An Analytic solution for the Hadoop Configuration Combinatorial Puzzle based on General Factorial Design

R. Sathia Priya[1*], A. John Prakash[2] and V. Rhymend Uthariaraj[2]
[1] Ramanujan Computing Centre, College of Engineering, Anna University
Chennai, Tamil Nadu 600 025 India
[1] Department of Computer Science and Engineering, Loyola-ICAM College of Engineering and Technology
Chennai, Tamil Nadu 600 034 India
[e-mail: spriyarobert@gmail.com]
[2] Ramanujan Computing Centre, College of Engineering, Anna University
Chennai, Tamil Nadu 600 025 India
[e-mail: johnprakash@annauniv.edu, vrunathan@gmail.com]
[*]Corresponding author: R. Sathia Priya

Received February 3, 2022; revised March 8, 2022; accepted September 21, 2022; published November 30, 2022

Abstract

Big data analytics offers endless opportunities for operational enhancement by extracting valuable insights from complex voluminous data. Hadoop is a comprehensive technological suite which offers solutions for the large scale storage and computing needs of Big data. The performance of Hadoop is closely tied with its configuration settings which depends on the cluster capacity and the application profile. Since Hadoop has over 190 configuration parameters, tuning them to gain optimal application performance is a daunting challenge. Our approach is to extract a subset of impactful parameters from which the performance enhancing sub-optimal configuration is then narrowed down. This paper presents a statistical model to analyze the significance of the effect of Hadoop parameters on a variety of performance metrics. Our model decomposes the total observed performance variation and ascribes them to the main parameters, their interaction effects and noise factors. The method clearly segregates impactful parameters from the rest. The configuration setting determined by our methodology has reduced the Job completion time by 22%, resource utilization in terms of memory and CPU by 15% and 12% respectively, the number of killed Maps by 50% and Disk spillage by 23%. The proposed technique can be leveraged to ease the configuration tuning task of any Hadoop cluster despite the differences in the underlying infrastructure and the application running on it.

Keywords: ANOVA, Big data, Configuration tuning, General Factorial Design, Hadoop, Impactful parameters

1. Introduction

The dawn of the internet era has dramatically revolutionized the way the world operates. A host of disruptive technologies and platform-based businesses have transformed the way we conduct our common cores. These technology-aided human activities generate incredible volumes of data loaded with invaluable insights. The current global internet population of about 5.17 billion [1] generate approximately 2.5 quintillion bytes of data each day [2] and the onslaught of technologies like Internet of Things, Social networks and Cloud platforms has further ramped up the speed of data generation, looming the Big data whirlwind to gargantuan proportions. Businesses have realized that their success proposition lies in unlocking the wealth of insights buried in these massive amounts of data. Over the past decade, a majority of enterprises have turned towards Big data and AI to improve their efficiency, competence and resilience in their business landscape [3]. The worldwide spending on Big data initiatives which was $180 billion in 2019 is projected to reach a whopping $274.3 billion in 2022 [2]. Yet, this burgeoning trend is just the first of a nine inning game.

Despite the incredibly promising opportunities upfront, a host of challenges involving data capture, transfer, analysis, visualization, security, etc., confront technologists seeking robust solutions for seamless Big data application performance. In 2006, Doug Cutting and Mike Cafarella responded to the clarion call with their ingenious Big data solution called Hadoop [4]. Hadoop is a large-scale distributed storage and parallel computing platform based on Google's MapReduce. It comprises of Java-based software utilities for core Hadoop together with a host of other components like Hive, Pig, Zookeeper and Ambari. It works together with real-time stream-processing frameworks like Apache's Spark, Samza, Flume and Kafka. Thanks to the vibrant Hadoop developer community, today Hadoop has evolved as the most comprehensive open source ecosystem for Big data computing.

Hadoop is deployed on a network of physical machines in order to realize the large scale storage and computing requirements of Big data applications. However, the distributed nature of Hadoop clusters and the disparate capacities of the commodity hardware that make up the cluster infrastructure spur up a gamut of issues. Often inconspicuous small technical glitches trigger performance deterioration and hardware failures which could easily beset even the adept Hadoop professionals. According to Bobby Johnson, formerly responsible for running Facebook's Hadoop cluster, "the Hadoop community has so far failed to account for the poor performance and high complexity of Hadoop. If you have that power and you know how to use these tools, then this thing is super powerful [5]".

The stability and performance of Hadoop clusters are closely knit with three factors viz., Hadoop configuration settings, cluster capacity and application profile. While decisions on cluster infrastructure are primarily made during the initial phase of cluster deployment, and the management of Hadoop applications are carried out using sophisticated machine learning and AI based algorithms as per the enterprise need, the routine activities of monitoring and configuration tuning turn out to be crucial to ensure cluster health and application performance. In practice, Hadoop configuration tuning is all about identifying the subset of parameters critical to an application's performance and tweaking them to optimal levels. But, Hadoop has over 190 parameters whose individual and interactive effects on the system performance are too complex to capture. For this reason, Hadoop administrators start with default settings and iteratively modify them by trial and error, each time picking a few parameters they presume would improve the performance. However, such trial and error fixes may lead to unprecedented detrimental consequences. And these stopgap solutions lack the analytical rigor that could help explain the impact of the chosen parametric settings. Many researchers [6]

have attempted to derive the optimal configuration to maximize the performance of Hadoop. Some have dealt with tuning a parametric subset while others have attempted optimizing the entire set of parameters. While optimizing parametric subsets results in sub-optimal solutions, the comprehensive approach is cluttered with high complexity. More so, then the prospect of applying analytically rigorous computational procedures onto the intractable set of Hadoop configurations can badly flounder because of their computationally infeasibility. Hence, we propose a technique called General Factorial Design based on Design of Experiments to extract only the parameters that exert significant impact on the performance for the purpose of configuration tuning. In doing so we alleviate the dimensionality problem in the configuration tuning exercise. We have demonstrated our technique with a subset of parameters usually picked by the authors experimenting with performance tuning. Our method successfully identifies the impact of each parameter and also their interaction effects on the performance. It provides the necessary analytical base for Hadoop administrators to validate their parameter tuning decisions. The following contributions are made through this paper:

1) The proposed technique efficiently quantifies the performance variations caused by different configuration settings. The total observed variance in any performance metric is decomposed and attributed to the respective parameters in order to determine which of the parameters really do matter in the configuration tuning exercise.

2) Apart from the direct impact of the main parameters, the technique is also able to estimate their interaction effects and the effect of noise factors on the performance. It thus eliminates the probability of interpretative flaws that stem out of naïve comparisons of metric graphs which may point to a wrong parameter as the one to be tweaked.

3) Our investigation encompasses a variety of crucial performance metrics like the number of task failures, disk spillage, resource consumption, etc., beyond the usual performance metric viz., job completion time.

4) The proposed technique for the analysis of impactful parameters is objective, generic and scalable with respect to any subset of parameters and the range of values they can assume.

The rest of the paper is organized in the following manner. Section II contains a brief overview of Hadoop with HDFS and MapReduce architectures and their functionalities to provide the context for our discussion. Section III describes some earlier researches in the area of configuration tuning. Section IV presents the problem of identifying impactful parameters. In Section V, we provide the statistical model for the impact analysis in Configuration tuning. The experimental setup detailing the cluster configurations, the experimental workload and the subset of parameters chosen for the impact analysis are described in Section VI. Results and discussions are presented in Section VII, followed by the conclusion in Section VIII.

2. Preliminaries

Hadoop was designed based on three of Google's research papers viz., Google File System [7], MapReduce [8] and BigTable [9]. In this section, we present an overview of Hadoop architecture and the functions of Hadoop daemons. We also briefly describe the MapReduce architecture and the application processing phases.

2.1 Hadoop Architecture and Daemons

The Hadoop framework comprises of Hadoop Distributed File System (HDFS) which

facilitates large-scale distributed data storage, Yet Another Resource Negotiator (YARN) for apportioning and distributing cluster resources and MapReduce for parallel processing of data-intensive applications. The master daemon of HDFS, called NameNode (NN) maintains the metadata of the data blocks stored across the cluster and the slave daemons, called DataNodes (DN) store the application data. The Secondary NameNode is a house-keeping daemon which regularly checkpoints the metadata using FSimage and Editlogs. The Standby NameNode acts as a backup for the active NameNode and assumes charge in the event of NameNode failure. The Yarn daemons viz., Resource Manager (RM), Scheduler and Application Manager (AM) are responsible for resource allocation and task scheduling. The slave daemons called the NodeManagers (NM) which reside in the DataNodes are responsible for task execution and reporting progress. The Application Manager which resides in one of the DataNodes manages the resource needs of a single application and monitors its life-cycle.

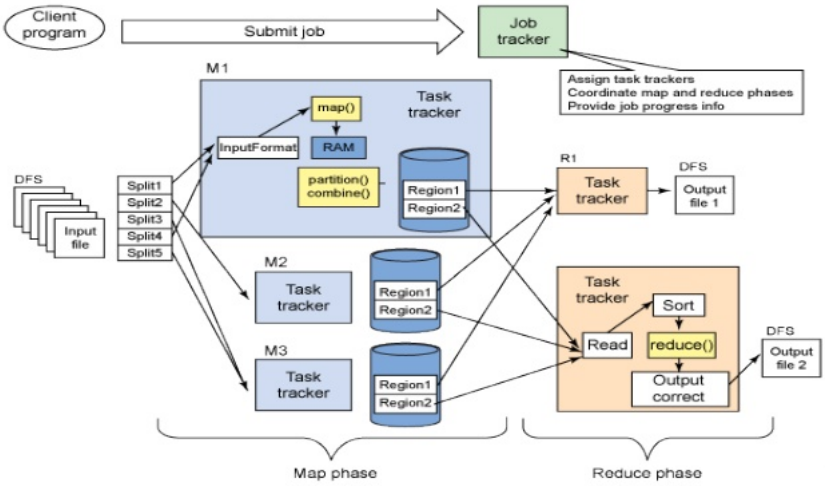

Fig. 1. Hadoop Overview

2.2 Data Storage and Application Processing

To facilitate large-scale data storage across multiple nodes in a cluster, HDFS partitions the data into small chunks of size 128MB (default in Hadoop 2.x) called blocks. The client retrieves the list of DNs from the NN and writes each block to some DN based on topological proximity. The blocks are replicated (default replication factor is 3) and stored across the racks to insure application performance against accidental data loss. However, the downside of this is the increased data redundancy and substantial wastage of cluster resources.

Hadoop applications are implemented using MapReduce programming paradigm to enable distributed and parallel computing. The applications are implemented as mapper and reducer tasks along with the driver code which sets up the environment to run the MapReduce job. When the user submits the job, the mapper is replicated sufficiently to process the data blocks stored in the HDFS. The mapper reads the input splits and generates $\langle key, value \rangle$ pairs which are further processed by the reducer tasks to aggregate and write the final output back to HDFS. The resources needed for executing the map and reduce tasks are provisioned by the Resource Manager using containers. The tasks are scheduled by the Scheduler component of YARN. The job execution is accomplished in the following phases:

Map Phase: The mapper code first reads and processes the data block stored in the HDFS and produces $\langle key, value \rangle$ pairs.

Shuffle and Sort Phase: The mapper outputs are grouped by the keys, sorted and transferred to an appropriate number of reducers in the shuffle phase.
Reduce Phase: The reducer aggregates and combines the $\langle key, value \rangle$ outputs of the Shuffle phase and generates the final output of the MapReduce application.

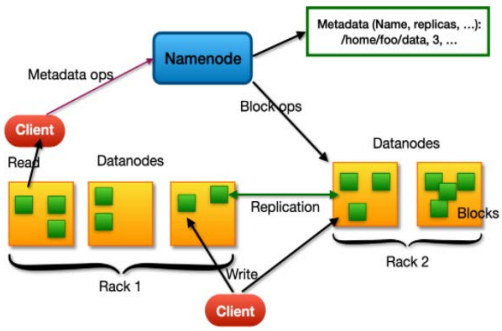

Fig. 2. HDFS Architecture

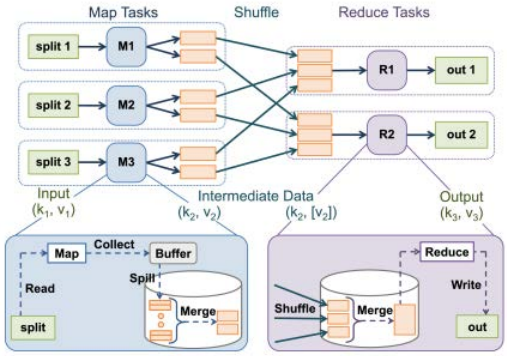

Fig. 3. MapReduce Job Execution Phases

3. Related Works

The current researches in Hadoop configuration tuning mainly revolve around machine learning, simulation, cost-based models, experiment-driven approaches and adaptive models. We present some excerpts of our literature survey in this section.

Khan et al. [10] worked on optimization of Hadoop configuration parameters using Gene Expression Programming (GEP). Their Enhanced Parallel Detrended Fluctuation Analysis (EPDFA) algorithm uses an objective function to identify the optimal values for a set of ten Hadoop parameters and determines the mathematical correlation among them based on the job execution history. Liao et al. [11] implemented a search-based auto-tuner called Gunther to evaluate variation in MapReduce application performance for different configuration settings. It uses Genetic algorithm to identify parameter settings that achieve near-optimal job execution time. The evaluation of the algorithm is performed on two clusters with a subset of six parameters considered most relevant to application performance. The Feature Selection method of tuning Hadoop configuration, proposed by Liu et al. [12] applies anisotropic Gaussian Kernel in the objective function based on the Kernel clustering algorithm in order to improve the accuracy of evaluating the importance of each MapReduce parameter. Gradient descent is used to minimize the kernel width iteratively so that the clusters of the selected features match closely with the original features. They follow a deductive approach of selecting 10 parameters and further reduce the cardinality of the chosen parametric subset step by step from 8 to 3 by analyzing their impact on job execution time.

Bao et al. [13] developed the AutoTune algorithm which uses Latin Hypercube Sampling (LHS) to generate effective samples in high-dimensional configuration space and multiple bound-and-search method to derive the best configuration. AutoTune is tested on big data benchmarks in public cloud and achieves better results than the default configurations. Hua et al. [14] proposed a non-intrusive performance profiler called H-Tune, which predicts the execution time of MapReduce applications. A meta-heuristic configuration optimizer automatically searches and deploys the optimal configurations. The experiments were

performed with twelve parameters. According to the authors the optimal configuration is application and data specific and so it is more appropriate to consider all the relevant configurations for optimization. Chen et al. [15] presented a resource-based classification of a subset of eight Hadoop parameters depending on the application profile i.e., IO-bound, CPU-bound or hybrid. Zhu et al. [16] proposed BestConfig, which uses the divide-and-diverge sampling method and the recursive-bound-and-search method for parameter tuning of general systems with resource constraints. Peyravi et al. [17] reported the runtime efficiency of a MapReduce job by considering three categories of parameters that have higher impact on the runtime. They have modeled the runtime efficiency during each phase of the Hadoop execution pipeline using a weighing system based of job history. The benchmarks RMSE and MAPE have been used to evaluate the model.

Keke et al. [18] developed the CRESP cost model involving time cost, amount of input data and free system resources to find the optimal number of map and reduce slots. They have used a weighted linear combination of a set of non-linear functions. They performed variance analysis on different components of the MapReduce workflow to identify the possible sources of modeling error. Ahmed et al. [19] conducted a comparative study of Hadoop and Spark performances using Hibenchmark workloads using different combinations of parametric settings pertaining to resource utilization, input splits and shuffle groups. They used a subset of nine Hadoop parameters and eight Spark parameters for the experiment. The analysis is conducted by comparing Hadoop and Spark based on three performance metrics viz., execution time, throughput and speedup.

It is observed that there is little consensus among the authors about the choice of parameters for any given application or resource category. This is due to the huge cardinality and complex interdependence of Hadoop parameters and the effect of extraneous factors during the application execution. Thus, given the application and cluster setup, the problem of identifying the set of impactful parameters still remains a crucial area of investigation.

4. Impactful Parameter Selection based on General Factorial Design

The Hadoop configuration set consists of more than 190 parameters related to I/O management, slot resource allocation, memory management, concurrency, map and reduce configurations, etc. [20]. The objective of Hadoop configuration tuning is to maintain cluster health and enhance production efficiency. The default configurations may not produce the desired efficiency due to variations in the cluster capacity and application runtime dynamics. Hadoop professionals start with the default configuration and work iteratively to enhance application performance. From the pilot runs of an application, the job execution is monitored closely to check whether the performance meets the expected level of service. If not, the probable causes are tracked and necessary adjustments are made to the relevant parameters to ease the performance bottlenecks. For example, the Terasort application is both CPU and memory intensive at the shuffle and sort phase. It implies that tuning the CPU and memory-specific parameters can result in performance enhancement. The rest of the parameters which are left untuned assume default values automatically. The choice of parameters to be tuned is left to the discretion of the Hadoop administrator who relies on his intuit and experience to resolve issues. In case, some critical parameter is tweaked indiscreetly, it may result in application and hardware failures. Hence, Hadoop administrators pay due diligence in considering the consequences of tuning the parameters they pick to resolve performance issues. We highlight some of the implications of tuning the test parameters used in our experiment here:

i) *dfs.replication* specifies the desired number of replications of each data block. Accordingly, HDFS replicates data blocks and stores them in different machines on separate racks as per the Data placement policy. The default replication factor is 3. It means thrice the amount of input data is stored in the cluster. Higher replication values increase data redundancy and consume more cluster resources. Nevertheless, high-availability of data in a failure prone Hadoop environment is a desirable property. It is important to determine an appropriate value of the parameter to strike the right balance between high-availability of data and job performance.

ii) *mapreduce.reduce.shuffle.parallelcopies* specifies the number of threads used to fetch the mapper outputs to reducers. Hadoop is designed for distributed storage and parallel computing. However, an indiscriminate increase in the number of parallel copies in the shuffle phase may not always speed up the job process. On the contrary, if the value the parameter is set greater than the number of available cores, many threads will be forced into waiting state and the network would be overwhelmed.

iii) *mapreduce.task.io.sort.mb* specifies the amount of buffer memory used while sorting files. When the mapper starts producing the intermediate output it does not directly write the data on the local disk. Rather it writes the data in memory and sorts them. Each map task has a circular memory buffer to which it writes the output. By default, this circular buffer is of size 100 MB which can be modified using this parameter. The maximum value is 2047 MB. When the contents of the buffer reach the default threshold of 80%, a background thread would start spilling the contents to the disk. Map outputs will continue to be written to the buffer while the spill occurs, but if the buffer fills up during this time, the task execution on the mapper outputs will be stalled until the execution on the spilled data is completed. This would again prolong the job processing time.

iv) *mapreduce.job.reduces* sets up the number of reducers which defines the level of parallelism in the reduce phase. By default, the value is one, but no parallelism can be achieved with one reducer. If the number of reducers is increased, ideally, it should result in better load balancing and faster job execution. However, setting an appropriate number of reducers is a lot trickier as it depends on the mapper outputs.

Depending on the varying hardware configurations and the application demands, the parameters have to be suitably adjusted to increase productivity. Bad configuration settings could make things go haywire in the complex Hadoop environment. Hence, monitoring and configuration tuning are indispensable to keep the cluster up and running. However, the large cardinality of the Hadoop parametric set and the unknown range of workable values make the problem of identifying the optimal configuration, an insoluble combinatorial puzzle. According to Levitin [21], most computer scientists believe that no solution exists both in theory and practice for such combinatorial problems. Even working with a subset of parameters can be complicated as the number of combinations grows exponentially with every addition of parameter. The complexity is further exaggerated by the interaction and noise effects. In this context, we propose an objective technique of narrowing down the search space for the sub-optimal solution. Our approach is to extract the parameters which significantly impact the performance using a quantitative technique based on General Factorial Design.

5. The Proposed Analytical Model

General Factorial Design is a statistical experimental design technique which aims to investigate a system's response to the changes in the factors influencing it. Its purpose is to inspect the main effects and interaction effects of those factors set at different levels. In our experiment, the design examines the effect of configuration settings of four parameters viz., A: *mapreduce.job.reduces* (2 levels: 1, 10), B: *dfs.replication* (3 levels: 1, 2, 3), C: *mapreduce.reduce.shuffle.parallelcopies* (2 levels: 5, 10) and D: *mapreduce.task.io.sort.mb* (3 levels: 64, 128, 256) on performance metrics like Job Completion Time, CPU utilization, Memory utilization, number of killed map tasks, etc., each of which is a response variable of the Hadoop system. The response variable denoted by Y_{ijklm} is the linear sum of the main effects, interaction effects (2-way, 3-way and 4-way) and noise effects, given by

$$Y_{ijklm} = \mu + A_i + B_j + C_k + D_l + (AB)_{ij} + (AC)_{ik} + (AD)_{il} + (BC)_{jk} + (BD)_{jl} + (CD)_{kl} + (ABC)_{ijk} + (ABD)_{ijl} + (ACD)_{ikl} + (BCD)_{jkl} + (ABCD)_{ijkl} + e_{(ijkl)m} \quad (1)$$

$$where \begin{cases} i = 1,2,\cdots,a \\ j = 1,2,\cdots,b \\ k = 1,2,\cdots,c \\ l = 1,2,\cdots,d \\ m = 1,2,\cdots,n \end{cases}$$

Y_{ijklm} : response variable, viz., Job completion time, memory utilization, etc.
a,b,c,d : number of levels of the 4 factors A, B, C & D respectively
n : number of replications of each test case
$e_{(ijkl)m}$: error due to extraneous factors
μ : overall effect which is estimated by the grand mean
A_i, B_j, C_k, D_l : response values of the main effects
$(AC)_{ik}$, ... : response values of 2-way interactions
$(ABC)_{ijk}$, ... : response values of 3-way interactions
$(ABCD)_{ijkl}$: response values of 4-way interactions

We set the Null Hypothesis H_0 and the Alternate Hypothesis H_1 to test whether the quantum of variance produced by each of the main factors A, B, C and D and their interaction effects is significant or not.

For the main factor A: $H_0: A_1 = A_2 = \cdots = A_a = 0$ (1)
$H_1: atleastone A_i \neq 0$

Similar hypotheses can be defined for the variance produced by the other main effects B, C and D.

For the interaction factor AB: $H_0: (AB)_{ij} = 0, \forall i,j$ (2)
$H_1: atleastone (AB)_{ij} \neq 0$

Similar hypotheses can be defined for the variance produced by the other 2-factor interaction effects AC, AD, BC, BD and CD.

For the interaction factor ABC:
$H_0: (ABC)_{ijk} = 0, \forall i,j,k$ (3)
$H_1: atleastone (ABC)_{ijk} \neq 0$

Similar hypotheses can be defined for the variance effected by the other 3-factor interaction effects ABD, ACD and BCD.

For the 4-factor interaction effect of ABCD:

$$H_0: (ABCD)_{ijkl} = 0, \forall i,j,k,l \quad (4)$$
$$H_1: atleastone(ABCD)_{ijkl} \neq 0.$$

The total sum of squares of the variance is given by

$$SS_{Total} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c}\sum_{l=1}^{d}\sum_{m=1}^{n} Y_{ijklm}^2 - CF \quad (5)$$

where $CorrectionFator(CF) = \frac{T_{....}^2}{abcdn}$

The sum of squares of the main effects is given by the sum of their respective levels:

$$SS_A = \frac{\sum_{i=1}^{a} T_{i....}^2}{bcdn} - CF, SS_B = \frac{\sum_{j=1}^{b} T_{.j...}^2}{acdn} - CF$$
$$SS_C = \frac{\sum_{k=1}^{c} T_{..k..}^2}{abdn} - CF, SS_D = \frac{\sum_{l=1}^{d} T_{...l.}^2}{abcn} - CF \quad (6)$$

The two factor interaction sums of squares are given by

$$SS_{AB} = \frac{\sum_{i=1}^{a}\sum_{j=1}^{b} T_{ij...}^2}{cdn} - CF - SS_A - SS_B$$
$$SS_{AC} = \frac{\sum_{i=1}^{a}\sum_{k=1}^{c} T_{i.k.}^2}{bdn} - CF - SS_A - SS_C$$
$$SS_{AD} = \frac{\sum_{i=1}^{a}\sum_{l=1}^{d} T_{i..l}^2}{bcn} - CF - SS_A - SS_D$$
$$SS_{BC} = \frac{\sum_{j=1}^{b}\sum_{k=1}^{c} T_{.jk.}^2}{adn} - CF - SS_B - SS_C$$
$$SS_{BD} = \frac{\sum_{j=1}^{b}\sum_{l=1}^{d} T_{.j.l}^2}{acn} - CF - SS_B - SS_D \quad (7)$$
$$SS_{CD} = \frac{\sum_{k=1}^{c}\sum_{l=1}^{d} T_{..kl}^2}{abn} - CF - SS_C - SS_D$$

The three-factor interaction sums of squares are given by

$$SS_{ABC} = \frac{\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c} T_{ijk..}^2}{dn} - CF - SS_A - SS_B - SS_C - SS_{AB} - SS_{BC} - SS_{AC}$$
$$SS_{ABD} = \frac{\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{l=1}^{d} T_{ij.l.}^2}{cn} - CF - SS_A - SS_B - SS_D - SS_{AB} - SS_{BD} - SS_{AD}$$
$$SS_{ACD} = \frac{\sum_{i=1}^{a}\sum_{k=1}^{c}\sum_{l=1}^{d} T_{i.kl.}^2}{bn} - CF - SS_A - SS_C - SS_D - SS_{AC} - SS_{AD} - SS_{CD}$$
$$SS_{BCD} = \frac{\sum_{j=1}^{b}\sum_{k=1}^{c}\sum_{l=1}^{d} T_{.jkl.}^2}{an} - CF - SS_B - SS_C - SS_D - SS_{BC} - SS_{BD} - SS_{CD} \quad (8)$$

The four-factor interaction sum of squares is given by

$$SS_{ABCD} = \frac{\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c}\sum_{l=1}^{d} T_{ijkl.}^2}{n} - CF - SS_A - SS_B - SS_C - SS_D - SS_{AB} - SS_{AC} - SS_{AD} - SS_{BC} - SS_{BD} - SS_{CD} - SS_{ABC} - SS_{ABD} - SS_{ACD} - SS_{BCD} \quad (9)$$

The sum of squares of these main and interaction factors are denoted by $SSTr$ in the ANOVA table [2]. The error sum of squares is obtained by subtracting the sum of squares of all the main and interaction effects from the total sum of squares.

$$SSE = SS_{Total} - SS_{main\&interaction\_effects} \quad (10)$$

The degrees of freedom (*df*) is evaluated for the main factors using

$$df_{mainfactor} = no.oflevels - 1 \quad (11)$$
$$df_{2-factorinteractions} = (no.oflevelsoffactor1 - 1) \times (no.oflevelsoffactor2 - 1)$$
$$df_{3-factorinteractions} = (no.oflevelsoffactor1 - 1) \times (no.oflevelsoffactor2 - 1) \times (no.oflevelsoffactor3 - 1)$$
$$df_{4-factorinteractions} = (no.oflevelsoffactor1 - 1) \times (no.oflevelsoffactor2 - 1) \times (no.oflevelsoffactor3 - 1) \times (no.oflevelsoffactor4 - 1)$$

The computations for mean square error and F-statistic values are presented in the ANOVA table. These values are compared with the tabulated values viz., F-critical at 5% level of significance for the respective degrees of freedom.

Table 1. Analysis of Variance (ANOVA)

Source of variation i.e., Factors		Sum of Squares (SS)	Degrees of freedom (df)	Mean Square (MS)	F-statistic
Direct Impact of Main Factors					
A		SS_A	a-1	$MS_A = \frac{SS_A}{df_A}$	$F = \frac{MS_A}{MSE}$
⋮		⋮	⋮	⋮	⋮
D		SS_D	d-1	$MS_D = \frac{SS_D}{df_D}$	$F = \frac{MS_D}{MSE}$
Interaction Effects					
2-way	AB	SS_{AB}	$(a-1) \times (b-1)$	$MS_{AB} = \frac{SS_{AB}}{df_{AB}}$	$F = \frac{MS_{AB}}{MSE}$
	⋮	⋮	⋮	⋮	⋮
	CD	SS_{CD}	$(c-1) \times (d-1)$	$MS_{CD} = \frac{SS_{CD}}{df_{CD}}$	$F = \frac{MS_{CD}}{MSE}$
3-way	ABC	SS_{ABC}	$(a-1) \times (b-1) \times (c-1)$	$MS_{ABC} = \frac{SS_{ABC}}{df_{ABC}}$	$F = \frac{MS_{ABC}}{MSE}$
	⋮	⋮	⋮	⋮	⋮
	BCD	SS_{BCD}	$(b-1) \times (c-1) \times (d-1)$	$MS_{BCD} = \frac{SS_{BCD}}{df_{BCD}}$	$F = \frac{MS_{BCD}}{MSE}$
4-way	$ABCD$	SS_{ABCD}	$k = (a-1) \times (b-1) \times (c-1) \times (d-1)$	$MS_{ABCD} = \frac{SS_{ABCD}}{df_{ABCD}}$	$F = \frac{MS_{ABCD}}{MSE}$
Error		SSE	N-k	$MSE = \frac{SSE}{N-k}$	
Total		SST	N-1		

If F-statistic < F-critical, the corresponding H_0 is accepted, otherwise H_1 is accepted. The acceptance of H_0 implies that the corresponding parameter does not significantly affect the performance metric, while the acceptance of H_1 implies the contrary i.e., changes in the corresponding parameter produces statistically significantly variance in the performance metric.

Procedure

Step 1 Declare metric, number_of_replications, parameters, parameter_levels

Step 2 For each metric do

Step 2a Set H_0 and H_1

Step 2b Compute the following:

CF : Correction factor

SS_T : Total sum of squares for individual parameters

$SS_{p_i}, i = 1, \cdots, s$: Sum of squares of each parameter, where s is the number of parameters

$SS_{p_i p_j}, i, j = 1, \cdots, s \ \& \ i \neq j$: Sum of squares for 2-way interactions

$SS_{p_i p_j p_k}, i, j, k = 1, \cdots, s \ \& \ i \neq j \neq k, \ldots$: Sum of squares for 3-way interactions

$SS_{p_1 \cdots p_s}$: s-way interactions

SS_E : Error sum of squares

Step 2c For the ANOVA table compute the following with respect to each of the main parameters, the interaction effects and error term

df : Degrees of freedom

MSS : Mean sum of squares

$F - statistic$

Step 2d If $F - statistic < F - critical$ at 5% level of significance, then accept H_0 else accept H_1.

6. Experimental Setup

In this section we present the specifications of the Hadoop cluster, the workload and the parameter settings used for our experiment.

6.1 Cluster Configuration

To conduct the experiment we built a 4-node Hadoop cluster with Apache Hadoop 3.2.1 release, as it is the latest stable version at the time of experimentation. The cluster comprises of 3 Datanodes and a master node wherein both Namenode and Secondary Namenode services are deployed. The cluster configurations are presented in the table below:

Table 2. Experimental Cluster Configuration

Cluster Specifications		**Name node (NN)**	**Data node 1 (DN1)**	**Data node 2 (DN2)**	**Data node 3 (DN3)**
Software	Operating System	Ubuntu 16.04.2 (GNU/Linux 4.13.0-37-generic×86 64)			
	JDK	1.8.0			
	Hadoop	3.2.1			
Node Configurations	CPU	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz × 4	Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz × 4		
	Memory	15.5 GiB	15.5 GiB	7.7 GiB	15.5 GiB
	Disk	2.0 TB	983.4 GB	2.0 TB	2.0 TB
Workload	Terasort Benchmark				

6.2 Hadoop Parameter Combinations for Experimentation

We conducted a total of 72 experimental runs for the set of 36 combinations of Hadoop configuration settings, replicating each combination twice. We restricted the number of combinations and the experimental replication factor in order to make the experimental study viable. According to Montgomery [22], since resources are usually limited, the number of replicates the experimenter can employ may be restricted. In practice, the number of replicates is at least 2. The experiments were conducted in random order based on a simple random number generator. Replication and randomization help in attenuating the effect of noise in the experimental runs.

The performance metrics analyzed are job completion time, number of task failures (reduce/map tasks), resource utilization (CPU, Memory) and disk spillage. Note that the technique is applicable to any of the hundreds of Hadoop Performance metrics that one can capture through monitoring tools like Ganglia or Nagios. The Hadoop configuration settings used in the experiment are tabulated below:

Table 3. Sample of Hadoop Configuration Setting

Input Parameters with Levels	Configuration File Settings with Default Values
A:*mapreduce.job.reduces* Levels: {1, 10}	mapred-site.xml <*property*> <*name*>*mapreduce.job.reduces*</*name*> <*value*>*1*</*value*> </*property*>
B:*dfs.replication* Levels: {1, 2, 3}	hdfs-site.xml <*property*> <*name*>*dfs.replication*</*name*> <*value*>*3*</*value*> </*property*>
C:*mapreduce.reduce.shuffle.parallelcopies* Levels: {5, 10}	mapred-site.xml <*property*> <*name*>*mapreduce.reduce.shuffle.parallelcopies* </*name*> <*value*>*5*</*value*> </*property*>
D:*mapreduce.task.io.sort.mb* Levels: {64, 128, 256}	mapred-site.xml <*property*> <*name*>*mapreduce.task.io.sort.mb*</*name*> <*value*>*100*</*value*> </*property*>

6.3 Hadoop Workload

Terasort was used as the test application as it is suitable to test both HDFS and MapReduce parameters. It is both CPU and I/O intensive especially in the shuffle and sort phases and thus affects most of the performance metrics. Terasort benchmark application has 3 components viz., Teragen generates synthetic data for input, Terasort sorts the data and Teravalidate validates the sorted data. We generated 5GB i.e., 50 million rows of input data using Teragen and executed Terasort for our experimental study.

7. Results and Discussion

Application statistics were collected from YARN administration web UI and ANOVA computations were carried out for each performance metric. **Table 4** presents the collated F-values against their F-critical at 5% level of significance.

Table 4. F-statistics

Source of Variation	F – value								F-critical
	Degrees of Freedom	Job Completion Time	Killed Maps	Killed Reducers	Failed Maps	CPU Time Spent	Memory MB-sec	Vcores -sec	
A	1	29.53	0.09	250	1.08	366.28	64.82	65.14	4.113
B	2	0.3	1.18	0.19	0.1	1.95	0.28	0.29	3.259
C	1	0.29	0.09	0.07	0.17	0.27	0.24	0.25	4.113
D	2	7046.57	65451.91	4.97	3148.18	1632.14	26.6	26.5	3.259
2-Way Interactions	13	1.44	1.6	0.7	0.42	15.43	2.76	2.78	2.003
A*B	2	0.62	4.45	0.19	0.36	3.66	0.31	0.3	3.259
A*C	1	0.2	2.27	0.07	0	3.58	0.22	0.2	4.113
A*D	2	8.18	0.64	2.36	1.08	91.78	17.14	17.25	3.259
B*C	2	0.11	1.73	1.47	0.93	0.92	0.01	0.01	3.259
B*D	4	0.13	1.18	0.07	0.1	0.86	0.08	0.08	2.634
C*D	2	0.07	0.09	0.35	0.17	0.44	0.25	0.24	3.259
3-Way Interactions	12	0.18	4.82	0.59	0.55	2.1	0.11	0.11	2.033
A*B*C	2	0.09	1.73	1.47	0.75	3.19	0.01	0.01	3.259
A*B*D	4	0.27	9.91	0.07	0.36	1.42	0.1	0.09	2.634
A*C*D	2	0.14	2.27	0.35	0	3.33	0.27	0.28	3.259
B*C*D	4	0.17	2.55	0.8	0.93	1.61	0.1	0.1	2.634
4-Way Interactions	4	0.19	0.91	0.8	0.75	2.25	0.11	0.12	2.634
A*B*C*D	4	0.19	0.91	0.8	0.75	2.25	0.11	0.12	2.634
Error	36								
Total	71								

The highlighted F-statistics exceed F-critical values. The corresponding parameter or combination of parameters has significant impact on the particular performance metric. The graphs depicting the main and interaction effects exhibit the configuration settings that have significant impact. A sample of the graphs generated to explore the sub-optimal configuration settings is presented below.

a) **Impact on Job Completion Time**: Minimizing the application execution time is one primary concern in Big data processing. The Pareto chart for the standardized effects reveals that the parameters D and A and their interaction AD have significant impact on the job completion time. Other parameters have negligible impact. During the initial runs of the experiment, we observed that the Sort and Shuffle phase took significantly longer time than other phases for $A = 1$. We quickly infered that it may be due to the single reducer handling the entire workload. But after increasing A to 10, we observed an unexpected rise in the job completion time. For the parameter D, the job completion time slightly increased when the setting was modified from 64 to 128. But jobs began to fail uniformly when D was set to 256. The increase in buffer size did not improve the processing speed. In fact, at $D = 256$ no

application statistics were available due to job failures and we had to assign an incomparably large value for the metric at $D = 256$ to make computations feasible. **Fig. 4** indicates that the interaction effect of A and D is significant as well.

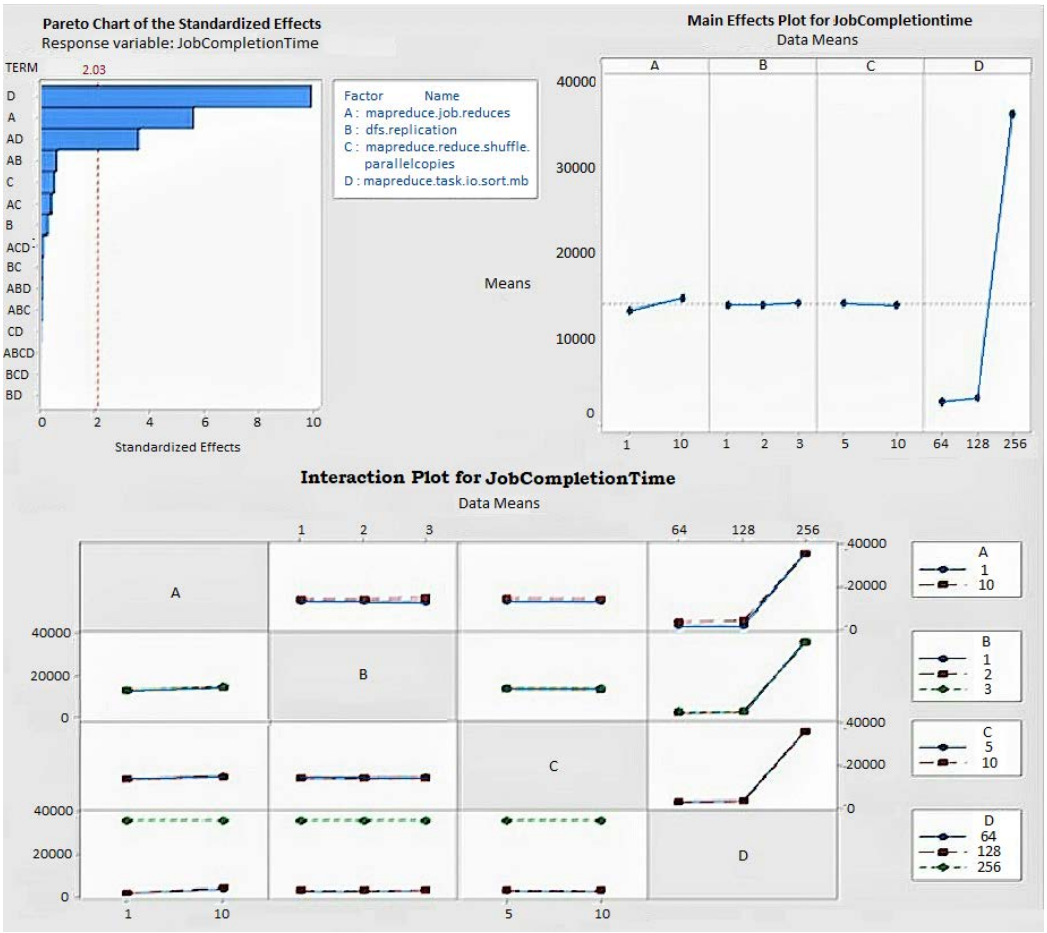

Fig. 4. Impactful Parameters for Job Completion Time

b) **Impact on Killed Maps**: MapReduce tasks are killed in circumstances where the task does not report progress or the scheduler needs the slot for some other task. In the event of task slow down, Hadoop performs speculative execution by running another instance of the straggler task on some other DataNode. The task which finishes first is accepted and the other is killed. Speculative execution speeds up job completion but the subsequent task killing wastes cluster resources. Hence, fewer killed tasks indicate better performance. In our experiment, we observed that the parameter D and the interactions ABD and AB have significant impact on the number of tasks killed. At $D = 256$ all the map tasks get killed causing the job to fail. The Pareto chart in Fig.5 shows that the combined effect of A and B also increases the number of killed maps, irrespective of the value of D.

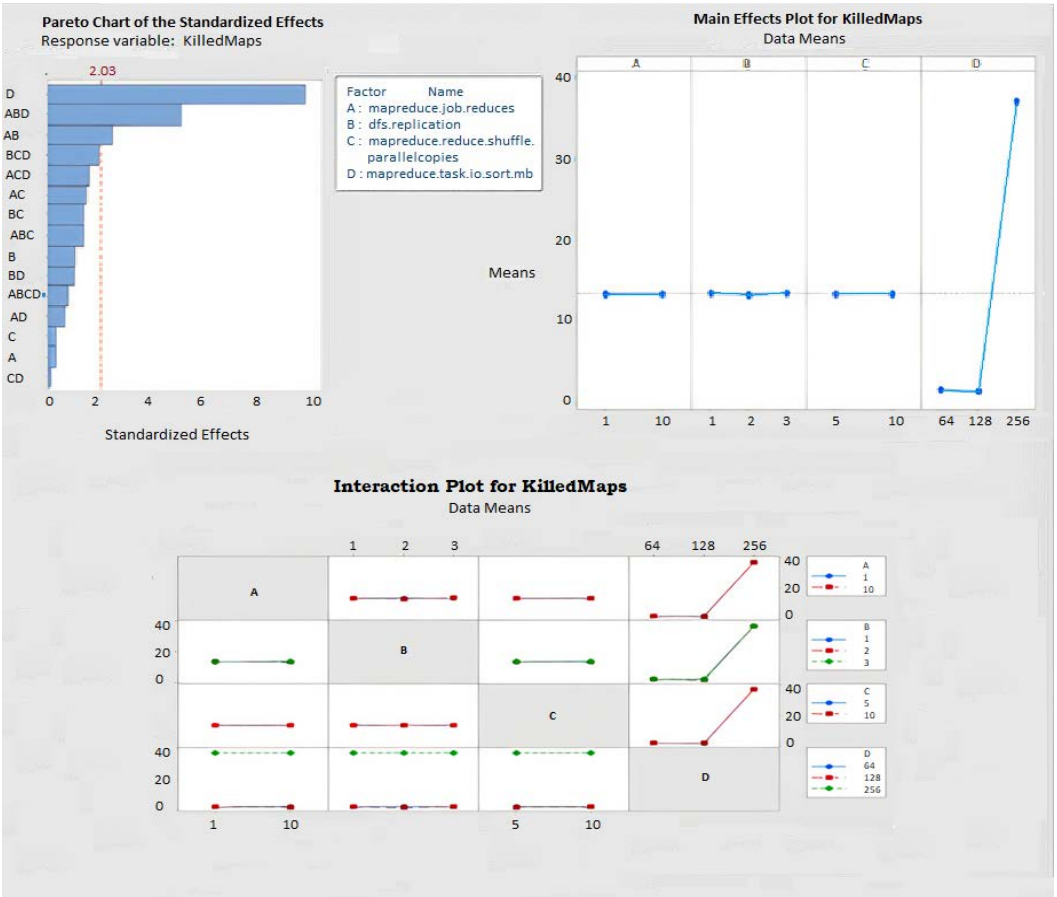

Fig. 5. Impactful parameters for Killed Maps

c) **Impact on Resource Consumption**: Hadoop performance heavily relies on the cluster resource availability and Hadoop configurations related to resources. The common resource bottlenecks in the Hadoop environment are CPU, RAM, network bandwidth and storage I/O. In Fig. 6 the impact of the parameters on CPUTimeSpent is depicted. It indicates an acute increase in CPUTimeSpent with the increase in A and D values. Similar trends are observed in vcores and memory utilization. At $D = 256$, job failures occurred as resources got severely constrained. Notice that the metric CPUTimeSpent plunges to zero at $D = 256$ as jobs stopped executing.

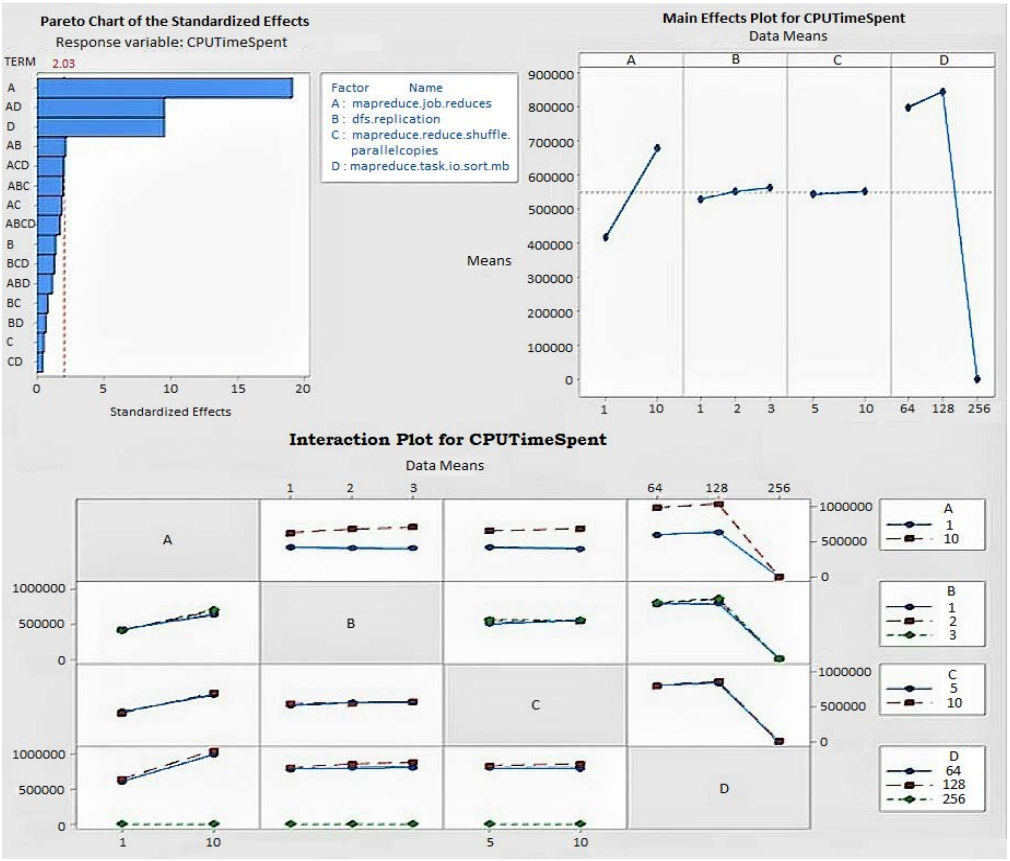

Fig. 6. Impactful parameters for CPUTimeSpent

d) **Impact on Disk Spillage**: Disk spillage can cause serious performance lags in a Hadoop cluster. It occurs due to insufficient memory for handling the mapper outputs and results in memory swaps. In order to minimize the amount of spilled records, Hadoop administrators increase the buffer size. Our experiments too showed a slight dip in the metric when D was increased from 64 to 128. However, job failures occurred with a further increase to 256. On the brighter side, disk spillage reduced when the number of reducers A was increased from 1 to 10 except when $D = 256$.

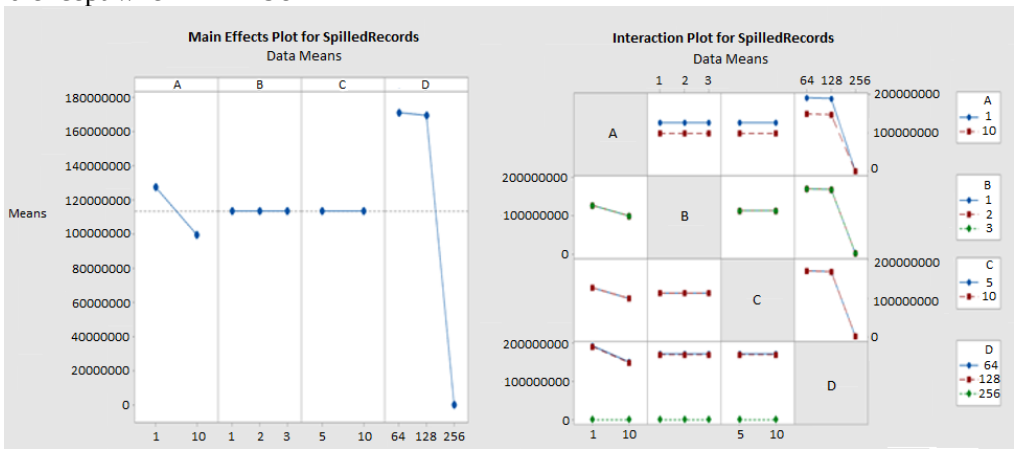

Fig. 7. Impactful parameters for SpilledRecords

Notice that the parameters B :*dfs.replication* and C:*mapreduce.reduce.shuffle.parallelcopies* do not significantly impact any of the Performance metrics listed. Overall, the experiments reveal that the factors A:*mapreduce.job.reduces* and D:*mapreduce.task.io.sort.mb* are impactful parameters both independently and combined. It indicates to the Hadoop administrator to focus his configuration tuning efforts on parameters A and D. Thus the General Factorial Design technique resolves the problem of identifying the right parameters to be tuned from among hundreds of Hadoop parameters.

Finally, in **Table 5**, we present the performance gain in terms of % decrease of the sub-optimal value of each metric against its value at the default setting $\langle 1,3,5,100 \rangle$. Most of the performance metrics show a significant performance enhancement at $\langle 1,1,10,128 \rangle$ when compared with the default setting.

Table 5. Performance gain observed in various metrics

Performance metrics	**% decrease**	**Sub-optimal Parametric settings $\langle A, B, C, D \rangle$**
Job Completion Time	22%	$\langle 1,2,10,128 \rangle$
Killed Maps	50%	$\langle 1,3,10,64 \rangle$
Killed Reducers	0%	$\langle 1,1,10,128 \rangle$
Failed Maps	0%	$\langle 1,1,10,128 \rangle$
CPU Time Spent	12%	$\langle 1,1,10,64 \rangle$
Memory MB-sec	15%	$\langle 1,1,10,128 \rangle$
vcores-sec	21%	$\langle 1,1,10,128 \rangle$
Disk Spillage	23%	$\langle 10,1,10,128 \rangle$

8. Conclusion

Configuration tuning is crucial to achieve service level performance for Big data applications. However, big data processing platforms like Hadoop have a large set of parameters which makes the goal of achieving performance optimality a farfetched proposition. Rather, aiming for improved performance with a sub-optimal configuration is a more pragmatic approach. This article presented a statistical technique to identify a subset of parameters having significant impact on the performance of a particular Big data application called Terasort. We performed an empirical analysis based on General Factorial Design on a subset of parameters by varying the configuration settings. Our experimental results revealed that out of the initial subset of parameters, *mapreduce.job.reduces* and *mapreduce.task.io.sort.mb* caused significant performance variance in terms of most of the Hadoop performance metrics, both independently and combined. The performance variance caused by other parameters was statistically insignificant. The configuration setting, thus deduced, showed considerable improvement in most of the performance metrics, especially the Job completion time which was reduced by 22%. The technique adaptable to any parametric subset as it is generic, scalable and completely objective. The General Factorial Design technique effectively resolves the murkiness underlying the configuration tuning exercise and provides Hadoop administrators the required base to substantiate their parameter tuning decisions.

References

[1] "Data never sleeps 9.0," 2021. [Online]. Available: https://www.domo.com/learn/infographic/data-never-sleeps-9

[2] Christo Petrov, "25+ Impressive Big data statistics for 2021,". [Online]. Available: https://techjury.net/blog/big-data-statistics/#gref

[3] New Vantage Partners, "Big data and AI Executive Survey," 2019. [Online]. Available: https://www.tcs.com/content/dam/tcs-bts/pdf/insights/Big-Data-Executive-Survey-2019-Findings-Updated-010219-1.pdf

[4] Marco Bonaci, "A History of Hadoop," 2015. [Online]. Available: https://medium.com/@markobonaci/the-history-of-hadoop-68984a11704

[5] Alex Woodie: "Hadoop has failed us, Tech experts say," 2017. [Online]. Available: https://www.datanami.com/2017/03/13/hadoop-failed-us-tech-experts-say/

[6] Herodotos Herodotou, Yuzing Chen, Jiaheng Lu, "A survey on Automatic Parameter Tuning for Big Data Processing Systems," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1-37, March 2021. Article (CrossRef Link)

[7] Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung, "The Google File System," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29-43, Dec 2003. Article (CrossRef Link)

[8] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: A Flexible Data Processing Tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72-77, Jan 2010. Article (CrossRef Link)

[9] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1-26, Jun 2008. Article (CrossRef Link)

[10] Mukhtaj Khan, Zhengwen Huang, Maozhen Li, Gareth A. Taylor, Phillip M. Ashton and Mushtaq Khan, "Optimizing Hadoop Performance for Big Data Analytics in Smart Grid," *Hindawi*, vol. 2017, Nov. 2017, Article ID 2198262. Article (CrossRef Link)

[11] Guangdeng Liao, Kushal Datta and Theodore L. Willke, "Gunther: Search-Based Auto-Tuning of MapReduce," in *Proc. of Euro-Par 2013 Parallel Processing*, pp. 406-419, 2013. Article (CrossRef Link)

[12] Jun Liu, Sule Tang, Guangxia Xu, Chuang Ma and Mingwei Lin, "A Novel Configuration Tuning Method Based on Feature Selection for Hadoop MapReduce," *IEEE Access*, vol. 8, pp. 63862-63871, Apr. 2020. Article (CrossRef Link)

[13] Liang Bao, Xin Liu, Weizhao Chen, "Learning-based Automatic Parameter Tuning for Big Data Analytics Frameworks," in *Proc. of 2018 IEEE International Conference on Big Data (Big Data)*, pp. 181-190, Dec. 2018. Article (CrossRef Link)

[14] Xingchen Hua, Michael C. Huang and Peng Liu, "Hadoop Configuration Tuning with Ensemble Modeling and Metaheuristic Optimization," *IEEE Access*, vol. 6, pp. 44161-44174, Aug 2018. Article (CrossRef Link)

[15] Xiang Chen, Yi Liang, Guang-Rui Li, Cheng Chen and Si-Yu Liu, "Optimizing Performance of Hadoop with Parameter Tuning," in *Proc. of ITM Web of Conferences, The 4th Annual International Conference on Information Technology and Applications (ITA 2017)*, vol. 12, Sep. 2017. Article (CrossRef Link)

[16] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, Yingchun Yang, "BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning," in *Proc. of the 2017 Symposium on Cloud Computing*, New York, NY, USA, pp. 338–350, 2017. Article (CrossRef Link)

[17] Narges Peyravi and Ali Moeini, "Estimating runtime of a job in Hadoop MapReduce," *Journal of Big Data*, vol. 7, Jul 2020. Article (CrossRef Link)

[18] Keke Chen, James Powers, Shumin Guo and Fengguang Tian, "CRESP: Towards Optimal Resource Provisioning for MapReduce Computing in Public Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1403-1412, Jun 2014. Article (CrossRef Link)

[19] Ahmed N, Andre L., C. Barczak, Teo Susnjak and Mohammed A. Rashid, "A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench," *Journal of Big Data*, vol. 7, pp. 61351–61365, Dec. 2020. Article (CrossRef Link)
[20] Hassan Tariq, Harith Al-Sahaf and Ian Welch, "Modelling and Prediction of Hadoop Clusters: A Machine Learning Approach," in *Proc. of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pp.93-100, Dec. 2019. Article (CrossRef Link)
[21] Anany Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd Edition, New Jersey, US, Pearson Education, 2012.
[22] Douglas C. Montgomery, *Design and Analysis of Experiments*, 10th Edition, Arizona, John Wiley & Sons Inc., 2019.
[23] Tom White, *Hadoop: The Definitive Guide*, 3rd Edition, CA, US, O'Reilly Media, Inc., 2012.
[24] Dominique Heger, "Hadoop Performance Tuning – A pragmatic & Iterative Approach," *CMG Journal*, vol. 4, pp. 97-113, 2013.
[25] Matthias J. Sax, Malu Castellanos, Qiming Chen, Meichun Hsu, "Performance Optimization for Distributed Intra-Node-Parallel Streaming Systems," in *Proc. of IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pp. 62-69, Jun 2013. Article (CrossRef Link)
[26] Jinsong Yin and Yuanyuan Qiao, "Performance Modeling and Optimization of MapReduce Programs," in *Proc. of IEEE 3rd International Conference on Cloud Computing and Intelligence Systems*, Nov 2014. Article (CrossRef Link)

R. Sathia Priya received her M.E. degree in Systems Engineering and Operations Research from Anna University in 2011. She is an Assistant Professor in the Department of Computer Science and Engineering at Loyola-ICAM College of Engineering and Technology, Chennai, India. Her research interests are Big data and Machine Learning.

A. John Prakash is an Assistant Professor at Ramanujan Computing Centre, Anna University, Chennai. He received his Master's degree and PhD from Anna University, Chennai, India. He has authored a number of scientific publications in journals and conferences. His research interest includes Cryptography, Cloud computing and Optimization.

Dr. V. Rhymend Uthariaraj served as the Director of Ramanujan Computing Center, Anna University, Chennai. His research interests include Computer Networks, Network Security, Pervasive Computing and Wireless Networks. He has authored many research articles and books in his fields of interests. He is serving as an editorial member and reviewer of several reputed international journals.