**Application of Zipf’s Law for the Optimization of Hadoop Shuffle Phase and Comparison of Performance of Hadoop Job Execution based on Shuffle Parameters”** is a bonafide work carried out by **Aasia Afreen** (**1MS14CS002), Chayanika Bhandary (1MS14CS031) & Jyothi Kumari R (1MS14CS049)** in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science and Engineering during the year 2018. The Project report has been approved as it satisfies the academic requirements with respect to the project work prescribed for Bachelor of Engineering Degree. To the best of our understanding the work submitted in this report has not been submitted, in part or full, for the award of said degree.

### Signature of the Guide Signature of the HOD

Mrs. Geetha J Dr. Anita Kanavalli

**External Examiners**

Name of the Examiners: Signature

1.

2.

**DECLARATION**

We the students of final semester BE, Department of Computer Science and Engineering, Ramaiah Institute of Technology, Bangalore, hereby declare that the project entitled “**Application of Zipf’s Law for the Optimization of Hadoop Shuffle Phase and Comparison of Performance of Hadoop Job Execution based on Shuffle Parameters”,** report is completed and written by us under the guidance of **Mrs. Geetha J,** Dept of CSE, Bangalore for the partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering and has not been formed the basis for award of any other degree or diploma certificate.

Place: Bangalore

Date: 05-05-2018

(1MS14CS002 AASIA AFREEN)

(1MS14CS031 CHAYANIKA BHANDARY)

(1MS14CS049 JYOTHI KUMARI R)

**ACKNOWLEDGEMENT**

First and foremost, our utmost gratitude to *Mrs. Geetha J,* Dept of CSE, MSRIT whose sincerity and encouragement we will never forget. She has been our inspiration as we overcame all the obstacles in the completion of this project work.

Dr. Anita Kanavalli, Head of the Department of Computer Science and Engineering, had kind concern and consideration regarding project work and we would like to thank her for continuous support.

We would like to thank our beloved principal Dr. N.V.R Naidu for his support and encouragement.

This work would not have been possible without the guidance and help of several individuals who in one way or another contributed their valuable assistance in preparation and completion of this project.

We would like to express sincere thanks to all the teaching and non-teaching faculty of CSE Department and my dear friends who helped in all the ways while preparing the Report.

# ABSTRACT

Hadoop is an open source distributed processing framework that manages data processing and storage for big data applications running in clustered systems. It is at the center of a growing ecosystem of big data technologies that are primarily used to support advanced analytics initiatives, including predictive analytics, data mining and machine learning applications. Hadoop can handle various forms of structured and unstructured data, giving users more flexibility for collecting, processing and analyzing data than relational databases and data warehouses provide. It uses MapReduce for parallel processing. It consists of two major phases- the Map phase and the Reduce phase. There is an intermediate phase that goes unnoticed – the shuffle phase. According to analysis, it is found that one-third of the Map Reduce processing time is consumed by the shuffle phase and thus, an effort made in the direction of optimizing Hadoop performance by fine tuning the shuffle phase is legit. In this project, we attempt a feasibility test to check if the statistical law – Zipf’s Lawcan be applied to make smart decisions while spilling to improve the shuffle phase performance of Hadoop framework. We will also test the law against different datasets of varying sizes to check if the law has bias against some types of datasets. We will also attempt to compare the time required for Hadoop job execution by altering the different shuffle phase parameters (precisely *io.sort.factor, io.sort.mb, io.sort.spill.percent)* and create predictive models to predict the Hadoop job execution time according to the variation of the Hadoop Shuffle phase parameters*.*

**LIST OF FIGURES**

1. Hadoop MapReduce
2. Waterfall Model for our project
3. List of activities in the schedule
4. Gantt chart (Part 1)
5. Gantt chart (Part 2)
6. Existing Architecture
7. Proposed Architecture
8. Flow diagram
9. Proposed Algorithm
10. Zipf’s Law plot for Pride & Prejudice Dataset
11. Zipf’s Law plot for Jane Eyre – Autobiography Dataset
12. Zipf’s Law plot for Don Quixote Dataset
13. Zipf’s Law plot for War & Peace Dataset
14. Job execution time vs io.sort.mb for io.sort.spill.percent = 50%
15. Job execution time vs io.sort.mb for io.sort.spill.percent = 60%
16. Job execution time vs io.sort.mb for io.sort.spill.percent = 70%
17. Job execution time vs io.sort.mb for io.sort.spill.percent = 80%
18. Job execution time vs io.sort.mb for io.sort.spill.percent = 90%
19. Job execution time vs io.sort.spill.percent for io.sort.mb = 100mb
20. Job execution time vs io.sort.spill.percent for io.sort.mb = 120mb
21. Job execution time vs io.sort.spill.percent for io.sort.mb = 140mb
22. Job execution time vs io.sort.spill.percent for io.sort.mb = 160mb
23. Job execution time vs io.sort.spill.percent for io.sort.mb = 180mb
24. Job execution time vs io.sort.spill.percent for io.sort.mb = 200mb
25. Job execution time vs io.sort.mb for io.sort.spill.percent = 50%
26. Job execution time vs io.sort.mb for io.sort.spill.percent = 60%
27. Job execution time vs io.sort.mb for io.sort.spill.percent = 70%
28. Job execution time vs io.sort.mb for io.sort.spill.percent = 80%
29. Job execution time vs io.sort.mb for io.sort.spill.percent = 90%
30. Job execution time vs io.sort.mb for io.sort.spill.percent = 100%
31. Job execution time vs io.sort.spill.percent for io.sort.mb = 100mb
32. Job execution time vs io.sort.spill.percent for io.sort.mb = 120mb
33. Job execution time vs io.sort.spill.percent for io.sort.mb = 140mb
34. Job execution time vs io.sort.spill.percent for io.sort.mb = 160mb
35. Job execution time vs io.sort.spill.percent for io.sort.mb = 180mb
36. Job execution time vs io.sort.spill.percent for io.sort.mb = 200mb
37. Correlation Matrix
38. Colored correlation matrix
39. Configuring SSH keys
40. Modified .bashrc file for Hadoop setup
41. Modified Hadoop-env.sh file for Hadoop setup
42. Modified core-site.xml for Hadoop setup
43. Modified mapred-site.xml for Hadoop setup
44. Modified hdfs-site.xml for Hadoop setup
45. Hadoop WordCount
46. Failed Hadoop Job Execution
47. Successful Hadoop Job Execution
48. Landing page of UI
49. Prediction page of UI
50. Accuracy of prediction model

**LIST OF TABLES**

1. Hadoop Parameters
2. Testcase table – War and Peace
3. Testcase table – Jane Eyre
4. Testcase table – Don Quixote
5. Testcase table – Pride and Prejudice
6. Testcase table – Training for Prediction

#### Contents

***Declaration i***

***Acknowledgements ii***

***Abstract iii***

***List of Figures***

***List of Tables***

**1** [**INTRODUCTION**](#Introduction)

* 1. [General Introduction](#general_intro) 12
  2. [Problem Statement](#problem_statement_1_2) 15
  3. [Objectives of the project](#objectives_1_3) 15
  4. [Project deliverables](#deliverables_1_3) 15
  5. [Current Scope](#current_scope_1_5) 16
  6. [Future Scope](#future_scope_1_6) 16

1. [**PROJECT ORGANIZATION**](#project_org_2)
   1. [Software Process Models](#soft_proc_models_2_1) 17
   2. [Roles and Responsibilities](#roles_resp_2_2) 18
2. [**LITERATURE SURVEY**](#lit_sur_3)

**3.1** [Introduction](#intro_3_1) 19

**3.2** [Related Works with the citation of the References](#related_works_3_2) 19

**3.3** [Conclusion of Survey](#conc_surv_3_3) 24

1. [**PROJECT MANAGEMENT PLAN**](#proj_man_plan_4)
   1. [Schedule of the Project](#schedule_4_1) 25
   2. [Risk Identification](#risk_ident_4_2) 27
2. [**SOFTWARE REQUIREMENT SPECIFICATIONS**](#srs_5)

**5.1** [Project Overview](#overview_5_1) 27

**5.2** [External Interface Requirements](#ex_int_req_5_2) 27

**5.2.1** [User Interfaces](#UI_5_2_1) 27

**5.2.2** [Hardware Interfaces](#hard_int_5_2_2) 28

**5.2.3** [Software Interfaces](#soft_int_5_2_3) 28

**5.2.4** [Functional Requirements](#func_req_5_2_4) 28

**5.2.2** [Non Functional Requirements](#non_func_req_5_2_5) 29

1. [**DESIGN**](#design_6)
   1. [Introduction](#intro_6_1) 29
   2. [Architecture Design](#arch_des_6_2) 29
      1. [Existing Architecture](#exist_arch_6_2_1) 29
      2. [Proposed Architecture](#proposed_arch_6_2_2) 31
   3. [Graphical User Interface](#gui_6_3) 31
   4. [Flow Diagram](#flow_diag_6_4) 32
   5. [Conclusion](#con_6_5) 32
2. [**IMPLEMENTATION**](#imple_7)
   1. [Tools Introduction](#tools_7_1) 33
   2. [Technology Introduction](#tech_intro_7_2) 34
   3. [Overall view of the project in terms of implementation](#overal_view_7_3) 34
      1. [For testing Zipf’s Law](#for_testing_zipfs_law_7_3_1) 34
      2. [For optimization of job execution time by tuning 35](#for_opti_of_job_exec_time_7_3_2)

[Hadoop parameters](#for_opti_of_job_exec_time_7_3_2)

* 1. [Explanation of Algorithm and how it is been implemented](#expl_algo_7_4) 35
  2. [Conclusion](#concl_7_5) 37

1. [**TESTING AND RESULTS**](#testing_res_8)
   1. [Introduction](#intro_8_1) 38
   2. [Testing Tools and Environment](#testing_tools_env_8_2) 38
   3. [Test cases](#test_cases_8_3) 38
2. [**CONCLUSION & SCOPE FOR FUTURE WORK**](#con_scope_9)
3. [**REFERENCES**](#refere_10)
4. [**Appendix**](#appendix_11)
5. [Software Manual](#soft_man_1)
6. **INTRODUCTION**
   1. **General Introduction**

Matt Aslett defined Big Data as “Big Data is now almost universally understood to refer to the realization of greater business intelligence by storing, processing, and analyzing data that was previously ignored due to limitation of traditional data management technologies.” Today, “Big Data” seems to be more than just a hype word. It is of real concern with data growing at enormous rate and with increasing demand of processing of the data so generated. In such a scenario, Big Data platforms like Hadoop come to the rescue. Tuning Hadoop performance appears inevitable for satisfying the present needs and the ever-increasing demands of tomorrow.

If Forbes is to be believed, for each person living, 1.7 megabytes of new information will be generated per second. This fact only reiterates the immediate need to direct our attention towards managing the humungous amount of data. Hadoop MapReduce is one important answer that comes to our rescue. Quoting another statistics presented by the Forbes, Hadoop market is predicted to grow at a compounded annual rate of 58%. So, optimizing the Hadoop framework is of utmost importance.

Though Hadoop MapReduce framework has been accepted whole heartedly in the industry but time and again we have come to question the efficiency of the paradigm and lot of research has been in progress to access and improve the performance of the framework as a whole as well its components. Here we describe briefly the components of the MapReduce architecture.

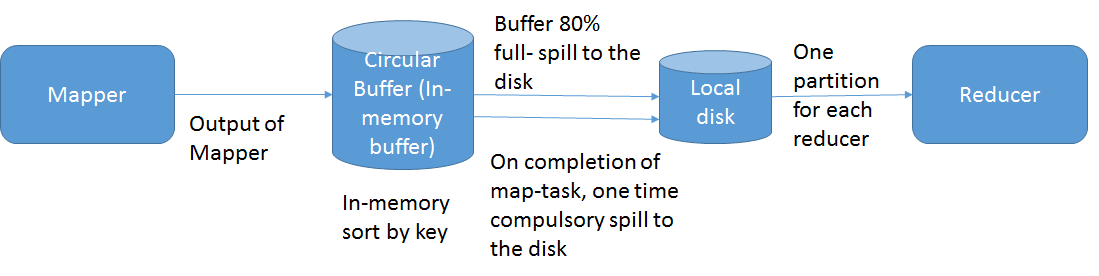


Figure 1: Hadoop MapReduce

The mapper maps the <key, value> input to intermediate <key, value> pairs and these are put into the circular buffer. The transformed intermediate output is not necessarily of the same form as that of the input. In the spilling phase, the map output is stored in an in-memory buffer and when the buffer is almost full, which is usually set to 80%, spilling to the local disk starts by a background thread. Spilling happens at least once- i.e. when the entire map-task is completed because all the partitions, one for each reducer, must be available on the disk.

Hadoop architecture uses the MapReduce paradigm and jobs on Hadoop can be run with different configurations that is fit for the current job in question.

Hadoop has over 190 parameters that can be configured for running jobs. Configuring the parameters lies solely in the hands of the user and thus most users tend to use Hadoop at its defaults. The following table shows few of the configuration parameters.

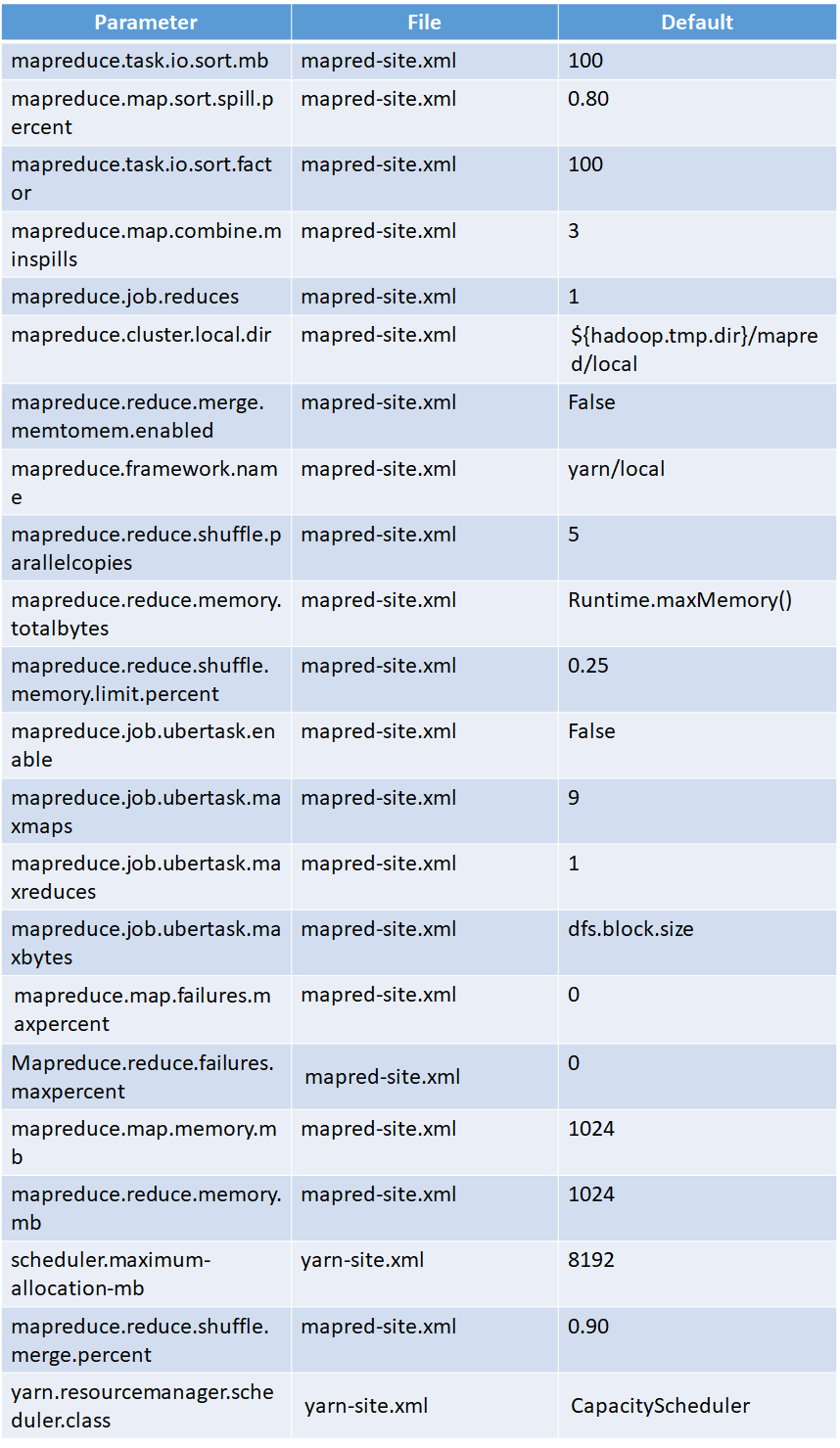


Table 1: Hadoop Parameters

* 1. **Problem Statement**

Hadoop MapReduce programming model provides for distributed computing which revolutionized the world of Big Data. With Hadoop becoming the household name in the technology world, it becomes utmost important to optimize the framework. In this project, we make distinct efforts in this regard. Firstly, we attempt to optimize the existing Hadoop architecture by facilitating correct tuning of configuration parameters relating to the shuffle phase. Once optimized, we also try to apply Zipf’s law for making smart decisions while spilling so as further introduce improvement in the job execution time.

.

* 1. **Objectives of the project**
* An attempt to compare the performance of Hadoop jobs by varying different Hadoop tuning parameters concerned with the Hadoop Shuffle phase.
* Also attempt to test for bias of Zipf’s law against different datasets of varying sizes which can be further used for Optimization of Shuffle phase of MapReduce paradigm.
  1. **Project Deliverables**
* A dataset generated by conducting experiments by varying the configuration parameters related to the shuffle phase of Hadoop – sort.io.mb, io.spill.percent, io.sort.factor and recording the job execution time for datasets of different sizes.
* Prediction of job execution times using different prediction models- Regression, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN).
* Visual test for bias of Zipf’s law against different datasets of different sizes and prove the applicability of Zipf’s law for optimizing the Hadoop shuffle phase.
  1. **Current Scope**

Big Data is changing the way how organizations in any field look at data. Be it the government, military or any other organization for that matter, everyone wants to harness the benefits of data insights. Big Data contributes in such scenarios and its contribution is conspicuous. With the importance of Big Data increasing at such tremendous rates, our project statement becomes very relevant. Optimizing the Hadoop architecture will directly impact a huge section of the industry that is heavily reliant on Apache Hadoop for its Big Data needs.

* 1. **Future Scope**

The future of Big Data shines bright and beautiful. Without Big Data any company is just impaired of its sense organs. It becomes deaf and blind with no insights into the data that is available to it for harnessing. With so much importance of the technology in question, it is bound that its optimization is of prime concern.

Optimizing the job execution time of any job is mission critical for industries using this technology to harness insights from terabytes of data. The proposed method in combination with other methods as proposed by different researchers across the world can be used to further optimize the existing MapReduce architecture of Hadoop.

Optimizing MapReduce, the universally accepted paradigm for BigData solution- Hadoop will benefit the entire technology society currently using Hadoop and also those that will use such technologies in the future. We see the sudden surge of Big Data customers shifting from Apache Hadoop to Apache Spark for advantages whose discussion is beyond the scope of this project. This project idea can very well be extended for implementation in Apache Spark to benefit a higher number of users of the Big Data technology with significant amount of gain in performance.

1. **PROJECT ORGANIZATION**
   1. **Software Process Models**

The software process model used in our project is the Waterfall model. The waterfall model represents a linear engineering software development model with its own advantages and pre-conditions.

Waterfall model belongs to that model of the SDLC (Software Development Life Cycle) in which the entire process is divided into phases and the output of the previous phase serves as input to the next phase.

Since the requirements for our project in hand is fixed and all the procedures are sequential, waterfall model suits our case the best. We firstly decided our problem statement after having vivid discussion among the teammates as well as our project guide, and then sorted out our software and hardware requirements based on the resources available to us.

We then contemplated on the datasets that would be used for the project and then started working on our implementation.

Implementation was divided into subtle modules each being taken up one after the other in a sequential fashion. Testing followed the completion of implementation.

In the end, the entire project process was documented.

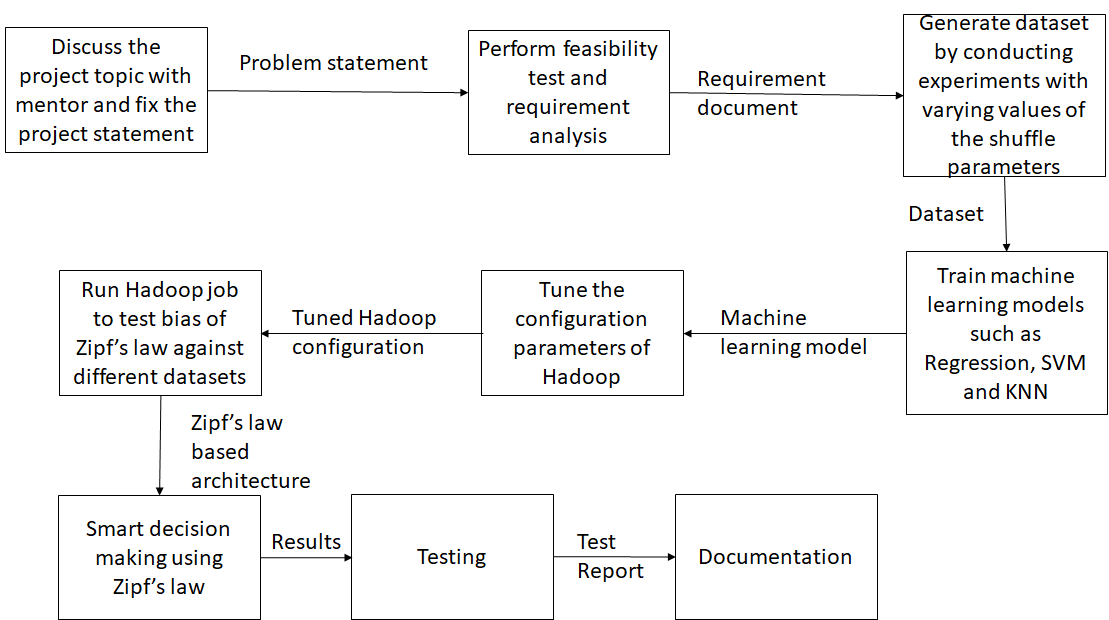


Figure 2: Waterfall Model for our project

* 1. **Roles and Responsibilities**

For any project to become successful, each and every team whatever the project they may be working on, the team must be well managed.

The team members must have a sense of optimism that together they can achieve everything against all odds and create the project with extreme interest.

We all took our roles and responsibilities very seriously and tried our level best to match each other’s interest in the project.

Each team member contributed significantly to the successful completion of the project. In the initial phases of the project, each team member individually surveyed certain number of research papers contributing to the literature survey.

The requirements and project management were handled by the members as a team.

Each team member made it a point to attend all team meetings and present valid ideas on the project progress.

The project was implemented jointly by all the team members each team member embarking on individual modules that were mutually decided.

It is remarkable that all the team members managed to sit together for the testing and verification of results using graphs.

The documentation work was equally divided among each of the teammates who did an incredible job coordinating and successfully completing the project.

1. **LITERATURE SURVEY**
   1. **Introduction**

Below we present a literature review of the relentless efforts of the researchers all over the world in optimizing the MapReduce paradigm especially the Shuffle phase.

* 1. **Related Works with the citation of the References**

**Hadoop and Shuffle Phase parameters:**

According to the paper, Optimizing Hadoop Parameters Based on Application resource Consumption [1], by Ziad Benslimane, performance of Hadoop is directly dependent on the number of map tasks that are launched as the number of reducers also depend on the former. Thus the main objective of the paper turns out to be finding the optimum number of map tasks that are to be launched.

The author makes a few assumptions- in a CPU-heavy applications, the number of parallel tasks should be less to avoid over-utilization of the CPU and large number of tasks should be spawned to distribute the load evenly among the CPUs. In CPU-light applications, large number of parallel tasks should be scheduled and less number of tasks should be spawned to reduce the overhead of each task.

The experiment consisted of two parts- determining if the task is CPU-heavy or CPU-light by running the job in a single node cluster and verifying the theoretical assumptions on multi-node cluster. The number of splits which is automatically decided by the Hadoop Map can be monitored using UNIX file split or cat. The test cases used for the experiment are Grep, PiEstimator and MRIF. Among these three, Grep and PiEstimator were found to be CPU-light tasks with CPU utilization of 35% and 4% respectively while MRIF being CPU-heavy task has 88% CPU utilization.

The theories that were proved in this paper are that in light CPU applications, the number of map tasks is less than equal to the number of CPUs and in case too many map tasks are to be run, run as many as possible in parallel. In case of CPU-heavy tasks, run at least as many map tasks as there are CPUs and the number of map tasks spawned in parallel should be less than or equal to two.

In the paper, Optimizing Hadoop for the Cluster [2], Hansen, the author claims that the default configuration of Hadoop is not always the most efficient. The number of mappers should be equal to the input size divided by the block size and the number of reducers is equal to 1.75 \* number of nodes \* maximum number of reducers that can run in parallel for best results. The results of the experiment follows that for WordCount application, the new configuration performed 2.5 times faster than the default Hadoop configuration and 1.7 faster than Cloudera’s configuration.

Karthik Kambatla, Abhinav Pathak & Himabindu Pucha, in their paper, Towards Optimizing Hadoop Provisioning in the Cloud [3], say that there is no hard and fast configuration rules that are applicable to all kinds of jobs. It suggests that to make proper utilization of the resources, a consumption profile history of the application is necessary.

The steps to generate the consumption profile history is enlisted as finding resource consumption by running the task against smaller number of nodes and dataset and then match the resource consumption signature with that of the signatures of the previous tasks stored in the database. The closest signature was assigned to the task.

The paper, Hadoop MapReduce Shuffle Phase Management [4], by Narendar Kumar, proposes that the time required for the shuffle phase can be reduced by using an SDN enabled structure. Having an SDN controller helps us control the network traffic and enable re-routing traffic through alternative routes thus reducing network congestion in the shuffle phase.

For benchmarking, the authors have used Hadoop Terasort. Different datasets of different sizes (2GB, 10GB, 20GB and 50GB) in a cluster of 16 nodes having an SDN based architecture including an SDN controller.

In the white paper, Advanced Hadoop Tuning and Optimizations [5], by Sanjay Sharma, it’s proposed that tuning of Hadoop configuration parameters is a black box art. It suggests users to change as many parameters as possible for better results. Among the many suggestions provided the most important ones include - mapred.compress.map.output should be set to TRUE for large clusters and large jobs, mapred.map/reduce.tasks.spectulative.execution should be set to FALSE if the average execution time is greater than 1 hour, mapred.tasktracker.map/reduce.tasks.maximum should be set in the range of [(cores per node) / 2 , 2 \* (cores per node)] for large clusters and the block size should be made larger for large datasets to reduce the number of map tasks spawned.

**Application of Zipf’s Law to Hadoop Shuffle Phase:**

In the paper, Speculative Executive Strategy Based on Node Classification and Hierarchy Index Mechanism for Heterogeneous Hadoop Systems [6], the authors propose to employ Node Classification along with a different hierarchical storage structure to keep the amount of time required for completion of map and reduce phases of a backup task to help in scheduling of tasks. Scheduling of tasks is based on an intuitive assumption that a backup task is selected to be scheduled only if the execution time of the backup task is at least 50% less than the remaining execution time of the currently running task. The authors of this paper claim that the new scheduling technique runs the Word Count problem 12 times faster than Hadoop original. They claim that because of hierarchical indexing of stored data, search time complexity is reduced i.e in case the data was stored as a list, then the time complexity would have been O(k) where k is the number of all items stored but because of hierarchical structure, the time complexity has been reduced to log 2 n+k/n where n in the number of nodes. Thus we can see that the time complexity has reduced to O(log 2 (n)).

In the paper, Optimizing MapReduce framework through Joint Scheduling of Overlapping phases [7], Huanyang Zheng, Ziqi Wan, and Jie Wu observe that map and shuffle tasks are different in nature. Map tasks are CPU intensive while Shuffle tasks are I/O intensive. Thus, they can be scheduled in parallel for effective utilization of resources. A new concept called strong pair has been introduced where a strong pair of jobs refers to two jobs that have same map and shuffle work load. A challenge faced in running Map and Shuffle phases in parallel is that it cannot be fully parallelized as shuffling depends on the map phase. If the shuffling rate is higher than that of the map phase, then the shuffle phase will have to wait for map intermediates to be generated. Thus to overcome such issues, a strong pair of jobs are selected such that when the shuffle phase of one job is running, map phase of the other can run forming a pipeline kind of architecture as time required for both of them is same in case of strong jobs. The authors have presented five algorithms- pair-based scheduling algorithm, couple based scheduling algorithm, generalized scheduling algorithm, group based scheduling algorithm and online group based scheduling algorithm based on the available job conditions.

In the paper, Reducing MapReduce Abstraction Costs for Text-Centric Applications [8], the authors have proposed two optimization techniques to improve MapReduce performance for text-centric tasks. The first approach, Frequency Buffering makes use of Zipf’s law to maintain an in-memory buffer which keeps track of frequently appearing keys during the shuffle phase. To find the frequent keys, the algorithm proposed by Metwally et al is used for profiling, which uses a table consisting of entries consisting of frequency numbers, and a linked list of keys which occur that many number of times. After profiling, the top k keys are predicted, reducing sort costs before the combine and reduce phases. If this optimization technique is implemented, 40% of the abstraction costs are reduced for WordCount, 30% for InvertedIndex, and 45% for WordPOSTag.

The second approach presented by the authors, is Spill Matcher which makes use of two types of threads (Support threads & Map threads). Here, the support threads are used to sort the data when a spill occurs, followed by combine() which writes output to intermediate local file. The map threads are used to merge sort the spilled tuples from different intermediate local files, following which combine() occurs before the reduce phase. To minimize the wait times between both these threads, we can make use of the bulk-service queuing model, which uses 2 assumptions: the producer threads fill the spill buffer as a Poisson process and the execution time for the consumer threads which process the spills are identically distributed random variables. If this approach is used, wait time for WordCount reduces by 90%, for InvertedIndex by 89%, for AccessLogSum by 77%, and by 83% for AccessLogJoin. WordPOSTag has negligible wait time in its slowest thread, and spill-matcher is not very effective for PageRank, cutting down the wait time by only 42% between the two threads.

Native Hadoop uses java based network transport stack on top of JVM for merging and shuffling phases. This proves to be a bottleneck in Hadoop framework. In the paper, JVM-Bypass for Efficient Hadoop Shuffling [9], the authors implemented a portable plug-in library for JVM-Bypassing that can help existing Hadoop framework to leverage TCP/IP protocol as well as RDMA protocol. This proposal supports RDMA protocol which was an inherent problem in Hadoop.

Yanfei Guo, Jia Rao, Dazhao Cheng, and Xiaobo Zhou, in their paper- iShufﬂe: Improving Hadoop Performance with Shufﬂe-on- Write [10], propose to separate the shuffle phase from the reduce phase and make it job-independent. It claims to solve the problem of skewed input data to the reducers as it predicts the size of the partitions to be fed to the reducers and also balances the distribution of the map outputs across the nodes. The iShuffle is designed to have 3 components- shuffler, shuffle manager, and task scheduler. For iShufle to function properly with minimal changes to the existing architecture, Shuffler is designed to be an independent component in the TaskTracker which takes the input from the combiner and shuffles the data. It performs shuffling each time the data is written to local disks by map tasks i.e shuffle-on- write.

* 1. **Conclusion of Survey**

It is seen that a lot of research has been undertaken in the field of tuning Hadoop parameters but it still remains a black box with no hard and fast rules concluded. This provides immense opportunity to explore more in this regard.

Zipf’s law also presents itself in light of optimizing Hadoop shuffle phase. If Zipf’s law can be tested for bias against different datasets and if it proves to have no bias against any particular dataset then the Hadoop shuffle phase can be designed in accordance to this statistical law and such a concept can also be extended to other popular Big Data platform like Apache Spark.

1. **PROJECT MANAGEMENT PLAN**
   1. **Schedule of the Project**

The following is the figure, which shows the schedule of our project, with all the activities involved that are listed and shown in the Gantt chart as per the Waterfall Software Process Model.

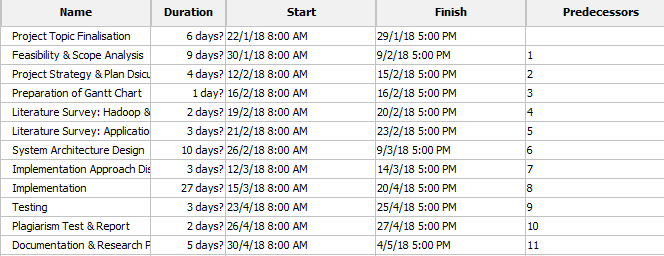


Figure 3: List of activities in the schedule

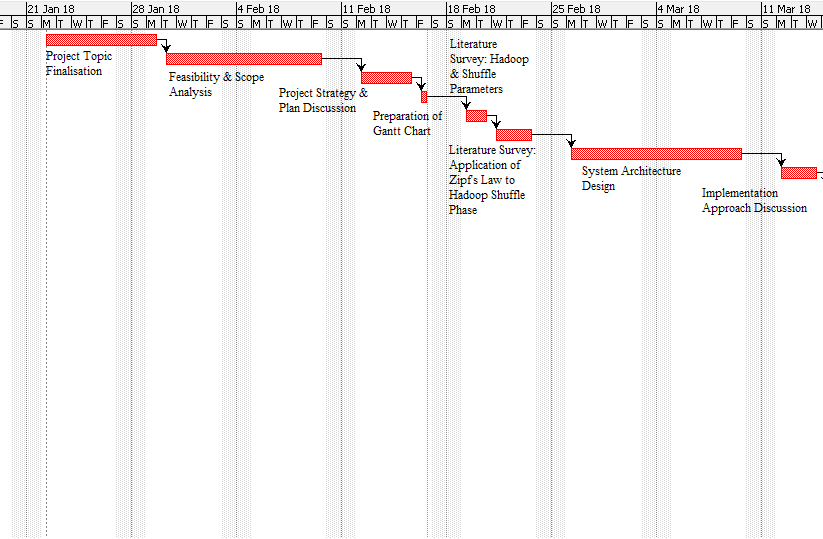


Figure 4: Gantt chart (Part 1)

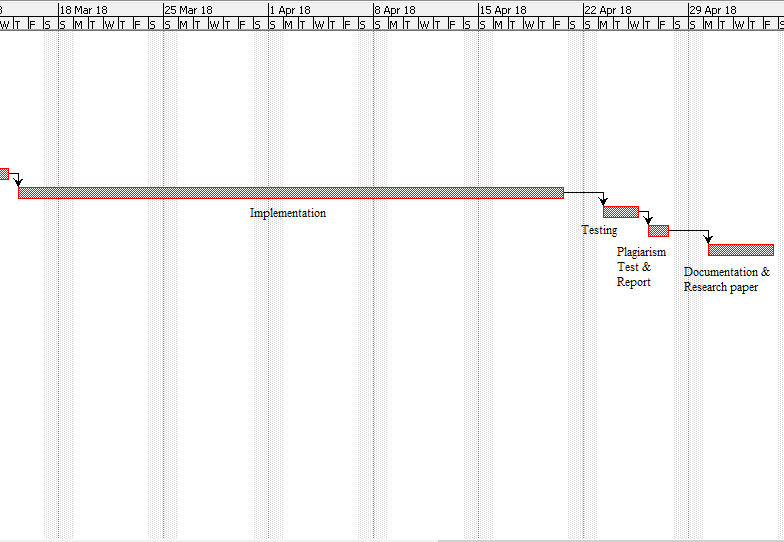


Figure 5: Gantt chart (Part 2)

\

* 1. **Risk Identification**

There are some situations where there is risk that project does not achieve its objectives. In this project main risk is the availability of good datasets and high quality hardware for testing the project.

Since it is a group project, selecting and implementing diverse ideas might be difficult. Work pressure and hurry to complete the other projects may lead to the poor quality of project delivery. The general risk of an error or omission in scope definition. Uncontrolled changes and continuous growth of scope. Project teammates may have different interpretations of the project topic.

1. **SOFTWARE REQUIREMENT SPECIFICATION**
   1. **Project Overview**

This project aims at optimizing the Apache Hadoop framework by fine tuning the existing architecture by precisely setting the configuration parameters related to the shuffle phase and then introducing a new approach to optimize the shuffle phase by applying Zipf’s law. To assist in achieving in such goals, we use technologies such as Apache Hadoop, Python, Java, and machine learning libraries like sklearn.

* 1. **External Interface Requirements**
     1. **User Interfaces**

The code is written in Java (Hadoop word count and sort) and in Python (sort and graph plotting using libraries like Matplotlib). For parameter tuning, Python Flask has been used for the UI and Python machine learning libraries - scikit, numpy, scipy, pandas have been used for the prediction model.

* + 1. **Hardware Interfaces**

All experiments are performed on a 4-node cluster. – one Intel core i5 processor 8GB RAM and three Intel core i3 processor 4GB RAM

All of these nodes must have –

* An Intel-compatible platform running Linux/Ubuntu. (Apache Hadoop compatible)
* At least 32 MB of RAM, a mouse, and enough disk space for storing the datasets and the results so obtained.
* The administrative privileges are required to install and run Apache Hadoop utilities under Linux/Ubuntu.
* A network connection for obtaining datasets from the Internet.
  + 1. **Software Interfaces**
* Apache Hadoop (version: 2.6.5).
* Python (version: 2.7.x).

Libraries - Matplotlib, Scikit, NumPy, SciPy, Pandas

* + 1. **Functional Requirements**
* Knowledge of cloud computing
* Knowledge of Unix OS
* Knowledge of Hadoop architecture
* Knowledge of MapReduce paradigm
* Scalability
* Platform compatibility
  + 1. **Non-Functional Requirements**
* Cost of transferring data
* Size of intermediate data
* Expected range of the configuration parameters and also the defaults for each of the parameter to be configured

1. **DESIGN**
   1. **Introduction**

For implementing our project, we have used Apache Hadoop Big Data platform. We have also used Python for sorting and getting our final results.

* 1. **Architecture Design**

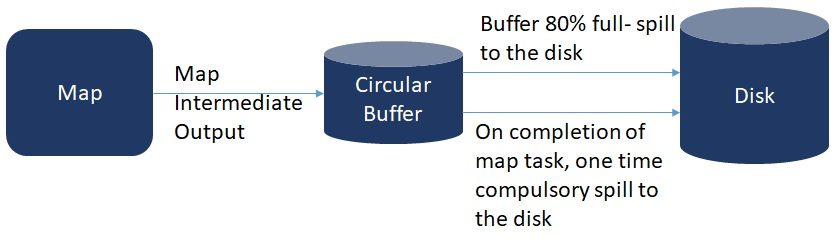
**6.2.1** **Existing Architecture:** 

Figure 6: Existing Architecture

In Hadoop MapReduce, after the Map phase and before the beginning of the Reduce phase is a handoff process, known as shuffle and sort. Here, data from the mapper tasks is prepared and moved to the nodes where the reducer tasks will be run. When the mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. During the reduce-shuffle phase, merging of intermediate results into an ordered file takes place after which the shuffling phase stops. During the shuffle phase, the intermediate map output is stored in the circular buffer with limited storage capacity (by default 100MB) and when the buffer is 80% full (this value is also configurable), the data is spilled onto the disk. Before the reduce phase starts, all the output is expected to be on the disk from where the reducer threads fetch the input for each of the reducers. This leads to a mandatory spill to the disk at the end of the shuffle phase.

The motivation for undertaking the project includes:

We see that spilling phase is a major hindrance to better performance because if the data is spilled more than once into the disk, 3 extra I/O needs to be performed to perform sorting by the key. The 3 I/O are writing onto the disk for the first time and then reading from and writing from the disk again to perform the sort and since disk access time is 6-7 times the buffer access time, spilling to the disk more than once is to be avoided to ameliorate the performance of this phase. It is also important to understand that if the spill thread is slower than the rate at which the map output is being produced, the circular buffer will get full and the map task has to be blocked until there is again some free space on the buffer.

In case the intermediate results are geographically distributed across different data centers, then data transferring is done through HTTP connections. This process is highly time-consuming as the bandwidth and topology affect remote file transfer.

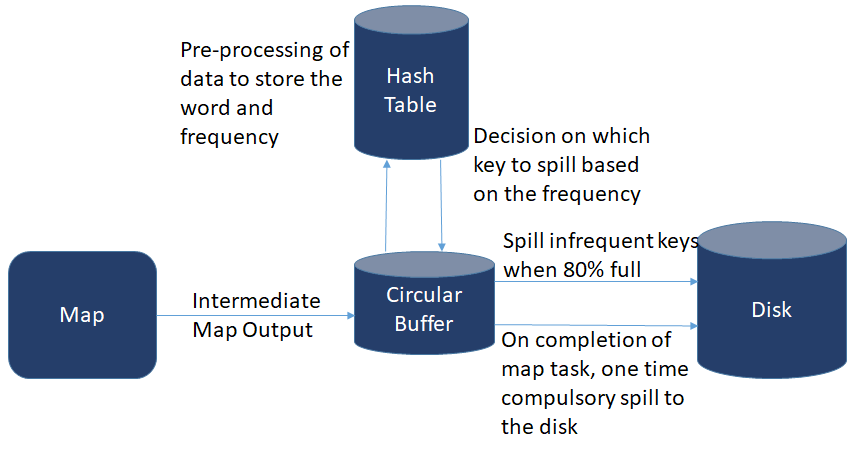
**6.2.2** **Proposed Architecture:** 

Figure 7: Proposed Architecture

In our undertaking of the current project, we propose a new design for the existing Hadoop architecture to use Zipf’s Law for gaining better performance in the shuffle phase. To make use of the frequency-rank relationship given by the Zipfian distribution, one needs to have prior knowledge of the text-centric application. If not so, the data needs to be preprocessed to gain an insight into the frequency of the keys appearing in the data. The formulating of an efficient algorithm for preprocessing of data is beyond the scope of the project. By using the prior knowledge of the data and the frequency of appearance of the keys in the data, smart decisions can be made as to which key is to be spilled onto the disk when the circular buffer is nearing saturation. The relatively infrequent keys are spilled.

* 1. **Graphical User Interface**

For the GUI, we have used Python Flask. The user gives four Hadoop Shuffle phase tuning parameter values as input, namely Dataset size, io.sort.mb, io.sort.spill.percent and io.sort.factor. The output will be the execution time that is predicted by the three prediction models – Regression, SVMs and KNN, will be displayed along with the accuracy of each model. Depending on the accuracy, the user can select the desired execution time from these 3 models.

* 1. **Flow Diagram**

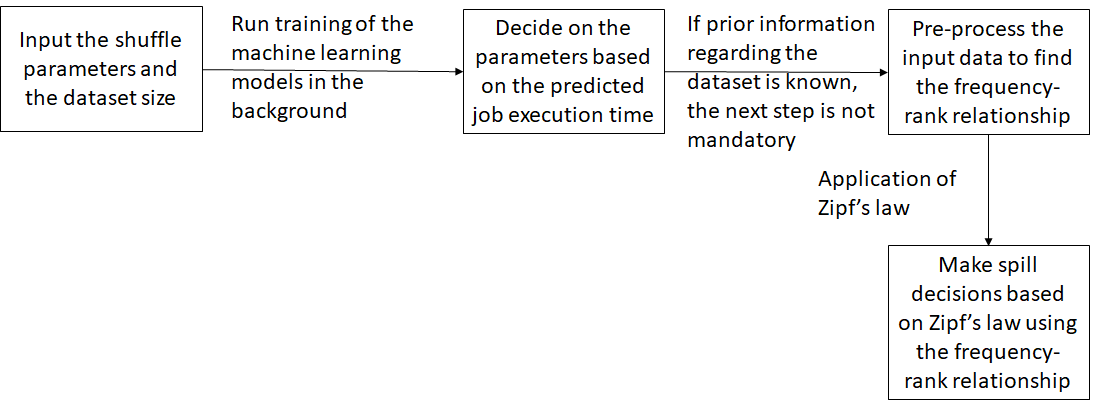


Figure 8: Flow Diagram

Step 1: The configuration parameters related to Hadoop Shuffle phase is inputted in the UI along with the dataset size.

Step 2: The generated dataset is used to train the machine learning prediction models namely Regression, Support Vector Machine and KNN. The model with greatest accuracy is considered and the parameters are tuned accordingly.

Step 3: The data is pre-processed to find the frequency-rank relationship. If such information is available to us before-hand, this step can be skipped.

Step 4: The relationship so obtained is used to make smart decisions while spilling in the shuffle phase.

* 1. **Conclusion**

Thus, we propose an efficient design that will make sure that the Shuffle phase of the Hadoop MapReduce paradigm is optimized by making use of Zipf’s Law for faster execution and also, making use of tuning parameters to make the job execution better.

1. **IMPLEMENTATION**
   1. **Tools Introduction**

The tools used by us in this project are namely SciPy, Python, Hadoop and Matplotlib.

* Python is a general purpose interpreted, interactive, object-oriented and high-level programming language.
* SciPy is a scientific python open source to perform Mathematical, Scientific and Engineering Computations. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization.
* Scikit is a free software machine learning library for Python users with in-built functions for prediction models like linear regression, SVM, KNN, etc.
* NumPy is a library for the [Python programming languag](https://en.wikipedia.org/wiki/Python_(programming_language))e, adding support for large, multi-dimensional [arrays](https://en.wikipedia.org/wiki/Array_data_structure) and [matrices](https://en.wikipedia.org/wiki/Matrix_(math)), along with a large collection of [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [mathematical](https://en.wikipedia.org/wiki/Mathematics) [functions](https://en.wikipedia.org/wiki/Function_(mathematics)) to operate on these arrays.
* Pandas is a [software library](https://en.wikipedia.org/wiki/Software_library) written for the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)) for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and [time series](https://en.wikipedia.org/wiki/Time_series). It is [free software](https://en.wikipedia.org/wiki/Free_software) released under the [three-clause BSD license](https://en.wikipedia.org/wiki/3-clause_BSD_license).
* Matplotlib is a plotting library for Python, used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It is very helpful in creating 2D plots.
* Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It’s designed to scale up from single servers to thousands of machines, each offering local computation and storage.
  1. **Technology Introduction**

The main technology we emphasize on in our project is Apache Hadoop and its MapReduce paradigm. Though Hadoop MapReduce framework has been accepted wholeheartedly in the industry but time and again we have come to question the efficiency of the paradigm and lot of research has been in progress to access and improve the performance of the framework as a whole as well its components.

The mapper maps the <key, value> input to intermediate <key, value> pairs and these are put into the circular buffer. The transformed intermediate output is not necessarily of the same form as that of the input. In the spilling phase, the map output is stored in an in-memory buffer and when the buffer is almost full, which is usually set to 80%, spilling to the local disk starts by a thread background thread. Spilling happens at least once- i.e. when the entire map-task is completed because all the partitions, one for each reducer, must be available on the disk.

* 1. **Overall view of the project in terms of implementation**

**7.3.1.** **For testing Zipf’s Law:**

The experiment was conducted on a 4 node Hadoop cluster. The datasets used for the purpose of verification of the concerned law are taken from Project Gutenberg comprising of English novels depicting the general language pattern. Four datasets of different sizes have chosen namely Pride and Prejudice, Don Quixote, War and Peace and Jane Eyre- the Autobiography. The datasets are fed to the WordCount MapReduce on the Hadoop cluster as input and the output so obtained are sorted in a descending fashion of the frequency of their occurrence. The highest frequency is used as reference for calculating the expected values in accordance to Zipf’s Law. Both the actual results and expected values are plotted on a graph to visually understand the deviation of the results obtained from the theoretical values.

**7.3.2** **For optimization of job execution time by tuning Hadoop parameters:**

The experiment was conducted on a 4 node Hadoop cluster. The datasets used were taken from Wikimedia Data Dumps. The three datasets are of sizes 1.2 GB, 832.4 MB and 2.1 GB respectively.

In our experiment, we vary the values of the Shuffle phase parameters - io.sort.mb (between 100 to 200), io.sort.spill.percent (between 50 to 100) and io.sort.factor (either 10 or 11). Along with the above mentioned 3 parameters, data size also becomes a parameter to obtain the job execution time. The Hadoop job execution time is recorded for each run after the alteration is done for the respective values.

By doing this we have generated datasets which contain the Hadoop parameters and execution time as different columns. These datasets are used to train ML models like Multiple Regression and SVMs, which later predict the Hadoop job execution time given the different parameter values provided by the user through the GUI.

Thus, we can approximately predict the job execution time depending on the tuning performed on the different Shuffle phase parameters.

* 1. **Explanation of Algorithm and Implementation**

For our approach where we suggest using Zipf’s Law using a Hash Table during the Shuffle phase, the proposed algorithm assumes that –

1. There is only a single time spill from the circular buffer to the disk for simplicity.
2. The frequency rank relationship of all the keys are presumed to be known (pre-processing of input data is required).
3. The number of keys to be kept in the buffer is entirely intuitional and is assumed to be 5% of the total keys available.
4. It is also known that the buffer access time is much lower than that of the disk access time.

The algorithm, with such assumptions, mathematically proves that the job execution time in a traditional system is much higher than the one proposed.

We have tried out all mathematical calculations that are best suited for these applications, and found out that our approach has greater advantage than the traditional system.

Our assumptions must always be satisfied for the algorithm to work and for us to show what advantage out proposed idea / algorithm has upon the traditional methods.

The algorithm is proposed as follows –

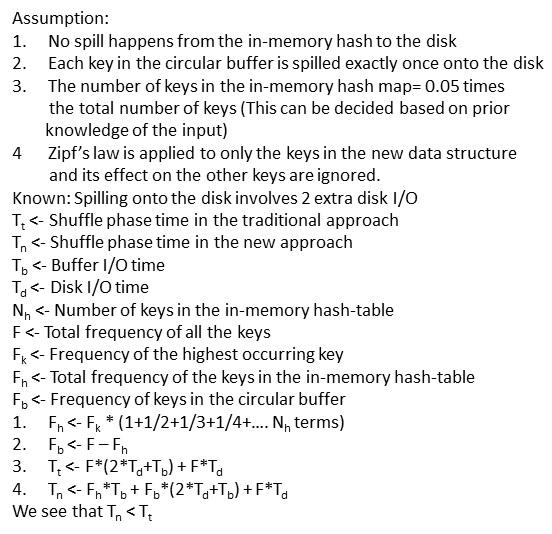


Figure 9: Proposed Algorithm

* 1. **Conclusion**

From the obtained results, it is seen that Zipf’s law is followed by all the datasets to maximum extent, and the correlation matrix shows the relationship between different parameters, including the execution time itself. The Machine Learning models used can help us predict the Hadoop job execution time when a set of parameters are given.

1. **TESTING AND RESULTS**
   1. **Introduction**

The Zipf’s law has to be tested for each dataset individually to ensure that the distribution proposed is not biased towards a particular dataset. Even though testing is usually automated in the industry, manual testing can suffice the purpose for small projects like this. Similar approach is used for parameter tuning.

* 1. **Testing Tools and Environment**

We have used Manual Testing as well as graph based testing for Zipf’s law verification.

Manual testing is the process of manually [testing software](https://en.wikipedia.org/wiki/Software_testing) for defects. It requires a tester to play the role of an end user whereby they use most of the application's features to ensure correct behavior. To guarantee completeness of testing

Expected results are computed using the Zipfian distribution and the obtained results are tested against the pre-computed expected results. Hence, visual testing can suffice.

* 1. **Test cases**

The datasets used for the verification of Zipf’s Law purpose are obtained from Project Gutenberg. Four datasets of different sizes have chosen namely Pride and Prejudice (727 KB), Don Quixote (2.4 MB), War and Peace (3.4 MB) and Jane Eyre- The Autobiography (1.2 MB).

Since the datasets used are extremely huge and incorporating all the keys in the dataset in a tabular format becomes a challenge here and thus the tables below show the 20 most frequent keys- their actual frequency and expected frequency according Zipf’s Law.

Dataset: War and Peace- 3.4 MB

|  |  |
| --- | --- |
| **Expected Frequency** | **Actual Frequency** |
| 31704 | 31704 |
| 15852 | 20564 |
| 10568 | 16320 |
| 7926 | 14855 |
| 6340 | 10018 |
| 5284 | 8228 |
| 4529 | 7631 |
| 3963 | 7630 |
| 3522 | 7230 |
| 3170 | 7188 |
| 2882 | 5520 |
| 2642 | 5305 |
| 2438 | 4411 |
| 2264 | 4201 |
| 2113 | 3882 |
| 1981 | 3694 |
| 1864 | 3368 |
| 1761 | 3325 |
| 1668 | 3270 |
| 1585 | 3225 |

Table 2: Testcase table – War and Peace

Dataset: Jane Eyre- The Autobiography - 1.2 MB

|  |  |
| --- | --- |
| **Expected Frequency** | **Actual Frequency** |
| 7455 | 7455 |
| 3727 | 6424 |
| 2485 | 6179 |
| 1863 | 5072 |
| 1491 | 4402 |
| 1242 | 4261 |
| 1065 | 2643 |
| 931 | 2396 |
| 828 | 2159 |
| 745 | 2046 |
| 677 | 1543 |
| 621 | 1503 |
| 573 | 1434 |
| 532 | 1429 |
| 497 | 1419 |
| 465 | 1399 |
| 438 | 1386 |
| 414 | 1385 |
| 392 | 1243 |
| 372 | 1209 |

Table 3: Testcase table – Jane Eyre

Dataset: Don Quixote - 2.4 MB

|  |  |
| --- | --- |
| **Expected Frequency** | **Actual Frequency** |
| 20910 | 20910 |
| 10455 | 16599 |
| 6970 | 13486 |
| 5227 | 12853 |
| 4182 | 7162 |
| 3485 | 7002 |
| 2987 | 6853 |
| 2613 | 5753 |
| 2323 | 5640 |
| 2091 | 4534 |
| 1900 | 4516 |
| 1742 | 4253 |
| 1608 | 4197 |
| 1493 | 3523 |
| 1394 | 3511 |
| 1306 | 3423 |
| 1230 | 3419 |
| 1161 | 3069 |
| 1100 | 2712 |
| 1045 | 2541 |

Table 4: Testcase table – Don Quixote

Dataset: Pride and Prejudice - 727 KB

|  |  |
| --- | --- |
| **Expected Frequency** | **Actual Frequency** |
| 4205 | 4205 |
| 2102 | 4121 |
| 1401 | 3662 |
| 1051 | 3309 |
| 841 | 1945 |
| 700 | 1858 |
| 600 | 1813 |
| 525 | 1795 |
| 467 | 1740 |
| 420 | 1419 |
| 382 | 1356 |
| 350 | 1306 |
| 323 | 1209 |
| 300 | 1167 |
| 280 | 1126 |
| 262 | 1119 |
| 247 | 1040 |
| 233 | 1038 |
| 221 | 1003 |
| 210 | 987 |

Table 5: Testcase table – Pride and Prejudice

For parameter tuning, the datasets used were taken from Wikimedia Data Dumps. The three datasets are of sizes 1.2 GB, 832.4 MB and 2.1 GB respectively.

The following table shows all the tests that were run for the purpose of training the prediction models to predict the job execution time based on the shuffle parameters – io.sort.mb, io.sort.spill.percent and io.sort.factor. The execution time predicted will be in terms of milliseconds (ms).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Size (MB) | io.sort.mb (MB) | io.sort.spill.percent (%) | io.sort.factor | Execution Time (ms) |
| 832.4 | 100 | 50 | 10 | 668 |
| 832.4 | 100 | 60 | 10 | 706 |
| 832.4 | 100 | 70 | 10 | 754 |
| 832.4 | 100 | 80 | 10 | 660 |
| 832.4 | 100 | 90 | 10 | 553 |
| 832.4 | 100 | 100 | 10 | 5293 |
| 832.4 | 120 | 50 | 10 | 723 |
| 832.4 | 120 | 60 | 10 | 741 |
| 832.4 | 120 | 70 | 10 | 751 |
| 832.4 | 120 | 80 | 10 | 762 |
| 832.4 | 120 | 90 | 10 | 686 |
| 832.4 | 120 | 100 | 10 | 5726 |
| 832.4 | 140 | 50 | 10 | 610 |
| 832.4 | 140 | 60 | 10 | 685 |
| 832.4 | 140 | 70 | 10 | 872 |
| 832.4 | 140 | 80 | 10 | 746 |
| 832.4 | 140 | 90 | 10 | 930 |
| 832.4 | 140 | 100 | 10 | 596 |
| 832.4 | 160 | 50 | 10 | 776 |
| 832.4 | 160 | 60 | 10 | 766 |
| 832.4 | 160 | 70 | 10 | 806 |
| 832.4 | 160 | 80 | 10 | 777 |
| 832.4 | 160 | 90 | 10 | 815 |
| 832.4 | 160 | 100 | 10 | - |
| 832.4 | 180 | 50 | 10 | 651 |
| 832.4 | 180 | 60 | 10 | 643 |
| 832.4 | 180 | 70 | 10 | 754 |
| 832.4 | 180 | 80 | 10 | 694 |
| 832.4 | 180 | 90 | 10 | 662 |
| 832.4 | 180 | 100 | 10 | 404 |
| 832.4 | 200 | 50 | 10 | 697 |
| 832.4 | 200 | 60 | 10 | 807 |
| 832.4 | 200 | 70 | 10 | 698 |
| 832.4 | 200 | 80 | 10 | 672 |
| 832.4 | 200 | 90 | 10 | 708 |
| 832.4 | 200 | 100 | 10 | - |
| 1200 | 100 | 50 | 10 | 930 |
| 1200 | 100 | 60 | 10 | 930 |
| 1200 | 100 | 70 | 10 | 1103 |
| 1200 | 100 | 80 | 10 | 1031 |
| 1200 | 100 | 90 | 10 | 1022 |
| 1200 | 100 | 100 | 10 | - |
| 1200 | 120 | 50 | 10 | 1084 |
| 1200 | 120 | 60 | 10 | 1085 |
| 1200 | 120 | 70 | 10 | 1002 |
| 1200 | 120 | 80 | 10 | 1101 |
| 1200 | 120 | 90 | 10 | 1023 |
| 1200 | 120 | 100 | 10 | - |
| 1200 | 140 | 50 | 10 | 955 |
| 1200 | 140 | 60 | 10 | 106 |
| 1200 | 140 | 70 | 10 | 1096 |
| 1200 | 140 | 80 | 10 | 1069 |
| 1200 | 140 | 90 | 10 | 1013 |
| 1200 | 140 | 100 | 10 | - |
| 1200 | 160 | 50 | 10 | 1144 |
| 1200 | 160 | 60 | 10 | 1109 |
| 1200 | 160 | 70 | 10 | 1194 |
| 1200 | 160 | 80 | 10 | 1193 |
| 1200 | 160 | 90 | 10 | 1013 |
| 1200 | 160 | 100 | 10 | - |
| 1200 | 180 | 50 | 10 | 1075 |
| 1200 | 180 | 60 | 10 | 975 |
| 1200 | 180 | 70 | 10 | 1041 |
| 1200 | 180 | 80 | 10 | 1028 |
| 1200 | 180 | 90 | 10 | 1081 |
| 1200 | 180 | 100 | 10 | - |
| 1200 | 200 | 50 | 10 | 1093 |
| 1200 | 200 | 60 | 10 | 1163 |
| 1200 | 200 | 70 | 10 | 1093 |
| 1200 | 200 | 80 | 10 | 1041 |
| 1200 | 200 | 90 | 10 | 1116 |
| 1200 | 200 | 100 | 10 | - |
| 2100 | 100 | 50 | 10 | 1876 |
| 2100 | 100 | 60 | 10 | 1649 |
| 2100 | 100 | 70 | 10 | 1904 |
| 2100 | 100 | 80 | 10 | 1672 |
| 2100 | 100 | 90 | 10 | 1847 |
| 2100 | 100 | 100 | 10 | - |
| 2100 | 120 | 50 | 10 | 1841 |
| 2100 | 120 | 60 | 10 | 1781 |
| 2100 | 120 | 70 | 10 | 1692 |
| 2100 | 120 | 80 | 10 | 1931 |
| 2100 | 120 | 90 | 10 | 1794 |
| 2100 | 120 | 100 | 10 | - |
| 2100 | 140 | 50 | 10 | 1859 |
| 2100 | 140 | 60 | 10 | 1781 |
| 2100 | 140 | 70 | 10 | 1765 |
| 2100 | 140 | 80 | 10 | 1841 |
| 2100 | 140 | 90 | 10 | 1755 |
| 2100 | 140 | 100 | 10 | - |
| 2100 | 160 | 50 | 10 | 2090 |
| 2100 | 160 | 60 | 10 | 2017 |
| 2100 | 160 | 70 | 10 | 1012 |
| 2100 | 160 | 80 | 10 | 2049 |
| 2100 | 160 | 90 | 10 | 2103 |
| 2100 | 160 | 100 | 10 | - |
| 2100 | 180 | 50 | 10 | 1725 |
| 2100 | 180 | 60 | 10 | 1695 |
| 2100 | 180 | 70 | 10 | 1753 |
| 2100 | 180 | 80 | 10 | 1794 |
| 2100 | 180 | 90 | 10 | 1714 |
| 2100 | 180 | 100 | 10 | - |
| 2100 | 200 | 50 | 10 | 1878 |
| 2100 | 200 | 60 | 10 | 1817 |
| 2100 | 200 | 70 | 10 | 1888 |
| 2100 | 200 | 80 | 10 | 1850 |
| 2100 | 200 | 90 | 10 | 1834 |
| 2100 | 200 | 100 | 10 | - |
| 1200 | 100 | 50 | 11 | 751 |
| 1200 | 100 | 60 | 11 | 606 |
| 1200 | 100 | 70 | 11 | 759 |
| 1200 | 100 | 80 | 11 | 549 |
| 1200 | 100 | 90 | 11 | 664 |
| 1200 | 100 | 100 | 11 | - |
| 1200 | 120 | 50 | 11 | 736 |
| 1200 | 120 | 60 | 11 | 742 |
| 1200 | 120 | 70 | 11 | 750 |
| 1200 | 120 | 80 | 11 | 725 |
| 1200 | 120 | 90 | 11 | 762 |
| 1200 | 120 | 100 | 11 | - |
| 1200 | 140 | 50 | 11 | 805 |
| 1200 | 140 | 60 | 11 | 776 |
| 1200 | 140 | 70 | 11 | 815 |
| 1200 | 140 | 80 | 11 | 816 |
| 1200 | 140 | 90 | 11 | 681 |
| 1200 | 140 | 100 | 11 | - |
| 1200 | 160 | 50 | 11 | 776 |
| 1200 | 160 | 60 | 11 | 781 |
| 1200 | 160 | 70 | 11 | 820 |
| 1200 | 160 | 80 | 11 | 782 |
| 1200 | 160 | 90 | 11 | 804 |
| 1200 | 160 | 100 | 11 | - |
| 1200 | 180 | 50 | 11 | 676 |
| 1200 | 180 | 60 | 11 | 772 |
| 1200 | 180 | 70 | 11 | 811 |
| 1200 | 180 | 80 | 11 | 657 |
| 1200 | 180 | 90 | 11 | 829 |
| 1200 | 180 | 100 | 11 | - |
| 1200 | 200 | 50 | 11 | 834 |
| 1200 | 200 | 60 | 11 | 823 |
| 1200 | 200 | 70 | 11 | 905 |
| 1200 | 200 | 80 | 11 | 651 |
| 1200 | 200 | 90 | 11 | 931 |
| 1200 | 200 | 100 | 11 | - |

Table 6: Testcase table – Training for Prediction

**For Zipf’s Law implementation:**

The following are the graphs plotted for each of the datasets taken. We can see here that our output (plotted in red) closely follows the theoretical output (plotted in blue).

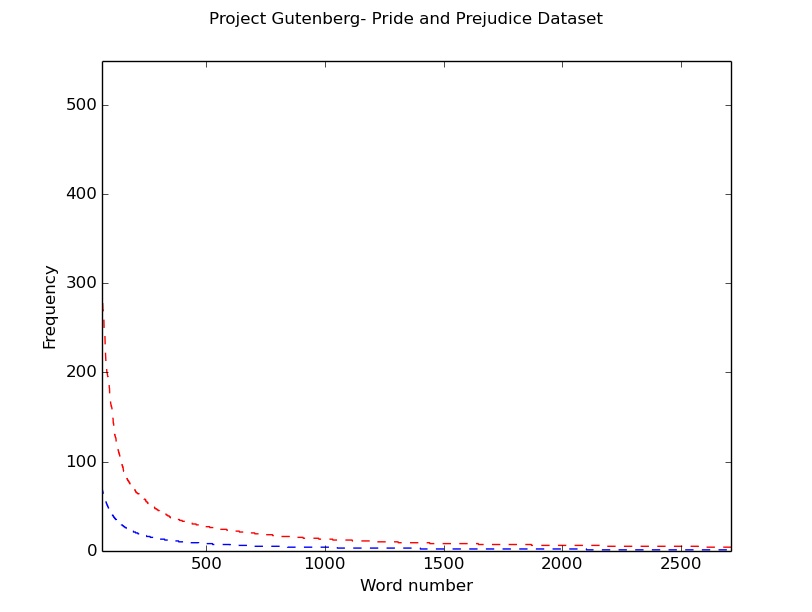


Figure 10: Pride & Prejudice Dataset

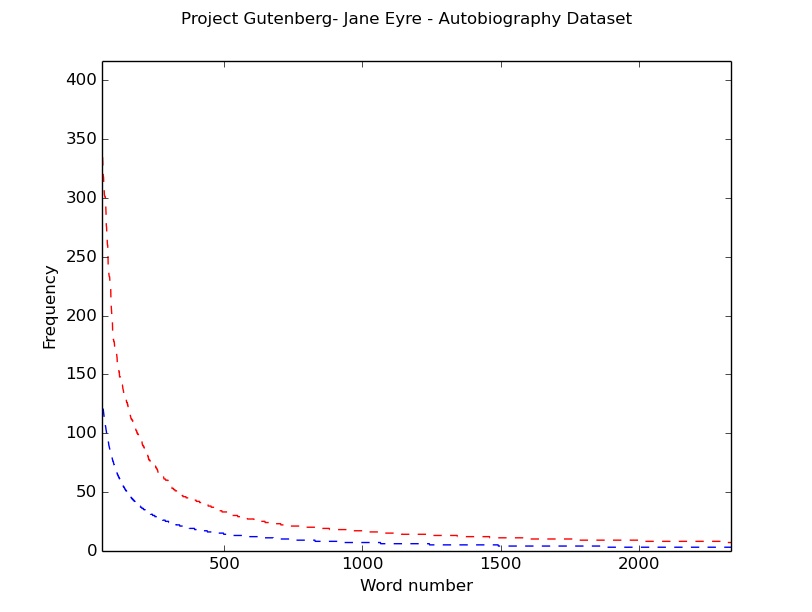


Figure 11: Jane Eyre – Autobiography Dataset

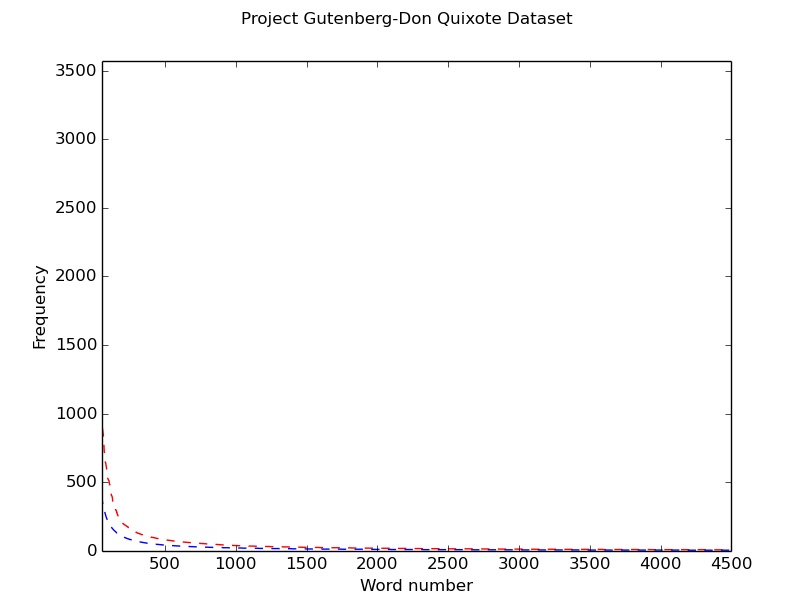


Figure 12: Don Quixote Dataset

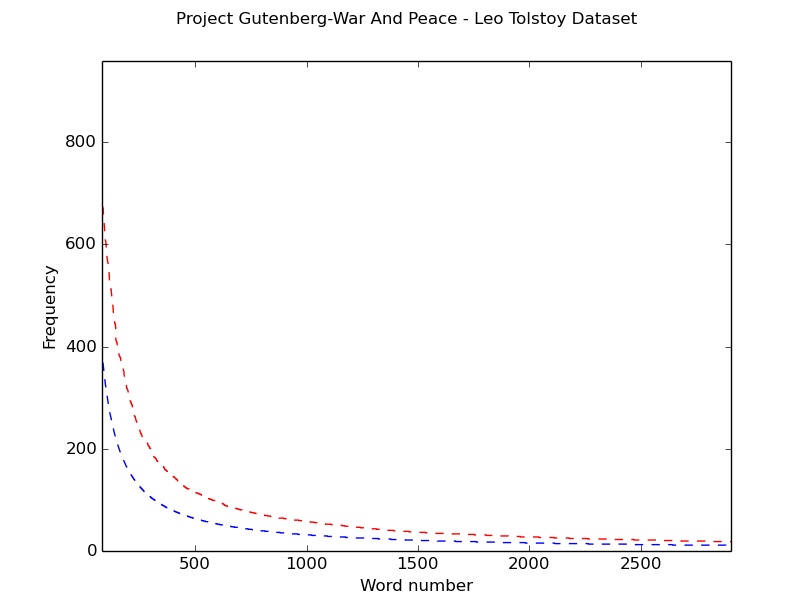


Figure 13: War & Peace Dataset

The datasets are fed to the WordCount MapReduce on the Hadoop cluster as input and the output so obtained are sorted in a descending fashion of the frequency of their occurrence. The highest frequency is used as reference for calculating the expected values in accordance to Zipf’s Law. Both the actual results and expected values are plotted on a graph to visually understand the deviation of the results obtained from the theoretical values. It is seen that the results obtained are very close to the expected values based on the Zipf’s law.

**For Performance comparison of Hadoop jobs based on Shuffle Parameters:**

We’ve plotted many graphs, out of which a few are displayed here.

The graphs try to show the dependency between different parameters.

For dataset size: 1.2 GB

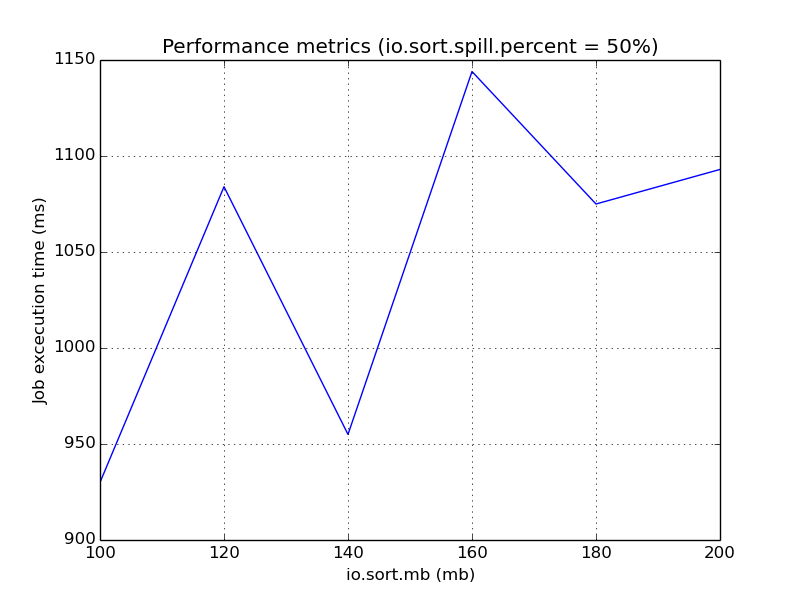


Figure 14: Job execution time vs io.sort.mb for io.sort.spill.percent = 50%

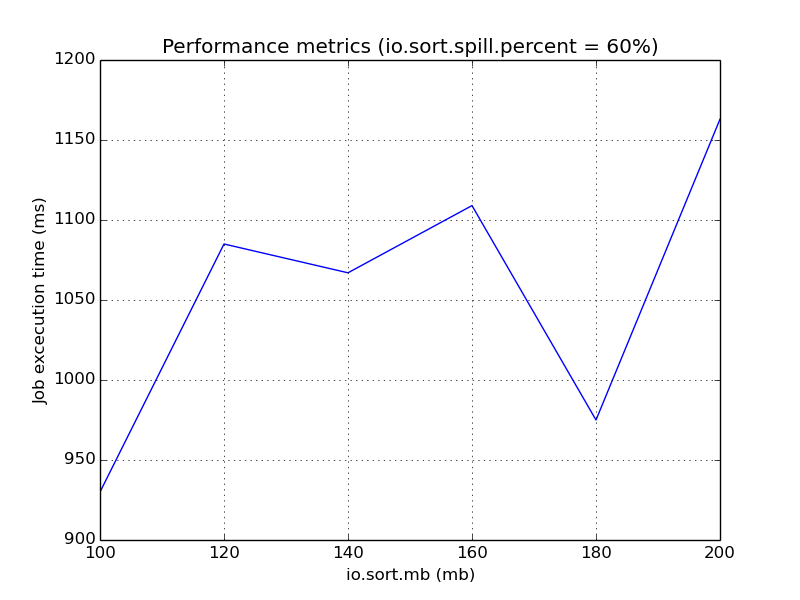


Figure 15: Job execution time vs io.sort.mb for io.sort.spill.percent = 60%

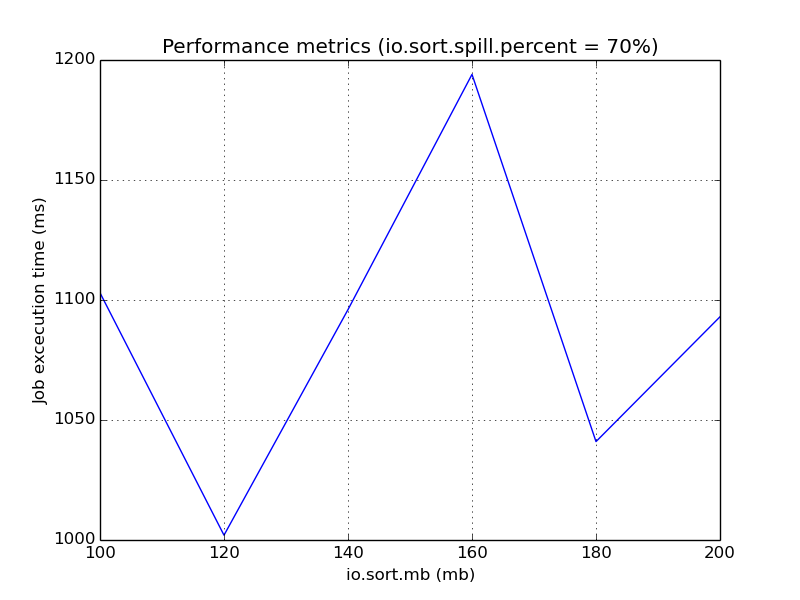


Figure 16: Job execution time vs io.sort.mb for io.sort.spill.percent = 70%

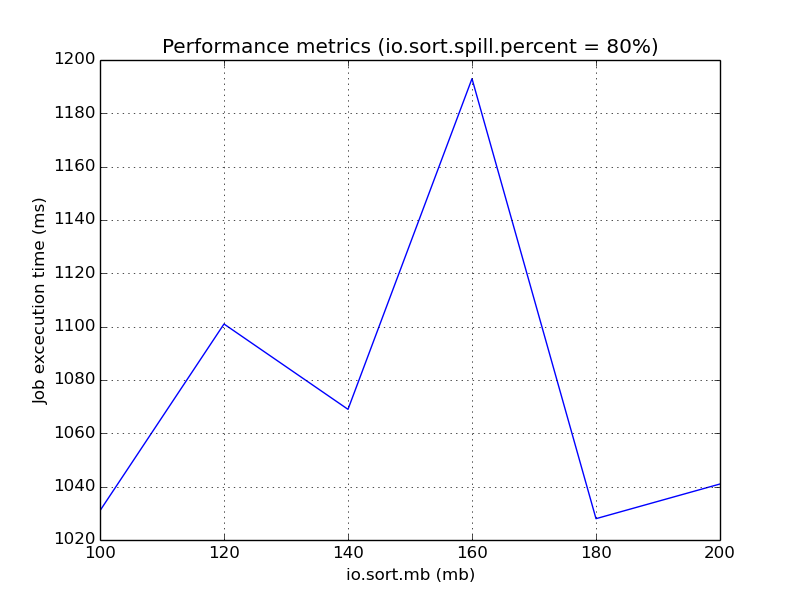


Figure 17: Job execution time vs io.sort.mb for io.sort.spill.percent = 80%

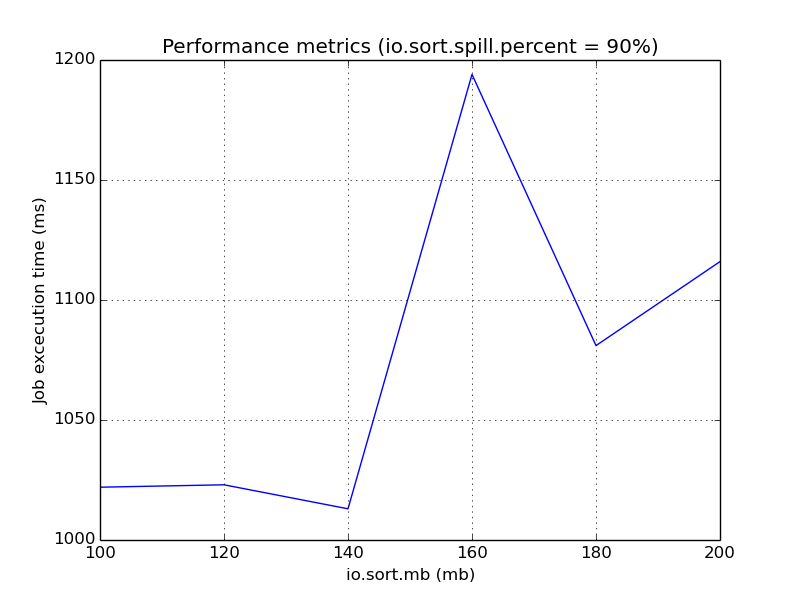


Figure 18: Job execution time vs io.sort.mb for io.sort.spill.percent = 90%

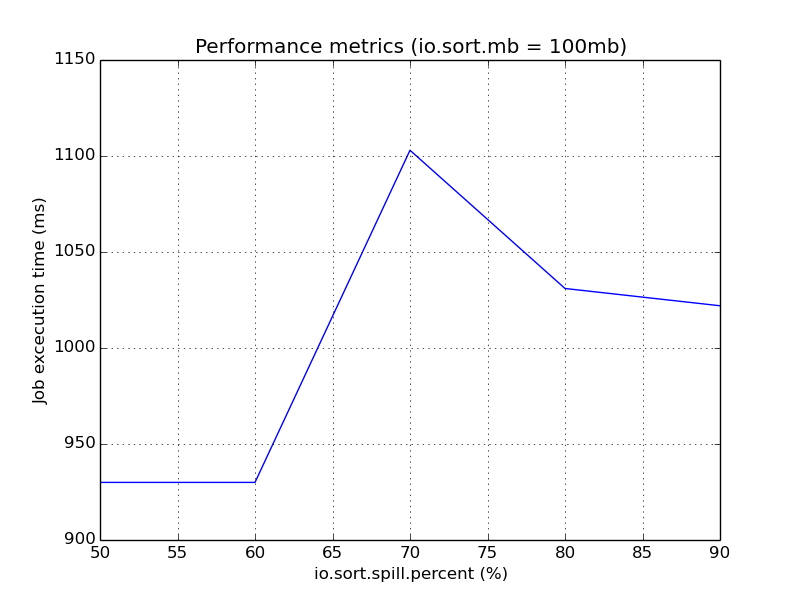


Figure 19: Job execution time vs io.sort.spill.percent for io.sort.mb = 100mb

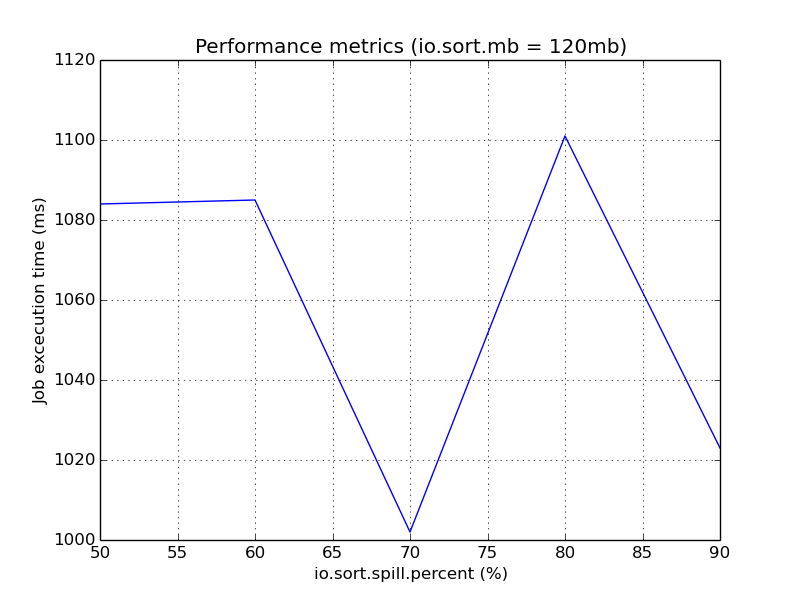


Figure 20: Job execution time vs io.sort.spill.percent for io.sort.mb = 120mb

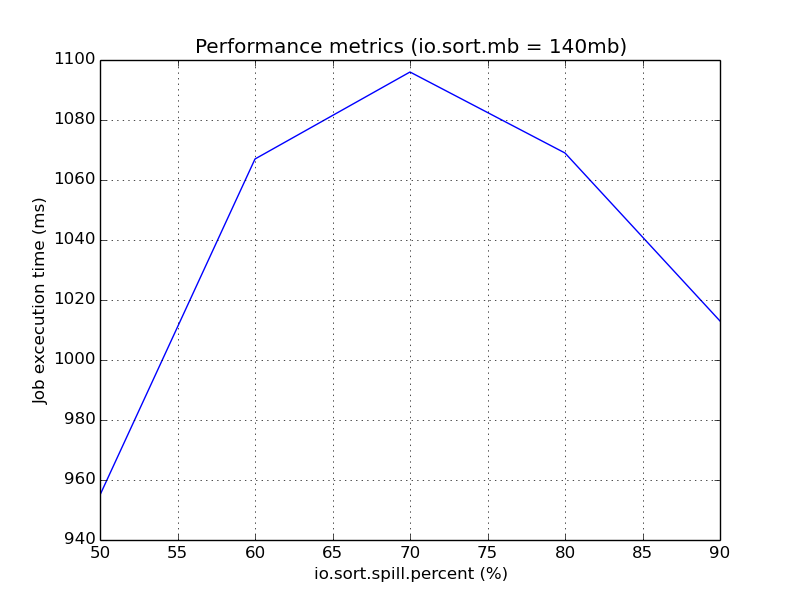


Figure 21: Job execution time vs io.sort.spill.percent for io.sort.mb = 140mb

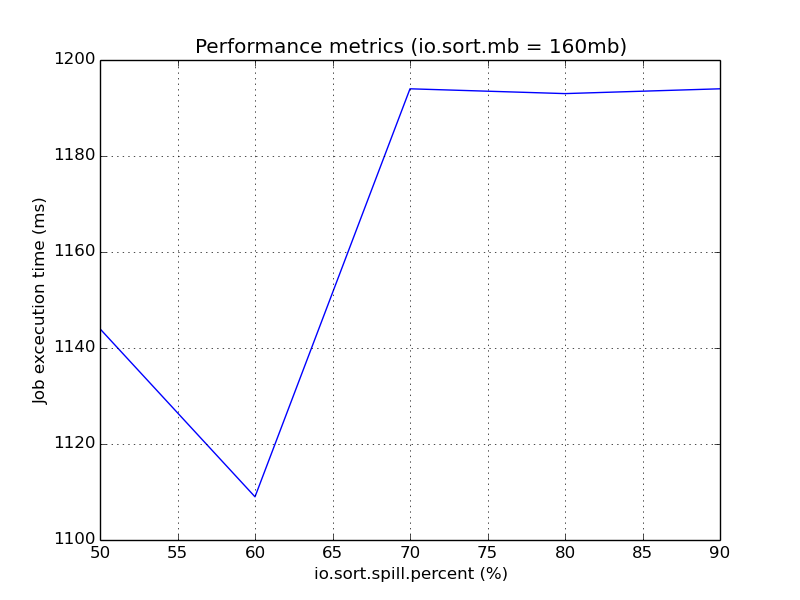


Figure 22: Job execution time vs io.sort.spill.percent for io.sort.mb = 160mb



Figure 23: Job execution time vs io.sort.spill.percent for io.sort.mb = 180mb

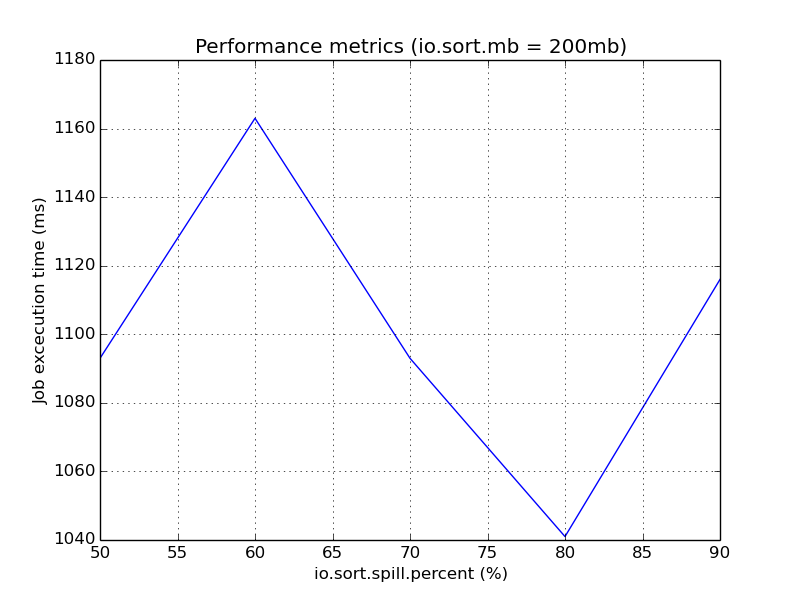


Figure 24: Job execution time vs io.sort.spill.percent for io.sort.mb = 200mb

For dataset size: 832.4 MB

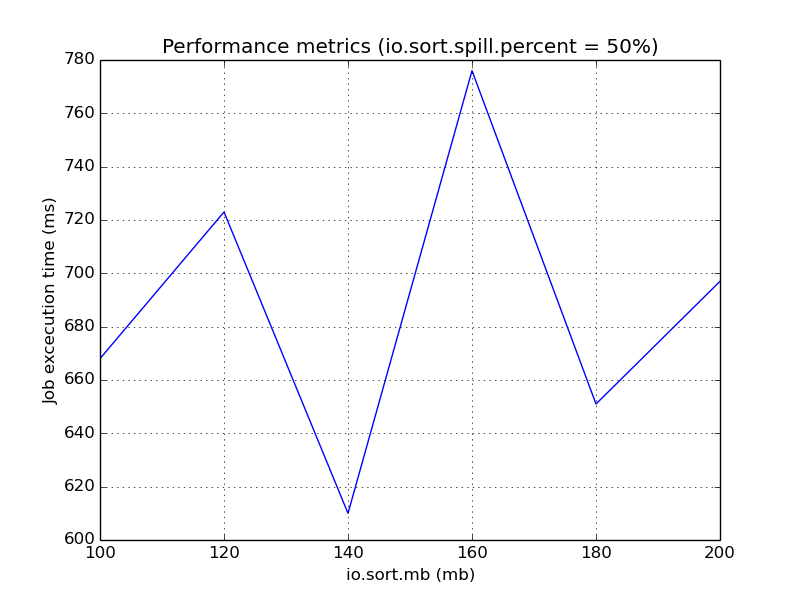


Figure 25: Job execution time vs io.sort.mb for io.sort.spill.percent = 50%

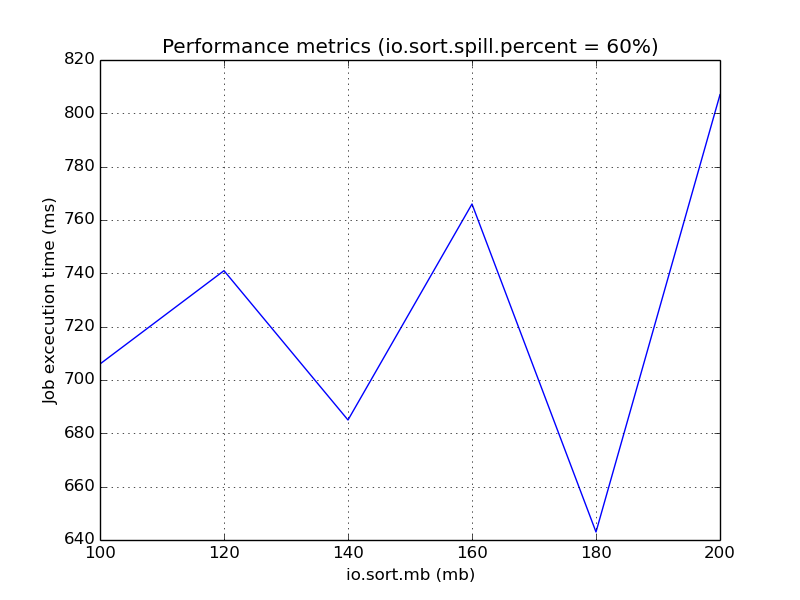


Figure 26: Job execution time vs io.sort.mb for io.sort.spill.percent = 60%

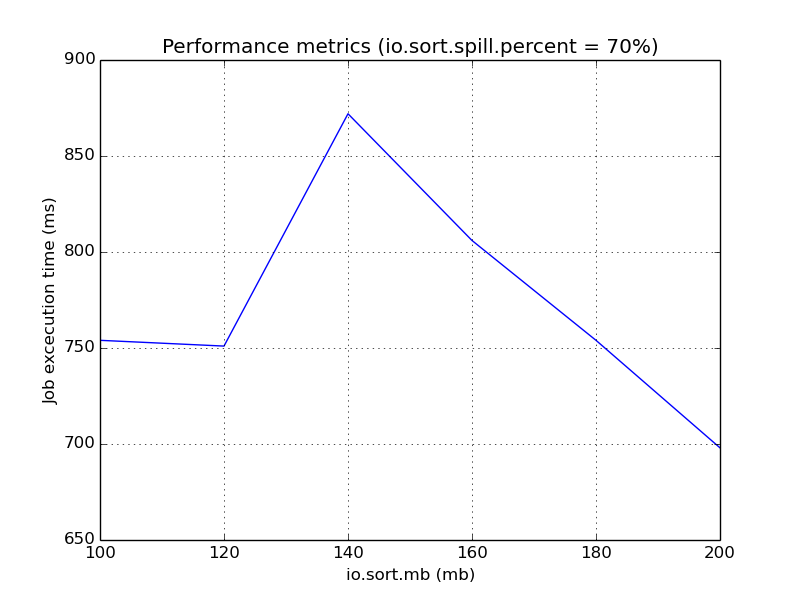


Figure 27: Job execution time vs io.sort.mb for io.sort.spill.percent = 70%

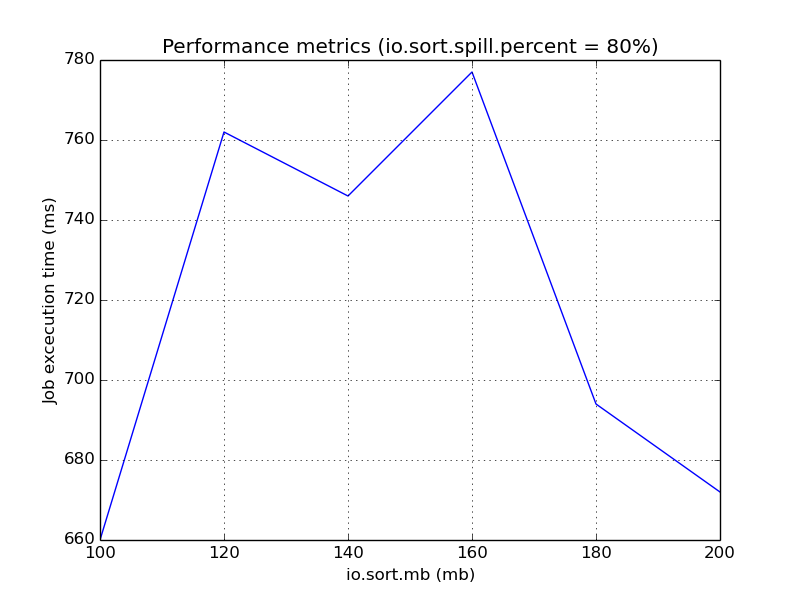


Figure 28: Job execution time vs io.sort.mb for io.sort.spill.percent = 80%

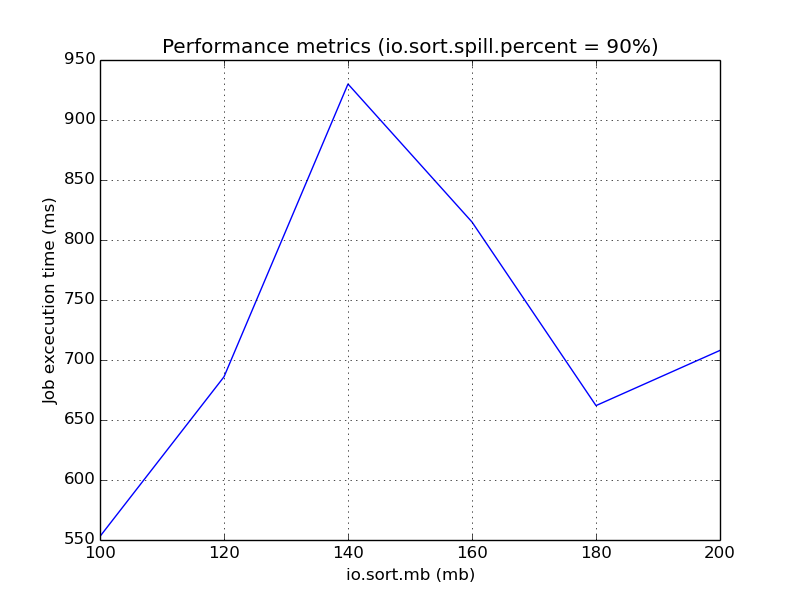


Figure 29: Job execution time vs io.sort.mb for io.sort.spill.percent = 90%

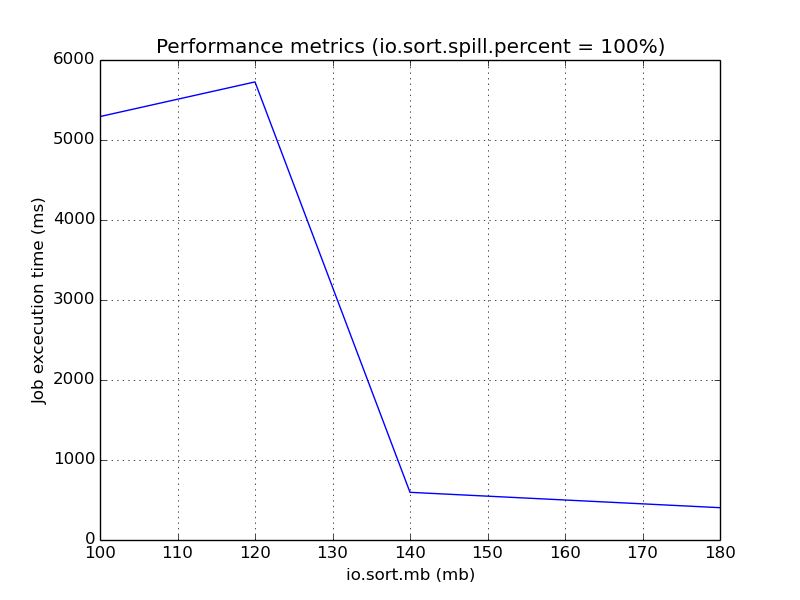


Figure 30: Job execution time vs io.sort.mb for io.sort.spill.percent = 100%

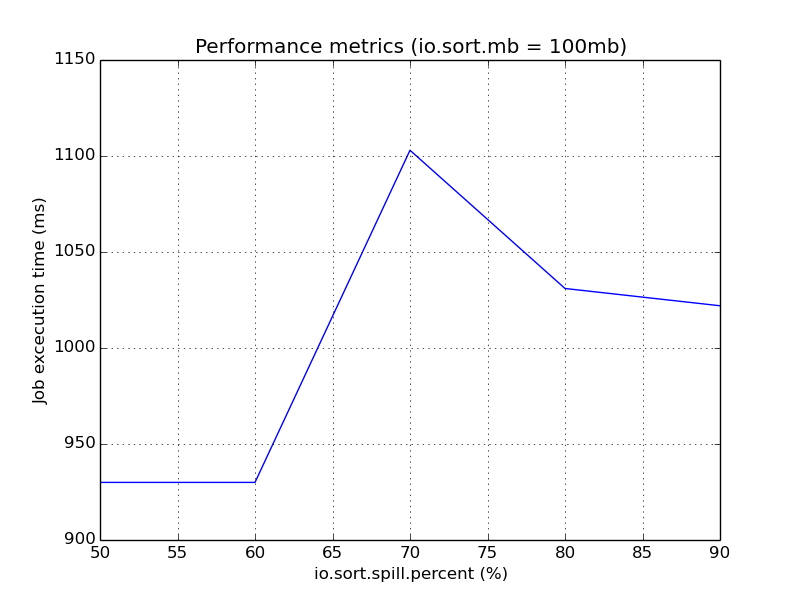


Figure 31: Job execution time vs io.sort.spill.percent for io.sort.mb = 100mb

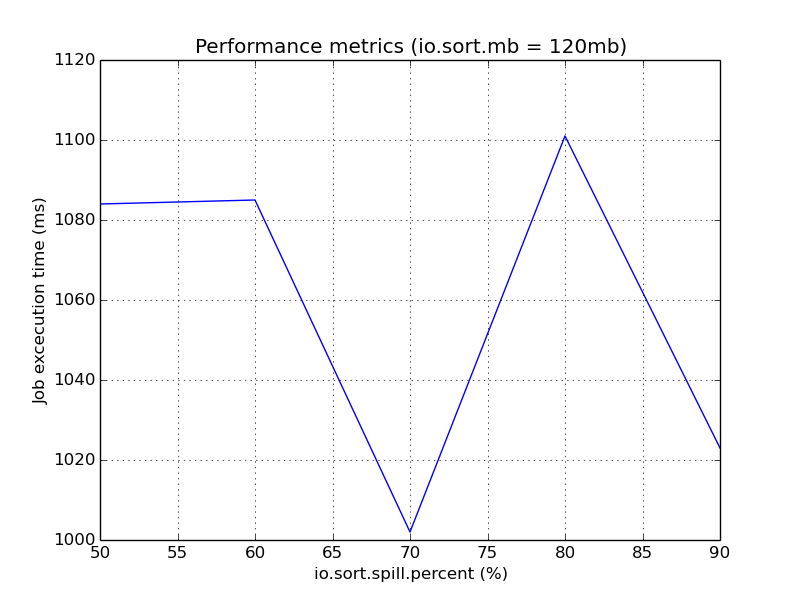


Figure 32: Job execution time vs io.sort.spill.percent for io.sort.mb = 120mb

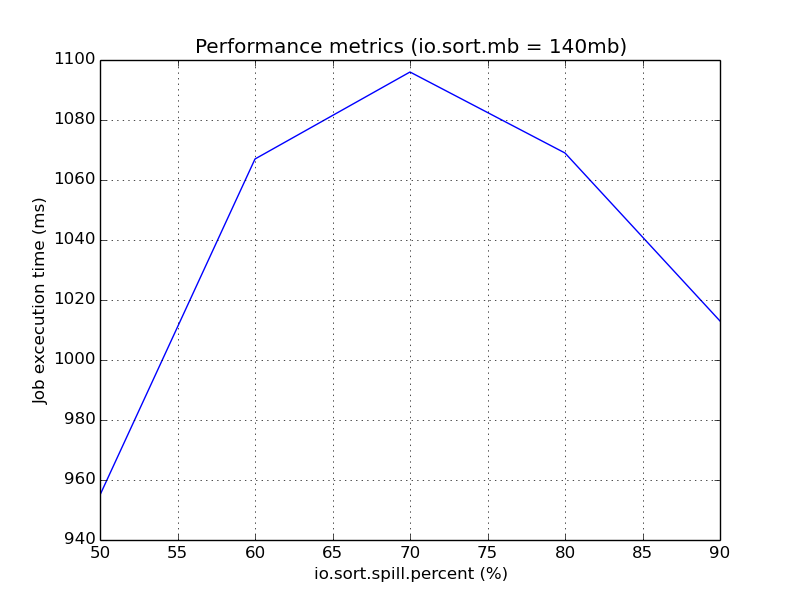


Figure 33: Job execution time vs io.sort.spill.percent for io.sort.mb = 140mb

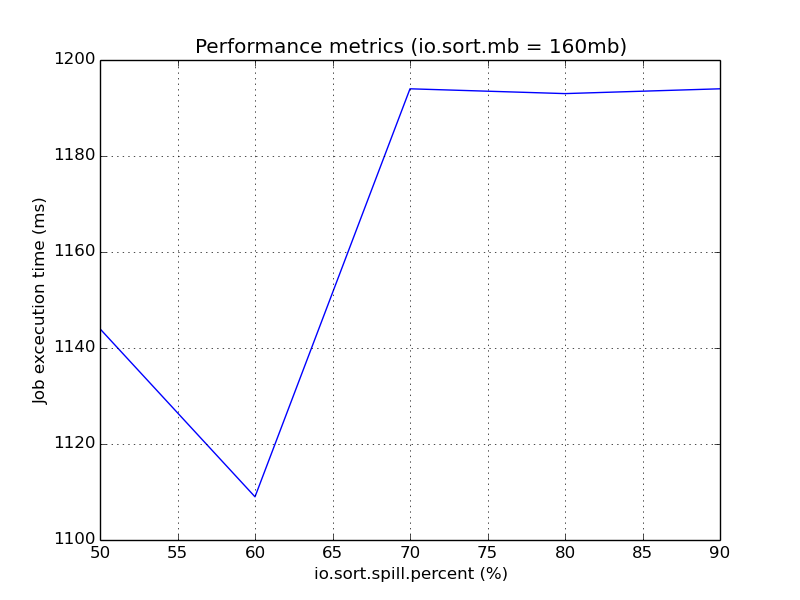


Figure 34: Job execution time vs io.sort.spill.percent for io.sort.mb = 160mb



Figure 35: Job execution time vs io.sort.spill.percent for io.sort.mb = 180mb

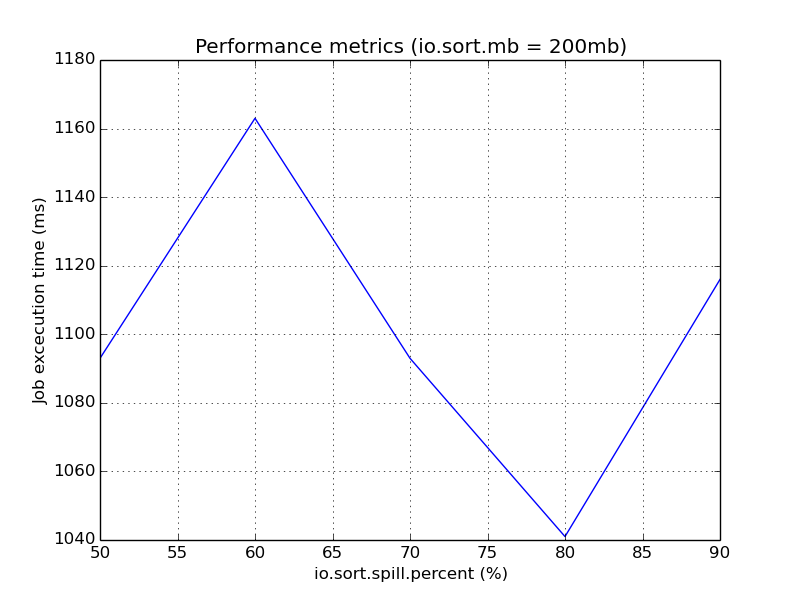


Figure 36: Job execution time vs io.sort.spill.percent for io.sort.mb = 200mb

It is seen from the graphs plotted that the performance of Hadoop is not always the best when it is used at defaults.

The following figure shows the correlation matrix of all the shuffle parameters in consideration.

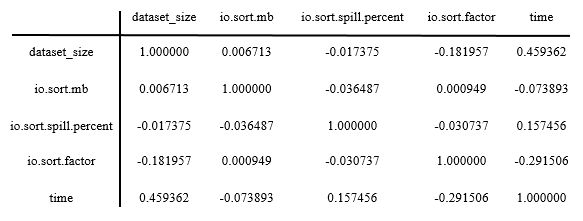


Figure 37: Correlation matrix

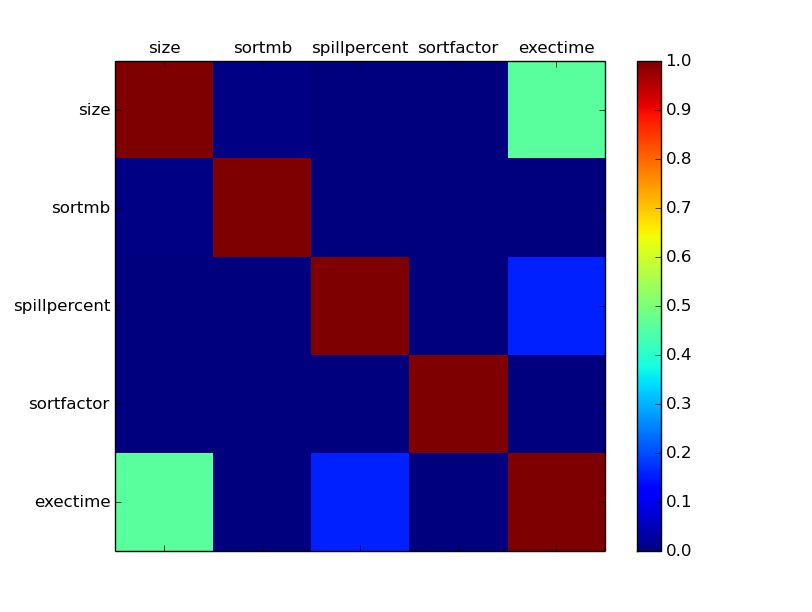


Figure 38. Colored Correlation Matrix

The colored correlation matrix shows correlation in a visually appealing way. The correlation matrix shows the relationship among all the parameters as to how the other parameter values would vary if any one parameter value is changed. For example, if the correlation between parameter A and B is 0.5, it means that if A is increased by 1 point, B increases by 0.5 points. Similarly, if the correlation value is -0.5, it means that if A is increased by 1 point, B decreases by 0.5 points. The correlation matrix shown above how the shuffle parameters, dataset size and job execution time are related to each other.

1. **CONCLUSION & SCOPE FOR FUTURE WORK**

We are striving to make a prediction of the (approximate) job execution time given the Hadoop tuning parameters and the dataset size using different machine learning techniques (This prediction may not be exactly correct because it also depends the map-reduce business logic but our efforts lies in approximation so that users have some idea about tuning and not use Hadoop like a black box). We also propose to suggest the most optimal tuning parameters for a dataset of the given size.

We see that the datasets used approximately follow Zipf’s Law, and thus this law could be used to optimize the Shuffle phase of Hadoop MapReduce. Later this optimization could even be extended to Apache Spark.

1. **REFERENCES**
2. Ziad Benslimane. Optimizing Hadoop Parameters Based on Application Resource Consumption. [Online]. Available: http://uu.diva-portal.org/smash/get/diva2:622285/FULLTEXT01.pdf
3. Hansen. Optimizing Hadoop for the Cluster. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.486.1265&rep=rep1&type=pdf
4. Karthik Kambatla, Abhinav Pathak & Himabindu Pucha. Towards Optimizing Hadoop Provisioning in the Cloud. [Online]. Available: https://pdfs.semanticscholar.org/1a19/bab56d8ae4a325b650de71cd1d908c7bd715.pdf
5. Narendar Kumar. Hadoop MapReduce Shuffle Phase Management. [Online]. Available: https://www.talentica.com/white-papers/Hadoop-Map-Reduce-Shuffle-Phase-Management.pdf
6. Sanjay Sharma. Advanced Hadoop Tuning and Optimization. [Online]. Available: http://www.impetus.com/impetusweb/whitepapers\_main.jsp?download=HadoopPerformanceTuning.pdf
7. Qi Liu, Weidong Cai, Jian Shen, Zhangjie Fu, Xiaodong Liu, Nigel Linge. Speculative Executive Strategy Based on Node Classification and Hierarchy Index Mechanism for Heterogeneous Hadoop Systems. [Online]. Available: http://icact.org/download/journal/Volume\_5\_Issue\_4.pdf
8. Huanyang Zheng, Ziqi Wan, and Jie Wu. Optimizing MapReduce framework through Joint Scheduling of Overlapping phases. [Online]. Available: https://ieeexplore.ieee.org/document/7568555/
9. Chun-Hung Hsiao, Michael Cafarella, Satish Narayanasamy. Reducing MapReduce Abstraction Costs for Text-Centric Applications. [Online]. Available: http://web.eecs.umich.edu/~nsatish/papers/ICPP-14-Hadoop.pdf
10. Yandong Wang, Cong Xu, Xiaobing Li, Weikuan Yu. JVM-Bypass for Efficient Hadoop Shuffling. [Online]. Available: https://www.cs.fsu.edu/~yuw/pubs/2013-IPDPS-Yu.pdf
11. Yanfei Guo, Jia Rao, Dazhao Cheng, Xiaobo Zhou. iShufﬂe: Improving Hadoop Performance with Shufﬂe-on-Write. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7506133/
12. Justin Ellingwood. An Introduction to Big Data Concepts & Terminology. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-big-data-concepts-and-terminology
13. JanBask Training. An introduction to the architecture and components of Hadoop Ecosystem. [Online]. Available: https://www.janbasktraining.com/blog/introduction-architecture-components-hadoop-ecosystem/
14. Apache Hadoop. MapReduce Tutorial. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred\_tutorial.html#Map+Parameters –
15. [Guru99 - What is MapReduce? [Online]. Available: https://www.guru99.com/introduction-to-mapreduce.html
16. Saman Biookaghazadeh. YARN Scheduler – Detailed Description. [Online]. Available: https://samanaghazadeh.wordpress.com/2016/05/05/yarn-scheduler-detailed-discription/
17. Hadoop Internals. Hadoop Configuration Parameters. [Online]. Available: http://ercoppa.github.io/HadoopInternals/HadoopConfigurationParameters.html
18. Wikipedia. Zipf’s Law. [Online]. Available: https://en.wikipedia.org/wiki/Zipf%27s\_law
19. Project Gutenberg. [Online]. Available: http://www.gutenberg.org/
20. Wikimedia Data Dumps. [Online]. Available: <https://dumps.wikimedia.org/>
21. Wikipedia – NumPy. [Online]. Available: <https://en.wikipedia.org/wiki/NumPy>
22. Wikipedia – Scikit. [Online]. Available: <https://en.wikipedia.org/wiki/Scikit-learn>
23. Wikipedia –SciPy. [Online]. Available: <https://en.wikipedia.org/wiki/SciPy>
24. Generalized Linear Models. [Online]. Available: http://scikit-learn.org/stable/modules/linear\_model.html
25. SciKit Learn - sklearn.svm.SVC. [Online]. Available: <http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
26. SciKit Learn- sklearn.neighbors.KNeighborsClassifier.
27. [Online]. Available: <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

**11.** **APPENDIX**

**1.**  **Software Manual**

Apache Hadoop can be run on two modes – Single Node Standalone cluster and Multinode Cluster

The difference between a single node cluster and a multinode cluster is that, in the former, Hadoop runs on one single machine and makes use of all its resources.

For the latter, the architecture is such that there is one Master node, which has control over the different slave nodes.

Typically, YARN (Yet Another Resource Negotiator) is responsible in both the configurations for managing the cluster.

The Resource Manager in YARN is generally located on the master in a multimode cluster and is required to allocate and manage the various resources like CPU, RAM, Memory etc between different slave nodes.

There exists a Node Manager on each node, where after the resource has been allocated to the application, the Application Master receives the container for execution of the MapReduce task on the slave node.

**To setup a Single node Cluster –**

The prerequisite required is that there must be a JDK installed on the machine for proper run of the Cluster.

A dedicated Hadoop user must be created onto the Ubuntu system that exists.

Install ssh by using sudo apt-get install ssh

Check if you have ssh by “which ssh” command.

Generate a public ssh key for the setting up of the cluster, and put it into the file containing the list of authorized keys for the remote communication on that particular node.

If you give in a command saying “ssh localhost” it must get you into your machine’s shell, or basically it would mean that ssh has been correctly configured and installed on the system.

Download Hadoop core distribution (tar.gz file) from –

http://www.eu.apache.org/dist/hadoop/core/

to get the necessary version you want to install.

Extract the file, and change the ownership to the dedicated Hadoop user that has been created already for Hadoop.

Modify the bash file for your system, and add the path of $JAVA\_HOME so as the node can recognize where java is on the system.

Modify Hadoop-env.sh as shown in the figure.

Modify all the 3 files – core-site.xml, hdfs-site.xml and mapred-site.xml as required.

Run Hadoop by using the command ./start-all.sh in the sbin folder within Hadoop directory.

**To setup a Multinode Cluster -**

The prerequisite for setting up a multinode cluster is that each node whether master or slave, must have single node Hadoop configured on the respective machine. Also make sure that each node has the same version of JDK installed for better compatibility.

Now, after the single node clusters have been set up, for running the multimode cluster there needs to be a way in which the slaves are going to connect with the master.

Before everything, the /etc/hosts file on each node must contain the IP address of each of the nodes along with their names, example –

172.102.13.15 master

172.102.13.14 slave1

Here comes in the SSH (Secure Shell) protocol which is a cryptographic protocol used to secure network connections and communications over an unsecured network like Hadoop Cluster.

The public key which is generated during installation for the Master, is distributed to the slaves and is copied into the file containing list of authorized keys for the slave.

In this way, the Master can SSH into the slave nodes. For example if we have a cluster with huser as the user on slave1 then the master can access the slave node by entering the following command into the shell - “ssh huser@slave1”

The master must be able to ssh into itself as well and that can be checked by typing “ssh master”.

After the SSH access has been verified, update the conf/masters file with master and conf/slaves file with the master and slave entries in the Hadoop system.

Following which, the 3 most important files that need to be updated in Hadoop folder are –

* Core-site.xml
* Hdfs-site.xml
* Mapred-site.xml

The core-site.xml file must have the name of the master entered instead of the localhost for the default parameter.

The hdfs-site.xml must be updated by the replication factor you want to give in, that is, it is by default 1, and needs to be changed to 2 or more depending on the number of nodes in the multimode cluster.

The mapred-site.xml file which has the map reduce job tracker must be updated from localhost to master so as the cluster can know the number of map tasks that are spawned.

Run Hadoop by using the command ./start-all.sh in the sbin folder within Hadoop directory on the slave nodes first, next on the master, to start the datanodes, that can me accessed by the namenode on the master node.

The following figures shows the Hadoop configuration of our multinode cluster.

This figure shows the configuring of ssh keys for Hadoop set-up.

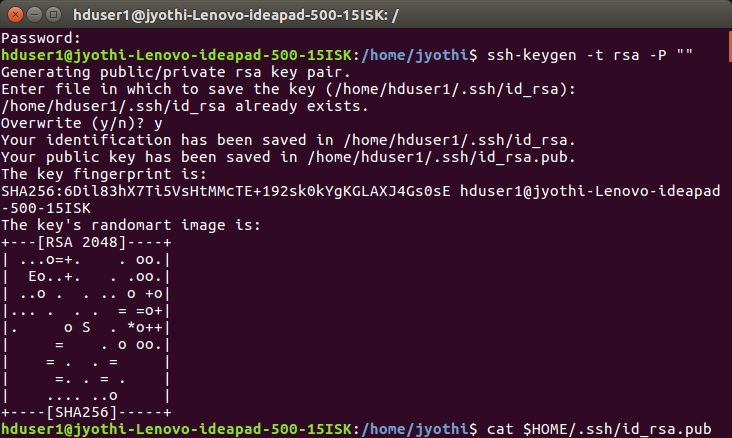


Figure 39: Configuring SSH keys

The command given is ssh-keygen –t rsa –P “”

Here this command is used to create the public key which needs to be distributed to the slave nodes for a multinode cluster.

In case of single node, the key is used for ssh access into the single node.

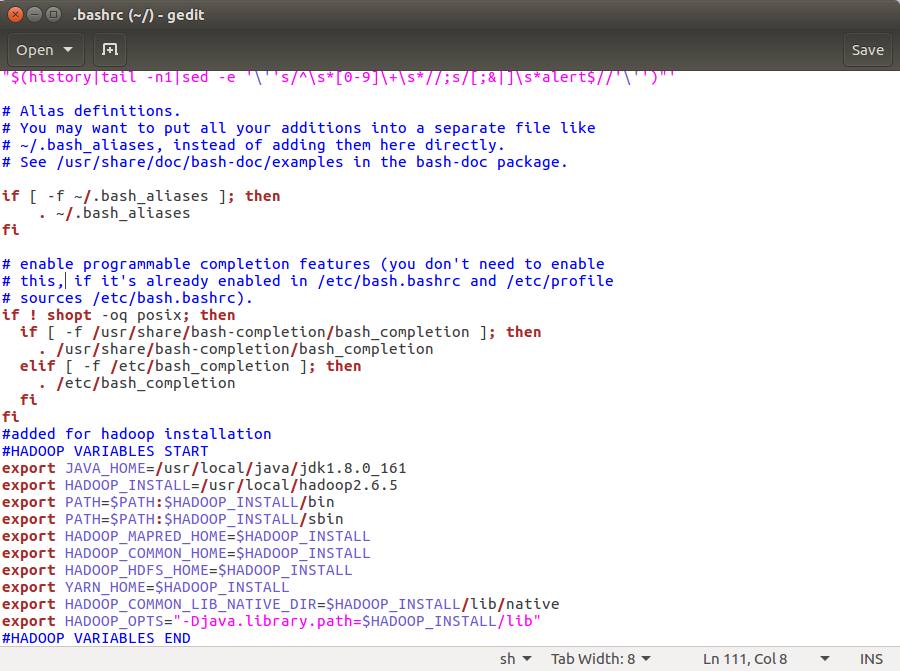


Figure 40: Modified .bashrc file for Hadoop setup

The bash file in the Ubuntu system, has the variables that the system needs to locate along with their paths.

JAVA\_HOME is one such variable that is added so as that the system or Hadoop file can know where the java JDK resides for execution.

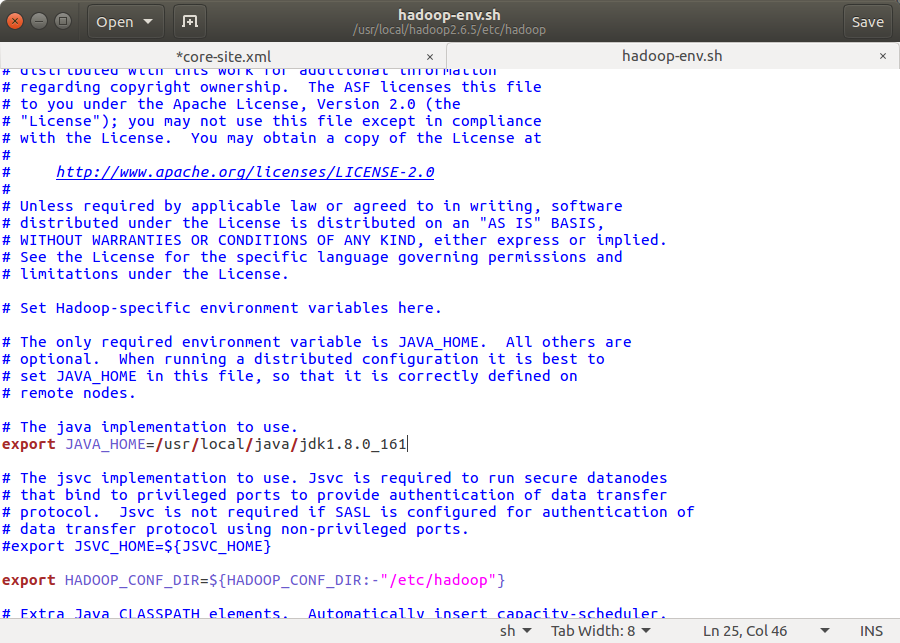


Figure 41: Modified Hadoop-env.sh file for Hadoop setup

The Hadoop-env.sh file has the path of the Hadoop directory and the path of the java directory for the applications running on the Hadoop cluster to access Hadoop and Java respectively.

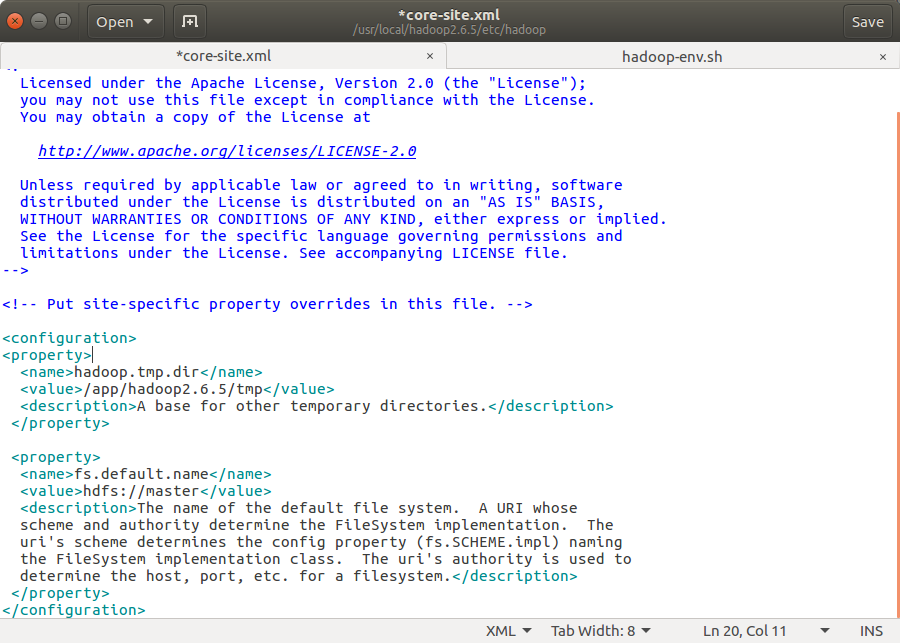


Figure 42: Modified core-site.xml for Hadoop setup

Core-site.xml file has the name of the default file system which decides the file system that is to be used in Hadoop which is generally HDFS.

The parameter or property here is used to determine the head or authority to carry out the execution of the file system.

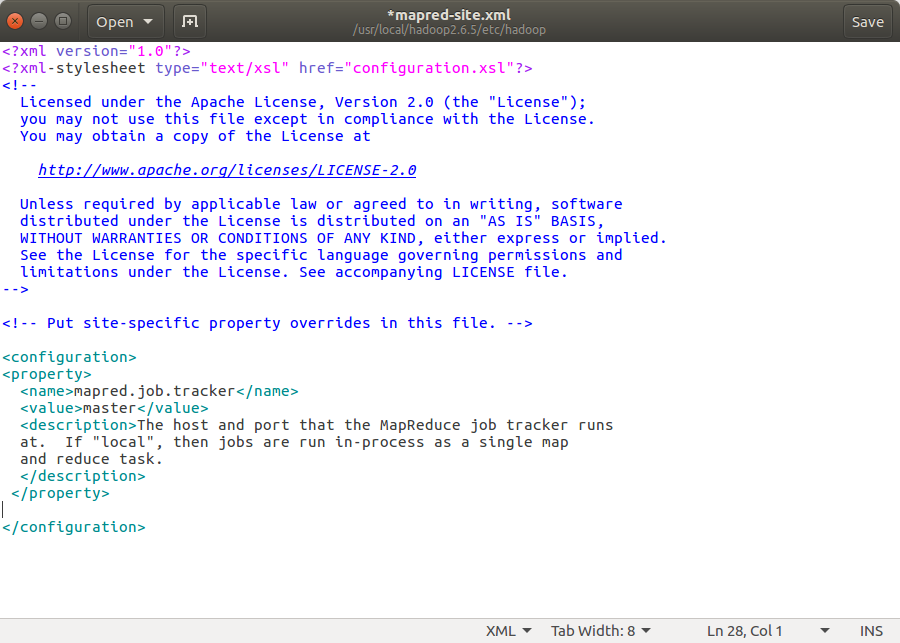


Figure 43: Modified mapred-site.xml for Hadoop setup

The mapred.job.tracker has the value of master for the multinode cluster.

It has localhost as the default value on a single node Hadoop Cluster.

This file is used to specify the host and port that the MapReduce job tracker needs to run on.

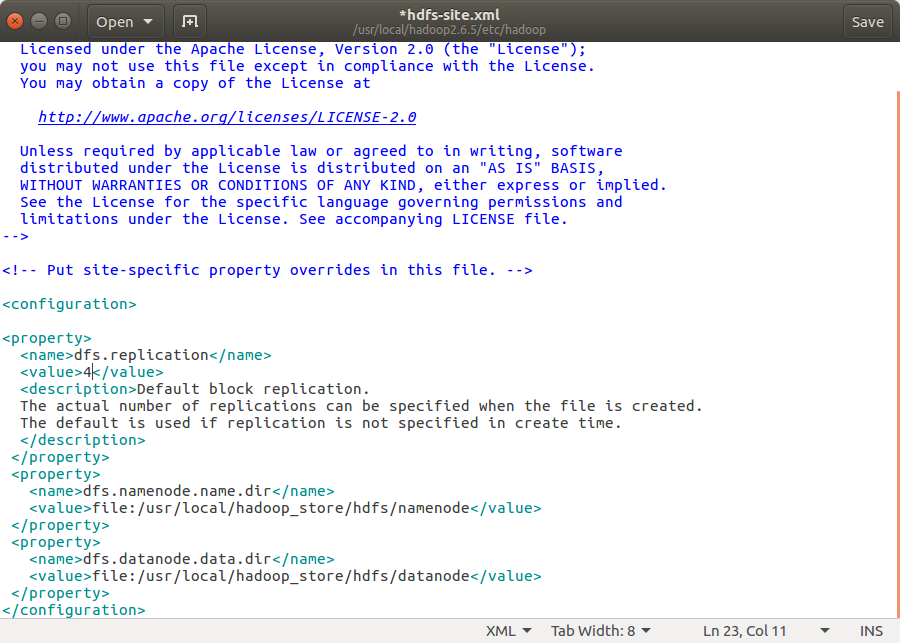


Figure 44: Modified hdfs-site.xml for Hadoop setup

Here, the dfs.replication value is set to 4 for our 4 node multinode Hadoop cluster.

For a single node standalone cluster the value would be set to 1 by default.

The following is a snapshot of the WordCount program that we ran:

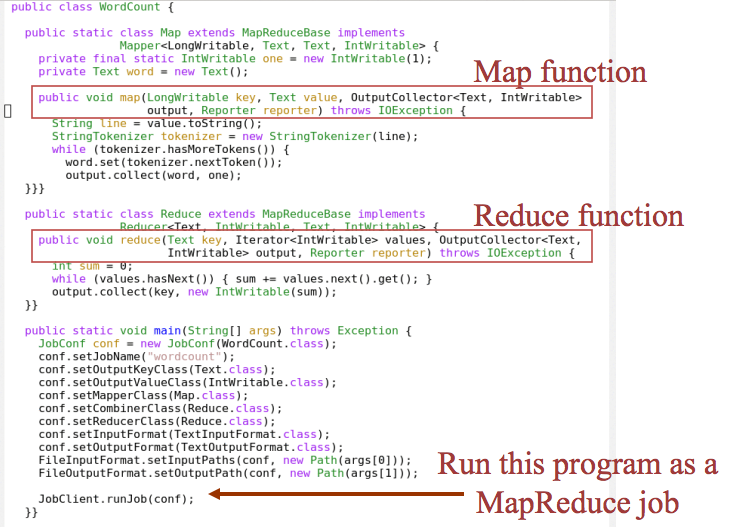


Figure 45: Hadoop WordCount

Hadoop MapReduce has a number of applications, out of which we have used WordCount program.

This program as the name suggests, is a program used mainly for text centric applications to carry out a frequency check for each word present.

The Mapper takes in the text and creates key value pairs.

For example, is our text has the following words –

“Banana Apple Banana Guava Orange Guava”

Then the Mapper for WordCount will create output like –

Banana, 1

Apple, 1

Banana, 1

Guava, 1

Orange, 1

Guava, 1

If the input split size is of 2 for example then the splits created will have the form and output as –

Split 1 –

Banana, 1

Apple, 1

Split 2 –

Banana, 1

Guava, 1

Split 3 –

Orange, 1

Guava, 1

The Shuffle Phase for the MapReduce WordCount will then sort the input splits by key to give the following intermediate output files –

File 1 –

Apple, 1

File 2 –

Banana, 1

Banana, 1

File 3–

Guava, 1

Guava, 1

File 4–

Orange, 1

The merge phase in Shuffle phase has the io.sort.factor set to 10 by default, that is by default 10 files will merge together for a reducer in Hadoop. But, for our example

which is rather small let us assume that the factor is set to 1, then we have 4 reduce threads corresponding to the 4 keys in our example.

The Reducer files will have the output –

File 1 –

Apple, 1

File 2 –

Banana, 2

File 3 –

Guava, 2

File 4 –

Orange, 1

Thus, this will be the output for the MapReduce that has been carried out using WordCount program, that is the words are counted in a smiliar fashion using our JAR file that we have used.

To run the WordCount Program,

* Download the datasets.
* Copy the file from the local file system to the HDFS system.
* Check if the file exists in HDFS.
* At a time, only one input file must be in the HDFS…!
* Run the wordcount JAR to count the words and their frequencies.
* Check the output of the wordcount and also time taken for execution.
* Remove the files, both input and outputm from the HDFS if you want to use a different input file.
* Repeat from the starting step to execute on another input file.

The screenshots below were taken at the time of running the Hadoop jobs after tuning the parameters to create different data points according to the dataset.

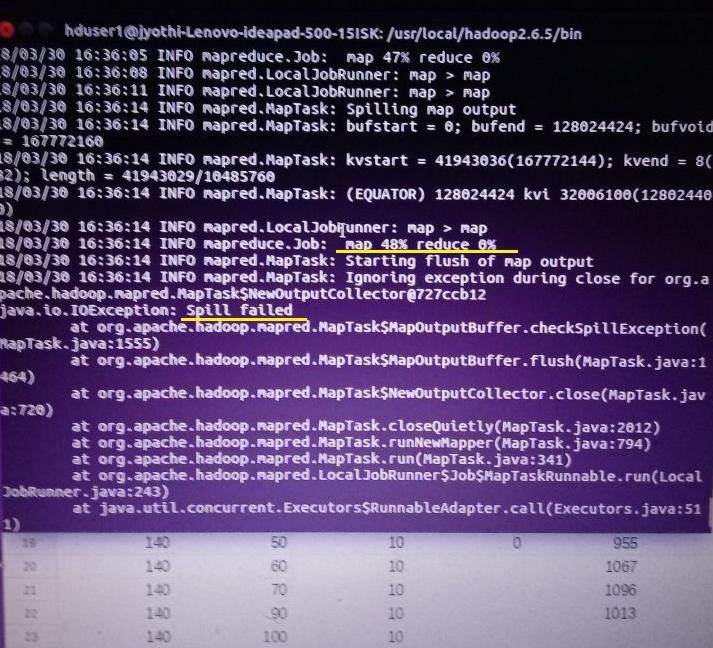


Figure 46: Failed Hadoop Job Execution

As highlighted in the figure above, this screenshot was taken when the Hadoop job failed, “Spill Failed” is the error message shown.

When the job is successful, the failed shuffles shown are zero, shuffled maps are indicated and as highlighted in the figure below, the time taken for execution can be noted.

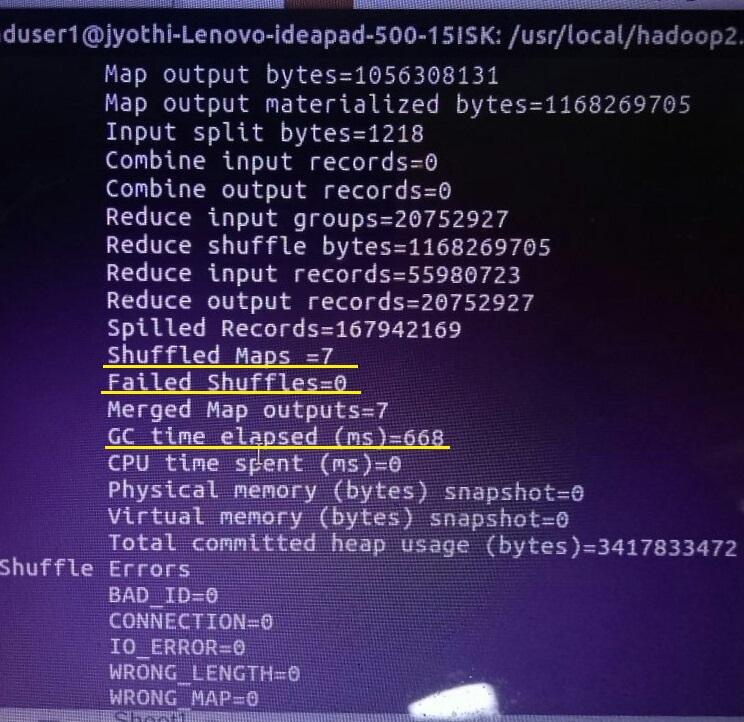


Figure 47: Successful Hadoop Job Execution

When the job is successful, the failed shuffles shown are zero, shuffled maps are indicated and as highlighted in the figure below, the time taken for execution can be noted.

The following figures below captures our GUI –

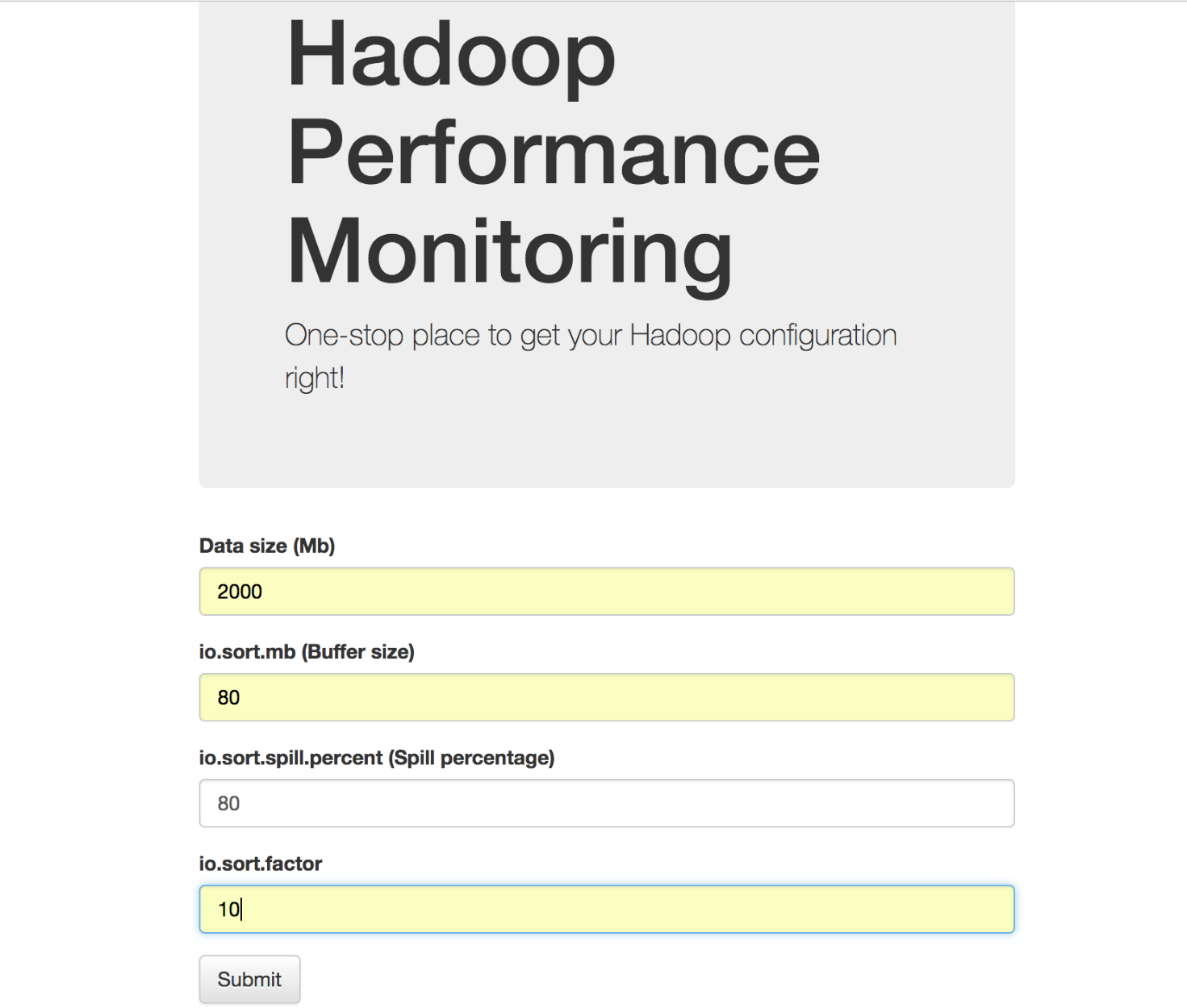


Figure 48: Landing page of UI

This is the GUI where one can enter the parameters and accordingly the values for the execution time are predicted.

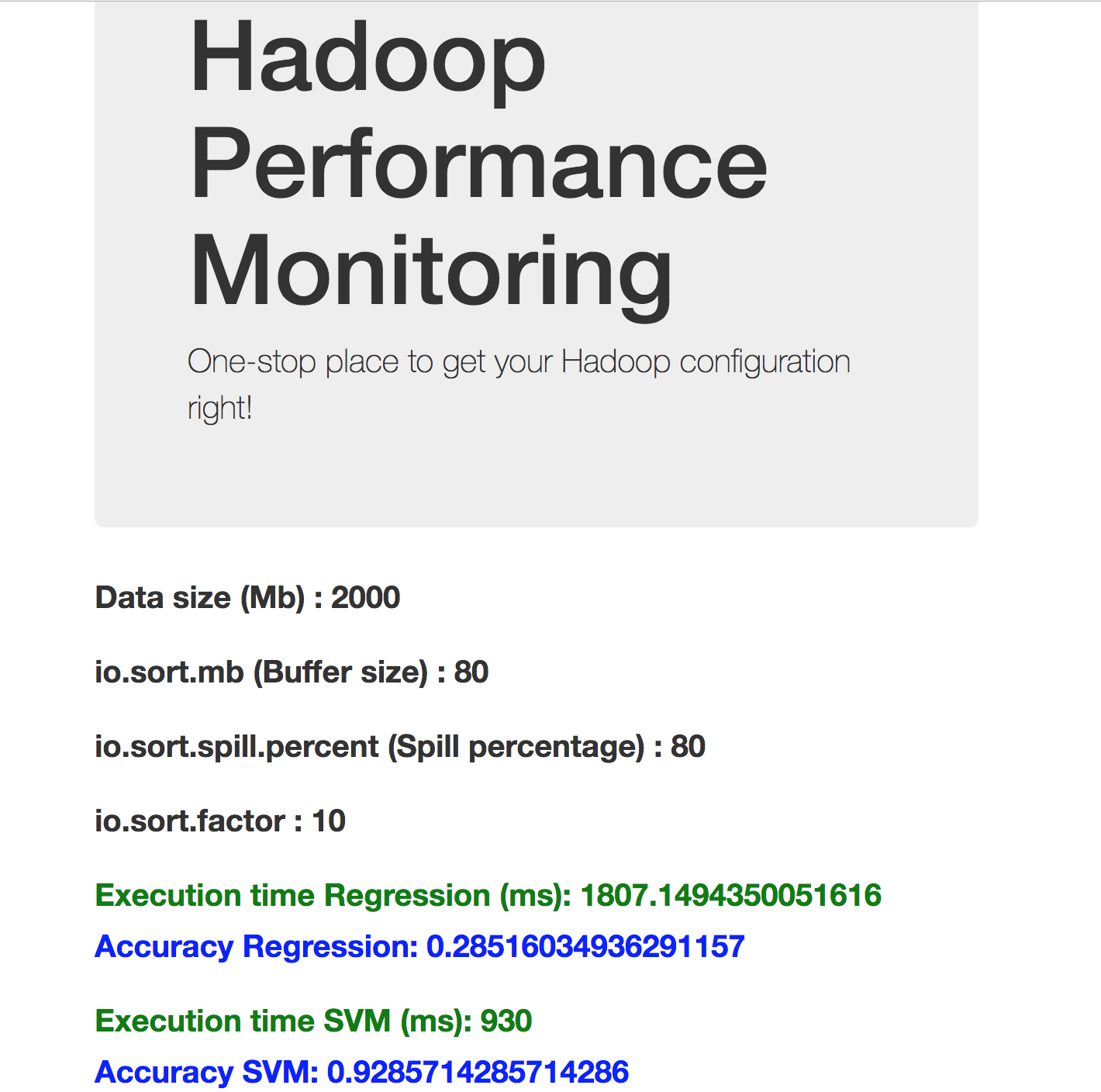


Figure 49: Prediction page of UI

The prediction due to different ML models is shown here.

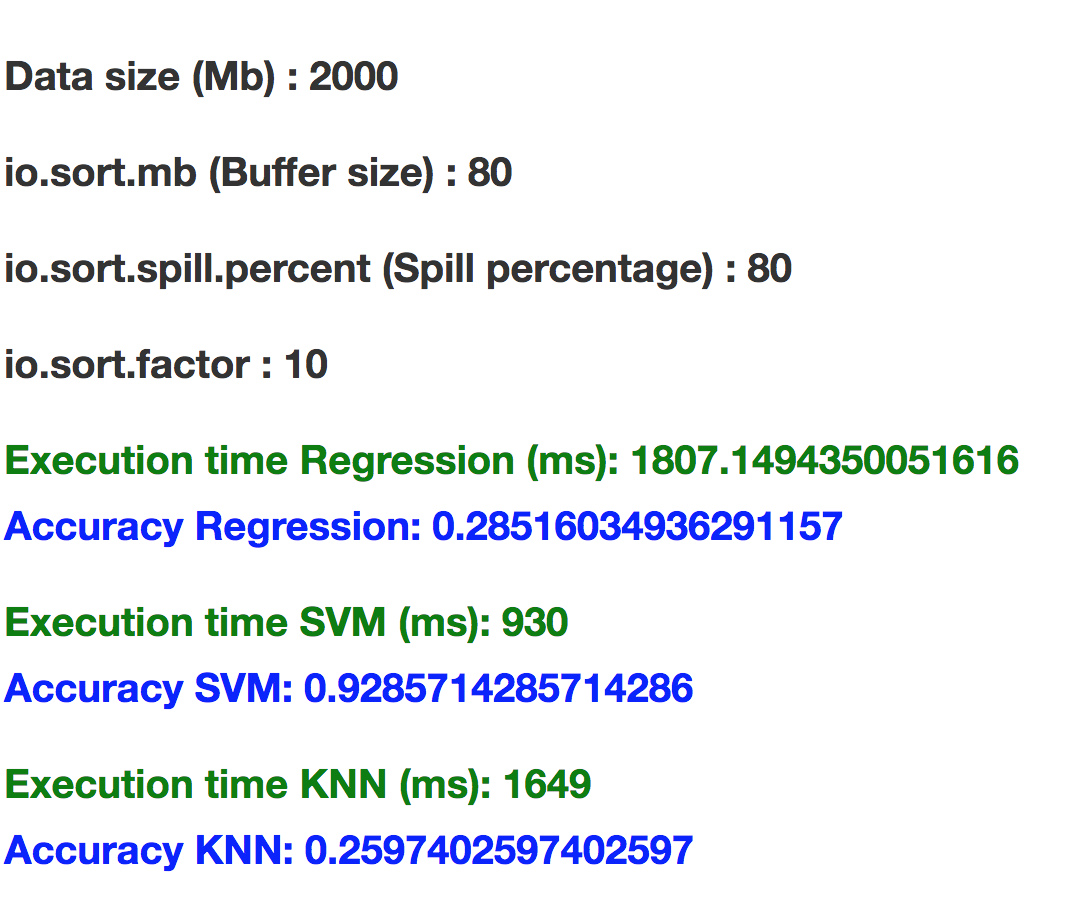


Figure 50: Accuracy of prediction models

This is the close up view of the accuracy and time predicted for each model.

The different models we used are –

* Support Vector Machines
* Regression (Multiple)
* K Nearest Neighbors Algorithm

For the above models, the accuracies observed are –

* 92.8 % for SVM
* 28.5% for Regression
* 25.9% for KNN

We observe that this accuracy also depends on the data fed, and for our final result set, the algorithm for SVM works the best.

The user is actually shown the accuracies of all the models and the final choice is left upon the user to choose the model that fits his/her data the best.

Thus, in our project we tried our best to optimize the Hadoop Shuffle Phase by applying Zipf’s Law and varying the Hadoop Configuration Parameters.