Ahmed *et al. Journal of Big Data* (2022) 9:67
https://doi.org/10.1186/s40537-022-00623-1

Journal of Big Data

RESEARCH Open Access

Runtime prediction of big data jobs: performance comparison of machine learning algorithms and analytical models

Nasim Ahmed[1*], Andre L. C. Barczak[1], Mohammad A. Rashid[2] and Teo Susnjak[1]

*Correspondence: nasim751@yahoo.com

[1] School of Mathematical and Computational Sciences, Massey University, Auckland 0745, New Zealand
Full list of author information is available at the end of the article

Abstract

Due to the rapid growth of available data, various platforms offer parallel infrastructure that efficiently processes big data. One of the critical issues is how to use these platforms to optimise resources, and for this reason, performance prediction has been an important topic in the last few years. There are two main approaches to the problem of predicting performance. One is to fit data into an equation based on a analytical models. The other is to use machine learning (ML) in the form of regression algorithms. In this paper, we have investigated the difference in accuracy for these two approaches. While our experiments used an open-source platform called Apache Spark, the results obtained by this research are applicable to any parallel platform and are not constrained to this technology. We found that gradient boost, an ML regressor, is more accurate than any of the existing analytical models as long as the range of the prediction follows that of the training. We have investigated analytical and ML models based on interpolation and extrapolation methods with k-fold cross-validation techniques. Using the interpolation method, two analytical models, namely 2D-plate and fully-connected models, outperform older analytical models and kernel ridge regression algorithm but not the gradient boost regression algorithm. We found the average accuracy of 2D-plate and fully-connected models using interpolation are 0.962 and 0.961. However, when using the extrapolation method, the analytical models are much more accurate than the ML regressors, particularly two of the most recently proposed models (2D-plate and fully-connected). Both models are based on the communication patterns between the nodes. We found that using extrapolation, kernel ridge, gradient boost and two proposed analytical models average accuracy is 0.466, 0.677, 0.975, and 0.981, respectively. This study shows that practitioners can benefit from analytical models by being able to accurately predict the runtime outside of the range of the training data using only a few experimental operations.

Keywords: Big data, Performance prediction, Machine learning, System configuration, HiBench, Apache Spark, Extrapolation and interpolation

Introduction

Due to the massive amount of data generated by social media [1], public health [2], industry and natural language processing [3], data storing and processing becomes a challenging task for organisations [4]. The organisations require a fast processing and intelligent system that can quickly process and present the insights of the data. Big data applications have become an ultimate choice in every organisation. There is a number of big data applications available, either in the form of physical clusters or cloud computing. In recent times cloud computing such as Amazon EC2, Google Cloud, Microsoft Azure has attracted tremendous attention. All these platforms allow the users to deploy their cluster virtually where they can choose and allocate resources according to their requirements. This virtualised platform also offers resources at very minimal prices. However, the enterprise needs to consider some data security concerns before selecting cloud computing services. On the other hand, the deployment of the physical Spark cluster is complex and expensive [5]. The physical cluster infrastructures offer numerous benefits and mitigate security concerns.

The deployment of these types of cluster infrastructures heavily depends on distributed parallel computing such as Apache Hadoop and Apache Spark. Due to the open-source, real-time data processing, and fault tolerance [6], Apache Spark has become an attractive framework after Hadoop. Spark supports various components, namely, MLliB for machine learning (ML), GraphX for image processing and Spark SQL [7] for structured data processing. More than 180 configurable Spark parameters play an essential role to support various types of jobs. Though the primary deployment of this cluster depends on the default parameters, however; Spark's performance heavily depends on its correct parameter selection and their configurations. The user must understand the relationship between the parameters and the cluster hardware availability and requirements because the parameter configuration and achieving optimum performance are always challenging and complex. The cluster parameter configuration is tedious work for the users because it requires a vast amount of time to configure and process data.

Due to this limitation, the performance prediction of this system is very challenging. In order to mitigate these challenges, several prediction models such as trial-end-error [8, 9], analytical [10], machine learning [11–14] were proposed by researchers but all these models have limitations; hence, in order to predict runtime for a certain job to run in a Hadoop cluster, one can use machine learning regression algorithms or equation fitting. Both methods need a certain amount of empirical data because there is no general analytic method that would cover different hardware and different configurations for a given cluster. In general, ML regression methods need more data to be accurate, specially if the predictions are made by extrapolation. On the other hand, equation fitting can be very accurate with very little data, but only if the equation reflects the true patterns of inter-node communication that emerges from the job execution. It is difficult to find a generic equation for a cluster because even specific algorithm implementations can influence the communication between nodes, and therefore a given forecasting equation can completely break down for a certain application.

In our previous works [15, 16], we have concluded that two parameters are crucial when determining the runtime: the *size* of the workload, and the *number of executors* available to run the job. We have tested two main models to generate equations that can fit empirical data. The first model assumed that limited communication happens between the nodes, working only with a certain number of neighbouring nodes. The second model assumed that a fully-connected graph between the nodes reflects the communication pattern. Also, in these models the complexity of the algorithm was taken into consideration. Only two types of workloads were tested in terms of complexity of the algorithms, either they were linear or quadratic when considering the growth of the runtime as a function of the workload size.

The motivation and the key contribution of this paper are as follows:

- We accomplished extensive performance prediction accuracy comparison based on machine learning and existing analytical models. We achieved very good accuracy when only limited empirical data is available. Our underlying intention is that practitioners would run a few jobs, preferably with short runtimes, and be able to predict the runtime of longer untested dataset sizes.
- We investigated KRR regression parameter relationship between alpha and degree. Our analysis found that, for most of the workloads, the best R-squared can be achieved by selecting the small degree with alpha. Our analysis also found higher degrees can produce best R-squared but the data overfitting can be a major limitation. For the GBR regression, we kept all the parameters default.
- We extensively measured the analytical and ML regression models accuracy based on interpolation and extrapolation methods using k-fold cross validation. ML methods are not accurate when one tries to extrapolate predictions from small amounts of data. However, ML methods are much better at making adjustments to existing data, and can do interpolation very well [17]. The equations are derived to fit data well, and using the correct one can yield more accurate extrapolations of runtime forecasts than ML methods.

The remainder of the paper is organised as follows: "Apache Spark architecture" section provides a brief overview of the Apache Spark architecture. "Related work" section discusses some notable recent advances on Spark performance prediction using machine learning algorithms. "Prediction methods" section explains evaluation methods of both the analytical models and the ML regressors. "Experimental setup" section discusses the experimental setup while "Performance evaluations and analysis" section presents the performance analysis for the two approaches using interpolation method with cross-validation technique. "Performance analysis using extrapolation" section shows a detailed analysis of the extrapolation method, splitting the data into two categories, size and number of executors. "Discussion" section discusses the limitations of each approach, and the consequences of extrapolating data with a small number of experiments. Finally, "Conclusion" section concludes the paper with hints for extending the work in future.

Apache Spark architecture

Apache Spark is a parallel data processing framework that can rapidly process large amounts of data, often in real-time [18]. It can also perform data processing in the distributed cluster platform. Apache Spark has become an open access [6] project, and a popular data processing engine in many organisations. Its development has centred at the University of California, Berkeley's AMPLAB by the group of researchers that Matei Zahari led in 2009 [19]. The Spark codebase is donated to the Apache foundation as an open-source tool and has been maintained since then. In 2010, Spark proposed a Resilient Distributed Dataset (RDD) [20] that mitigates the limitation of the MapReduce cluster computing paradigm. It works as an immutable collector of the objects. RDD splits the input data set into logical partitions and the partition data stored in the memory where worker nodes compute parallel operations. Spark RDD has two operations: transformation and actions. The transformation function uses the existing RDD as input and produces the new RDD from the existing one. Whenever the transformation function becomes active, it creates a new RDD. The action operation activates when it works on the actual dataset. A typical Apache Spark architecture representation is shown in Fig. 1.

The Spark application has three important components: the spark driver program, Spark executor, and the resource manager, where the action operation is forwarded from the executor towards the driver. The driver converts the user code in most tasks, and the executors run the code among the nodes. In this operation, the cluster manager is responsible for the resource allocation in the cluster. The cluster manager allocates the resources whenever the Spark driver program [21] requests and shares the information with the worker nodes. In Spark, the workflow is managed by a directed acyclic graph (DAG) [22]. The DAG consists of sequences of vertices and edges. The vertices represent the RDDs, and the edges represent the operation of the RDD. The DAG forwards the new job towards the stage level. The task consists of the initial input data and the RDD partition at each stage level. Spark creates two stages with the submitted job; firstly, ShuffleMapStage and secondly, ResultStages. At the ShuffleStage, the output data is stored for the following stages in the DAG. At the ResultStage, either single or multiple partitions functions are targeted the RDD. Spark can operate with many programming languages, such as Java, Scala, Python and R, and supports Spark SQL, ML, GraphX processing, and Spark Streaming. These programming language libraries offer comprehensive benefits for the user to develop applications. Spark allows the integration of various tools from the Hadoop technology ecosystem, where the resource management and job scheduling is maintained by Apache YARN (Yet Another Resource Negotiator) [23]. A cluster monitoring tool like Ambari assists with the monitoring the workloads running in the cluster.

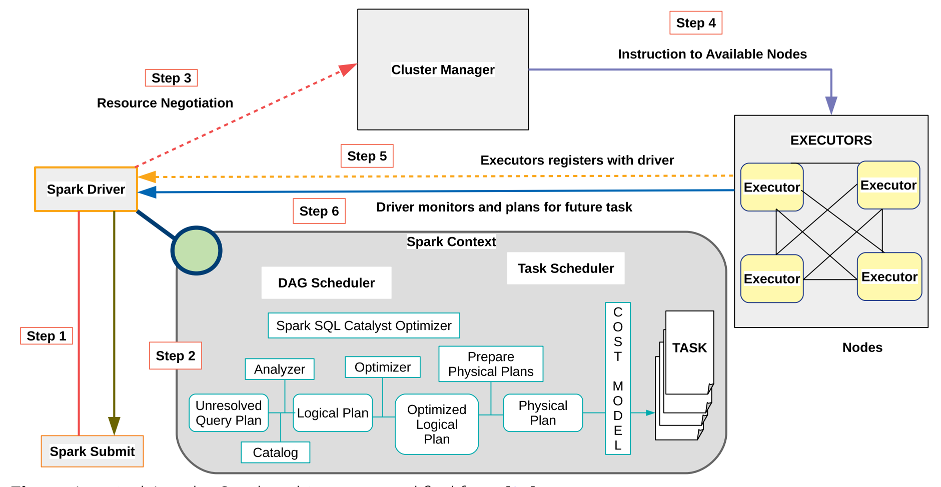

Fig. 1 A typical Apache Spark architecture modified from [21]

Related work

The runtime performance prediction of big data processing on a cluster is a challenging task. In the recent past, many prediction techniques [8–10, 24], Gray-Box techniques [25–27] and auto tuning techniques [12, 13, 28–30] have been proposed by researchers. However, the ML approach has become very popular and has received significant attention. In the following section, we will present recently published works based on ML techniques.

Prediction using machine learning

Douglas de Oliveira et al. [31] proposed an interpretable predictive ML model based on decision trees from which patterns are extracted. The decision tree model is used to classify the parameter performance by considering the training data. They used the extracted patterns and configured the system parameters for the workflow execution that significantly improved the system performance. Besides, they also considered two essential aspects: input data partitions and distributed data partitions through nodes. They found that the proposed predictive model can achieve 70% accuracy, and the accurate data partitioning knowledge can help choose the workflow function.

Christoph Boden et al. [32] presented an interesting work using ML algorithms for a large-scale distributed settings of Apache Spark and Flink performance. They implemented analytical models which are similar to ML algorithms and tuned the parameters to assess the scalability of the system concerning the data size and dimensionality of the data. They carried out a comprehensive investigation based on a single-node implementation with data size and data dimensionality. They found that several ML algorithm problems exhibit high dimensionality due to data scaling and model size scaling. So, they employed both supervised learning algorithms (batch gradient descent and TreeAggregate) for Flink and Spark, respectively. For the unsupervised learning algorithm, kmeans clustering was used. The proposed benchmark algorithm was placed on top of Apache Flink and Apache Spark and analysed the performance using non-representative workloads such as Wordcount, Grep, and Sort. They found that when the data size increased, the system behaviour exhibited a linear increment. They concluded that the system can perform robustly with the increasing data size of both Flink and Spark with 4.6 billion data points. Spark fails to train when the data size is beyond 6 million dimensions for scaling the model dimensionality. They concluded that current data flow systems could process an increased amount of data points but are incapable of coping with high dimensional data, which is a key requirement for large scale ML algorithms.

Christoph Boden et al. [33] proposed a novel data processing system based on a ML algorithm in their second work. This work categorized the proposed data processing system into three major groups: Clustering, Classification, and Recommender Systems. The raw data is transformed into extracted features for the data pre-processing, and the training data set is represented by a numerical data matrix. For this implementation, they have used kmeans, Batch Gradient Descent, and Matrix Factorization algorithms. As per their suggestion, logistic regression is a compelling choice for the prediction problem that can easily handle many data sets. They concluded that the latest data processing system requires more hardware resources to obtain a comparable prediction quality.

Ali Mostafaeipour et al. [34] presented an empirical analysis of the Hadoop and Spark frameworks that considers three criteria such as runtime, memory, and network usages. They implemented the K-nearest neighbour (KNN) algorithm on various data-sets for both frameworks. This analysis demonstrated that with small data sets, Spark offers faster data processing than Hadoop. They also found that Spark is suitable for quick data processing because it processes the data in-memory. As for memory utilisation, Hadoop requires less memory than Spark, and Spark requires fewer network usages than Hadoop. Another empirical study of Apache Spark performance prediction based on ML algorithms is proposed by Mehdi Assefi et al. [35]. The authors have examined both qualitative and quantitative attributes of the framework. This study leverages the Apache ML library to handle big data analytics and evaluate the impact of multiple big data ML models such as classification and clustering on the different hardware and software configurations with big data analysis tasks. Some ML algorithms such as Support Vector Machine, Decision Tree, Naïve Bayes, Random Forest, kmeans are evaluated to analyse the ability of MLlib 2.0. They found that Apache Spark MLlib demonstrates better performance; in particular, this presented a noteworthy performance in terms of execution time.

Javaid [36] proposed a robust Spark performance prediction model based on ML algorithms. In this analysis, authors offered substantial experimental works and their applications with various data features. In order to build the performance model, they implemented four ML algorithms. They found that the gradient boost and Random Forest algorithm showed a better performance than the other algorithms on their datasets. In [37], the authors proposed a tool to predict the Spark application runtime before the deployment of the cluster. They claimed that the tool can be used for extensive Spark job profiling, determining the prior execution time and the system bottleneck. They claimed that the tool could predict a 20% error bound for the selected workloads. In [38], the authors proposed a ML-based auto-tune model for cluster parameter selection based on the Support Vector Regression (SVR) model and a practical end-to-end auto-tuning model by combining existing models with a smart search algorithm. They found that the overall performance of ML is much better than the traditional models. In particular, the SVR displayed the best performance for Sort. They concluded that the proposed model is robust and flexible, and adaptable to any changes.

Guoli Cheng [12] proposed a model based on the Adaboost ML algorithm. Adaboost is implemented at the stage level, and the classic projective sampling, including the data mining technique was applied to predict the Spark performance accurately. They used six benchmark workloads and five different data sizes. They concluded that the proposed model minimizes 9% runtime cost as compared to the previous model. In their recently published work [13], they stated that the performance trade-off heavily depends on the optimum configurations where the cost is an influential factor. So, they proposed a multi-object optimization algorithm model based on the Adaboost ML algorithm for Spark performance prediction. They applied six benchmark workloads and five different data sizes to evaluate the system performance. They claimed that the model can find the appropriate configuration setup and minimize the time and cost. They also concluded that the proposed method can improve execution time performance by 30% and cost by 40%.

Table 1 summarises some notable studies by considering the models used and their performance based on selected workloads. It can be noted that most of the works used ML and very few works proposed analytical models, but the workloads and model performance metrics are not similar in these works, which make it difficult to compare the accuracy between them. To the best of the authors' knowledge, the literature has not presented any comparative performance analysis based on standard performance metrics because no standard performance metrics have been recommended.

Unlike the reviewed ML models described in the related work section, we compare analytical models [15] with ML (kernel ridge regression (KRR) [39] and Gradient Boost Regression (GBR) [40]), ERNEST [41], Amdahl [42] and Gustafson [43] models. The runtime prediction based on ML models shows satisfactory performance as per the published work, but all ML models require large input data. On the other hand, we have seen that our published models 2D-plate model (4) and fully-connected (5) model are very effective and can predict runtime accurately with limited data points.

Table 1 Various models on Spark performance prediction

Published work	Workloads and data sets	Models	Metrics (error and accuracy)
Cheng [12]	WordCount, Kmeans, TeraSort, PageRank, Bayes, and Nweight	Adaboost, ensemble learners, multiple learners, and projective sampling	Average accuracy error: Adaboost (30 cases): 9.02%, ensemble learners: 18.63%, multiple learners: 21.98%, projective sampling: 14.09%
de Oliveria [31]	Data sets: astronomy and bioinformatics Data partitions: 3	Decision tree (DT)	Prediction accuracy: best 3 scenario out of 7: SC1: 90.4%, 88.8%, and 86.5%
Boden [32]	WordCount, Grep, and Sort	Logistic regression (LR), and Kmeans	High data dimensionality
Boden [33]	Data set: CriteoClick Logs and Netflix Prize Kmeans, logistic regression (LG), matrix factorization (MF), and gradient boost regression (GBR)	Kmeans, logistic regression (LR), matrix factorization (MF), and gradient boost regression (GBR)	MF: required more time than single LibMF, LR: Spark MLlib required more hardware resources, GBR: better than LR
Assefi [35]	Data sets: HEPMASS, SUSY, HIGGS, LIGHT, HETROACT I and II	Support vector machine (SVM), decision tree (DT), Kmeans, NaıveBayes (NB), Weka, and random forest (RF)	t-test: $p < 0.01$
Javaid [36]	KMeans, PageRank, sorting, WordCount, binomial logistic regression, linear regression, groupby decision tree classifier, single source shortest path, and breadth first search	Linear regression (LR), random forest (RF) gradient boost machine (GBM), and neural networks (NN)	Average accuracy error: LR, GBM, RF, and NN 10% (approx)
Singhal [37]	Wordcount, Terasort, Kmeans and SQL	Multi linear regression (MLR), MLR-quadratic (MLRQ), support vector machine (SVM), and analytical model	Prediction accuracy error: MLR, SVM, and, MLRQ: MAPE 22%, analytical models: 80%
Cheng [13]	WordCount, Kmeans, TeraSort, PageRank, Bayes, and Nweight	AB-MOEA/D, random forest (RF), and two-stage tree (TSt)	Prediction error: AB-MOEA/D: 3.6%, RF: 8.97%, TSt:14.57%

Prediction methods

Machine learning algorithms

Many studies have explored supervised ML models for runtime performance prediction of large systems. These techniques are known as black box solutions because they can make predictions on previously collected data. In this supervised ML model, the training phase uses the experimental data that comes according to system configuration parameters. Indeed, the collection of these data is tedious and requires significant time resources. In this paper, we used two regression algorithms, kernel ridge regression (KRR) and Gradient Boost Regression (GBR), and implemented them based on Sklearn implementation. We choose gradient boost algorithm because it is a popular method for a large cluster setup [44], whereas the kernel ridge regression can perform cross-validation and predictive variance more efficiently on small and large data [45]. In this implementation, the Python programming language is used to evaluate the models' performance. We introduce the algorithms and their model operation principles in the following section.

1. Kernel ridge regression

   In 2000, Cristianini and Shawe-Taylor [39] proposed the kernel ridge regression (KRR) algorithm. KRR combines ridge regression with the kernel trick. For the linear kernel, this communicates with the linear function in the space induced by the respective kernel and data but for the non-linear kernel, this communicates with the non-linear function. The KRR is a simplified version of the Supervised Vector Regression (SVR), and it is also known as the least square support vector machine (LS-SVM). It uses different loss functions and twelve regularisations. Regularisation always uses positive floating point values, improves the problem complexity, and minimises the estimates' variance. In KRR, the kernel mapping works internally, and the parameters are passed through the pairwise kernel. A kernel function expressed as: $K : \mathcal{X}x\mathcal{X} \rightarrow \mathcal{R}$, is a function that is symmetric to $K(x_1, x_2) = K(x_2, x_1)$ and positive definite. In this study, we employed the Polynomial kernels from the Sklearn implementation [46]. In the Polynomial kernel [47], the assigned values and its distance calculate as per their assigned values where the parameter values must be positive. We can express the polynomial kernel expression as follows: $Parameters : \alpha, c, d$ and the kernel function: $K(X_1, X_2) = (\alpha_{X_1^T X_2} + c)^d$.
2. Gradient boost regression

   The Gradient Boost Regression (GBR) algorithm is a popular algorithm used for building predictive models and for large cluster setups [44]. In 2002, Friedman [40] proposed a modified version of the GBR algorithm based on a regression tree of fixed sizes. At the regression problem, the boosting approach works as a form of "functional gradient decent". The boosting approach is an optimisation technique that minimises the loss function of the training data. In this case, the loss function measures the difference between the predicted values and training data. GBR algorithm generates the learners iteratively by combining the weak learners into a single strong learner. The fixed size of multiple decision trees is used as a weak learner to build the GBR. In this study, GBR is used the default parameters within the sklearn [48] implementation to evaluate the results. The GBR can be used in two ways, either as a regressor or classifier, with the former used in this study to predict the system runtime data.

Prediction models based on specific equations for parallel systems

For any parallel system, including Hadoop clusters running Spark, two parameters are the most influential in determining runtime: size and number of executors. In Spark, other parameters can deteriorate the performance. However, once these parameters relinquish enough resources for running a certain job, they do not have the ability to speed up the execution of that job. Therefore, while most parameters have a minimum threshold for the job to use the cluster's resources appropriately, one cannot improve the performance of a job beyond a certain point, limited by other factors such as size and number of executors available [9].

Also, in any parallel system the runtime has two components: parallelisable and non-parallelisable portions of time [49]. The parallelisable portion can be found as a function of the size of the job and the number of executors used. The non-parallelisable portion is more difficult as it depends on implementation and communication between nodes.

Since the early days of parallel systems, several models have been proposed for equations that can drive the runtime. Three important ones are Amdahl's law, Gustafson and Ernest. In our previous works, we have proposed two new models [15, 16] and have tested them against Amdahl [42], Gustafson [43] and ERNEST [41]. In order to compare the models, we adapted Amdahl's law and Gustafson's law as equations that determine runtime given the size and number of executors. These models use simple equations that can fit experimental data, and can be used to predict the runtime of jobs for different clusters, with specific hardware.

For completion, we summarise each model and the corresponding equations. For all the equations in this section, the following notations apply:

- S is the size of the workload (usually in GB),
- $f(S)$ is the function that expresses the runtime complexity of the algorithm,
- E is the number of executors, and
- a, b, c, d are the coefficients of the equations that need to be found via data fitting.

For $f(S)$, all the workloads used in this work were either linear or quadratic. Therefore, either $f(S) = S$ or $f(S) = S^2$. If the time complexity of the algorithm is known, its equation can replace $f(S)$.

Amdahl's law

In the early days of parallel systems, Amdahl proposed a performance model where the number of executors and the percentage of the non-parallelisable time drives the speedup of a job running with multiple executors when compared to the same job running on a single executor [42]. The equations can be modified to predict runtime given S and E:

$$runtime = af(S)\left(\frac{(1-b)}{E} + b\right) + c \tag{1}$$

Gustafson's law

Gustafson proposed an alternative model to that proposed by Amdahl [43]. The modified equation to predict runtime is:

$$runtime = \frac{af(S)}{E + b(1-E)} + c \tag{2}$$

ERNEST

More recently, Venkataraman et al. [41] proposed a model specifically for big data clusters called ERNEST. Their equation to predict runtime is:

$$runtime = \frac{aS}{E} + b\ log(E) + c\ E + d \tag{3}$$

2D-plate model

We proposed a 2D-plate model where the nodes communicate only with its direct neighbours [16] This model was based on insights by Wilkinson and Allen [49] that can be found in chapter 6, sections 6.3.2 and page 180. The details of how we derived equation (4) are in [16]. The equation is:

$$runtime = \frac{af(S)}{E} + b\ S\ E^c + d \tag{4}$$

Fully-connected node model

We also proposed an alternative model where the communication between nodes is assumed to work like a fully-connected graph. Both the 2D-plate and the fully-connected models were as accurate or more accurate than alternative models [15]. The equation for the fully-connected model is:

$$runtime = \frac{af(S)}{E} + b\,S^c\left(\frac{E(E-1)}{2}\right) + d \tag{5}$$

A special case of this equation was considered when the communication growth is linear in relation to the size S, i.e., $c = 1$:

$$runtime = \frac{af(S)}{E} + b\,S\left(\frac{E(E-1)}{2}\right) + d \tag{6}$$

Experimental setup

All our experiments have been conducted on a high-end Hadoop cluster. In 2016, the group of academicians and researchers designed and developed the cluster at Massey University, Auckland campus. This cluster is designed with a dedicated switch and

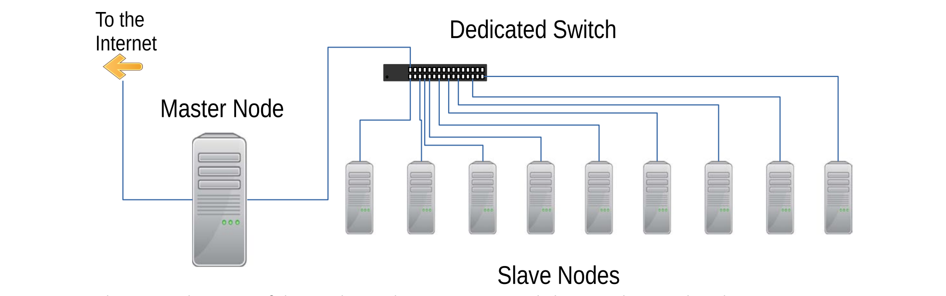

Fig. 2 A schematic diagram of the Hadoop cluster master and slave nodes used in the experiment

Table 2 Experimental configuration of the Hadoop cluster

Server configuration	
Processor	2.9 GHz
Main memory	64 GB
Storage	10 TB
Node configuration	
CPU	Intel (R) Xeon (R) CPU E3-1231 v3@3.40 GHz
Main memory	32 GB
Number of nodes	9
Storage	6 TB each, 54 TB total
CPU cores	8 each, 72 total
Software	
Operating system	Ubuntu 16.04.2 (GNU/Linux 4.13.0-37-generic x86 64)
Hadoop	2.4.0
Spark	2.1.0
JDK	1.7.0

different network infrastructures, similar to a Beowulf cluster [50]. In order to reduce the network latency and unwanted network resource utilization, all other network machines were isolated from this infrastructure.

A schematic diagram of the cluster is presented in Fig. 2, and the specifications for the servers and nodes are presented in Table 2.

HiBench workloads

It is a challenging task to evaluate the performance of a cluster. In the recent past, researchers presented CloudSuite [51] and CloudStone [52] benchmarks for cluster performance. Intel also proposed a Hibench suite under Apache Licence HiBench suite [53]. Since then, this has become a heavily used cluster performance testing tool, especially for Hadoop and Spark frameworks. The existing benchmark can be divided into three categories, such as Micro-Benchmarks, End-to-End benchmark, and Benchmark suite [54]. The Hibench suite has also been categorised into four categories: Micro-Benchmark, Web Search, SQL, and ML. In this experiment, there are five different workloads,

Table 3 Spark HiBenchmark workload considered for this study

Benchmark categories	Application	Input data size		Input samples
		Multiple-Exec.	Single-Exec.	
Micro benchmark	WordCount	313 MB, 940 MB, 5.9 GB, 8.8 GB, and 19.2 GB	3 GB, 5 GB, 7 GB, 10 GB, 12.8 GB, 14.4 GB, 16 GB, 18 GB, and 21.6 GB	–
Machine learning	kmeans	19 GB, 56 GB, 94 GB, 130 GB, and 168 GB	1 GB, 38 GB, 75 GB, 113 GB, 149 GB, and 187 GB	10, 30, 50, 70, and 90 (million samples)
	SVM	34 MB, 60 MB, 1.2 GB, 1.8 GB and 2 GB	200 MB, 400 MB, 600 MB, 800 MB, 1.35 GB, 2 GB, 2.3 GB, and 2.5 GB	2100, 2600, 3600, 4100, and 5100 (samples)
Web search	Pagerank	507 MB, 1.6 GB, 2.8 GB, 4 GB, and 5 GB	100 MB, 250 MB, 750 MB, 6 GB, 7 GB, 8 GB, 9 GB, and 10 GB	1, 3, 5, 7, and 9 (million of pages)
Graph	NWeight	37 MB, 70 MB, 129 MB, 155 MB, and 211 MB	20 MB, 55 MB, 99 MB, 141 MB, 175 MB, 214 MB, 247 MB, 262 MB, and 286 MB	1, 2, 4, 5, and 7 (million of edges)

Table 4 Workload application characteristics

Workloads	Stages	Parallel stages	Collect	Serialization	Deserialization	Shuffle	Aggregate
WC	2	No	Yes	–	–	Yes	–
SVM	209	No	Yes	No	Yes	Yes	Yes
Nweight	9	Yes	–	No	Yes	Yes	–
kmeans	20	No	Yes	Yes	Yes	Yes	–
Pagerank	5	No	–	No	Yes	Yes	–

comprising WordCount, kmeans, SVM, Pagerank, and NWeight. From four categories: Micro-Benchmark, ML, Web Search and Graph were taken into consideration. Table 3 presents the Spark Hibench workloads while Table 4 presents the workload application characteristics.

Cluster parameters configuration

In this work, a set of configuration parameters are considered to evaluate the performance of the system. Spark has more than 150 configurable parameters [8, 9] where the system performance heavily depends on the correct parameters selection. Therefore, we have judiciously selected only the parameters that are closely bound to system performance for evaluation purposes. However, cluster performance depends not only on the right parameters selection but also on tuning the parameters to achieve optimum system performance. We have seen the configuration of these parameters heavily depends on the cluster hardware, workload characteristics and size of the workloads. Out of numerous parameters, the most important parameters are the number of executors, executor memory, executor core size, and the driver memory. This experiment has therefore chosen a subset of only impactful parameters and tuned their values to achieve the best cluster performance.

Recently, several notable studies [26, 55] presented the importance and effectiveness of the outlined tunable parameters. Our study revealed that the right parameters selection is the primary requirement to get the best cluster performance. In our work, the chosen parameters are listed in Table 5. It can be seen from Table 5 that the default column presents the system default parameters, we tuned several parameters values including the default values that are listed in the range columns. Our investigation found that in most of the cases, the default values are not appropriate for our cluster performance. In some cases, for example *Spark.memory.fraction and Spark.memory.storageFraction*, there is no performance difference even if we use lower than the default values. On the other side, for example, *Spark.driver.memory, Spark.driver.cores, Spark.shuffle.file.buffer, Spark.reducer.maxSizeFlight*, the higher values showed better performance than the default values. Therefore, we have considered only those tuned values that are listed in the column of value used in the experiment column. The description of the parameters is presented in the description column. Our insight on the cluster performance, the selection of these parameters and their values are firstly based on the fact that the spark performance heavily depends on the available resources of the hardware. Secondly, these parameters and their values were chosen

Table 5 Description of selected Spark configuration parameters selected as the input of the proposed model

Parameters	Default	Range	Value used in the experiment	Description
Spark.executor.memory	1	1–12	12	Amount of memory to use per executor process, in GB
Spark.executor.cores	1	2–14	2–14	The number of cores to use on each executor
Spark.driver.memory	1	1–4	4	Amount of memory to use for the driver process, in GB
Spark.driver.cores	1	1–3	3	The number of cores to u for the driver process
Spark.shuffle.file.buffer	32	32–48	48	Size of the in-memory buffer for each shuffle file output stream, in KB
Spark.reducer. maxSizeInFlight	48	48–96	96	Maximum size of map outputs to fetch simultaneously from each reduce task, in MB
Spark.memory.fraction	0.6	0.1–0.4	0.4	Fraction of heap space used for execution and storage
Spark.memory. storageFraction	0.5	0.1–0.4	0.4	Amount of storage memory immune to eviction expressed as a fraction of the size of the region
Spark.task.maxFailures	4	4–5	5	Number of failures of any particular task before giving up on the job
Spark.speculation	False	True/false	–	If set to "true", performs speculative execution of tasks
Spark.rpc.message. maxSize	128	128–256	256	Maximum message size to allow in "control plane" communication, in MB
Spark.io.compression. codec	Snappy	lz4/lzf/snappy	Snappy	Compress map output files
Spark.io.compression. snappy. blockSize	32	32–128	32	Block size in snappy compression, in KB

since they control pivotal resources such as CPU, disk read and write, and memory. [56].

Performance evaluations and analysis

In this section, we present the comparative results between the analytical and ML models. To validate the system performance, we used five HiBench workloads with various data sizes. The system runtime characteristics are obtained by running jobs for five workloads using different number of executors and data sizes.

The proposed work is categorised into six stages: loading runtime data, data preprocessing, cross-validation and extrapolation methods, proposed models, ML models, and lastly, performance measurements (Fig. 3). To avoid overfitting and selection bias, a threefold cross validation process was used. The workload execution time is extracted from the Ambari history server log files, where a Python script is used to calculate the workload execution times. For the final graph presentation, each experiment was repeated at least three times, and the average time is considered as a final result.

We calculate the job execution time based on the job log files. We have collected all job log files from the Ambari history server and used a Python script to calculate the execution time. We found a fraction of time difference between the Python script and the Ambari server. One of the possible reasons for this time difference, Python scripts calculate log files independently while Ambari saves the execution time into the server, where the network latency can play an important role. In stage two of Fig. 3, data prepossessing is an essential step to achieve the best results from the models. Therefore, well-structured data is required to get the best performance from the models.

In stage three of Fig. 3 two types of data split were used. For the threefold cross-validation, a balanced split was used, where in both training and test data all sizes and number of executors are present in the data. For the extrapolation split, the training data receives all measured points in the middle of the range for either size or number of executors, also using the same proportion of data for the training set, with 66% of the data, and the test set, which uses the remaining 34% of the data.

In stage four, we applied the fitting to the equations of the analytical models (proposed and from the literature), and use two ML regression algorithms, namely KRR

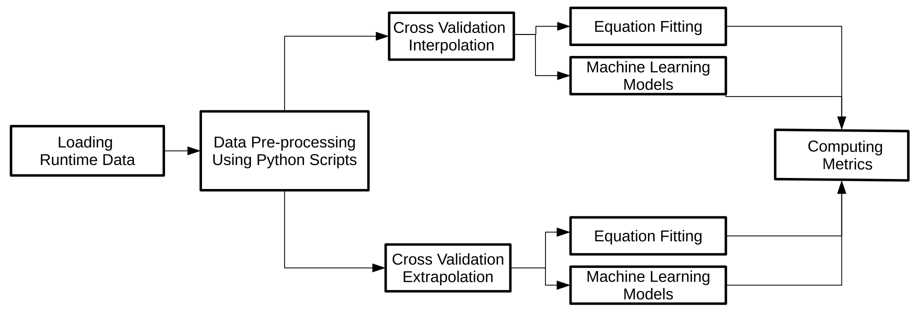

Fig. 3 The workflow of the performance analysis

and GBR. Finally, the performance of the models and ML regression is measured based on R-squared and Residual Relative Square Error (RRSE). Only the most accurate results have been used to present the graphs in Figs. 6, 7, 8, 9, 10.

Evaluation metrics

The choice of model evaluation metrics is an important factor for conducting comparative analysis. To verify performance in literature, researchers lean on a variety of metrics. In this study, the R-squared (R^2) and relative residual standard error (RRSE) are used. R-squared is used as a dominant index in regression algorithm to verify the predicted results accuracy, and RSE is used to determine the goodness-of-fit. The R^2 values (also known as Correlation Coefficient(R)) are presented as follows.

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}} \tag{7}$$

where SS_{rs} is the sum of the squares of the residuals and and SS_{tot} is the sum of the squares relative to the mean of the data. R-squared value is between 0 and 1. Higher values indicate a more optimal fit. The residual standard error is represented as follows:

$$RSE = \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{df}} \tag{8}$$

where $(y_i - \bar{y}_i)$ is the difference between the observed data and the predicted value using the model, and df is the degrees of freedom given by the number of sample size minus and the number of parameters being fitted. The relative standard error (RSE) is:

$$RRSE = \frac{RSE}{\mu} \tag{9}$$

The Residual Relative Standard Error ($RRSE$) metric allows us to distinguish the error between the observed points and the ones generated by the model. The smaller the $RRSE$, the better the fit accuracy.

Kernel ridge models

We used KRR from scikit-learn [46] implementation where only the polynomial kernel is considered, and others are ignored. For the model simplicity, we kept the coefficient and gamma parameter as a $default = 1$, and no other parameters but the different degrees and alpha values are examined to improve the model's accuracy. It can be noted that the proposed model produced the best R-squared results when $degree = 70$. Table 6 presents the best results of the individual workloads by measuring R^2, standard deviation and relative residual standard values (RRSE). Our study found that except for the Kmeans and Graph workloads, all three workloads produce the best results with degree 70 and alpha 1. In contrast, the Kmeans and Graph show the best results with $degree = 30$ and alpha value always achieves a better performance with $alpha = 1$. Our study also revealed that the small values of alpha improves the model performance and reduces the variant of the estimates for three workloads. We noticed that the model performance for the individual workloads is satisfactory. The R-squared comparison of KRR algorithm across different degrees and alphas on the performance of selected HiBench workloads are shown in Fig. 4.

Despite showing a better R-squared value for higher degrees, polynomial regression has a known issue related to over fitting [57]. We showed the results of higher degrees to make the point that ML approaches using polynomials and easily overfit. One can see that the fitting follows the experimental data very well, and it can encompass the full range of the parameters. However, when showing an example of fitting, one can see that the higher degree polynomial can create instability in the model. In Fig. 5 four KRR fittings show values interpolated between the experimental data for different sizes. The interpolation points are in the middle of the measured data points. What can be clearly seen in the higher degree polynomials is that the interpolated data extends out of the value range of the plot, even though the training and test data are still well within the model prediction. This means that even with a high R-squared value, the prediction of new sizes can be very inaccurate. For this reason, we kept the degree to a maximum of 4 in the experiments in "Performance comparison of ML and analytical models" and "Performance analysis using extrapolation" sections.

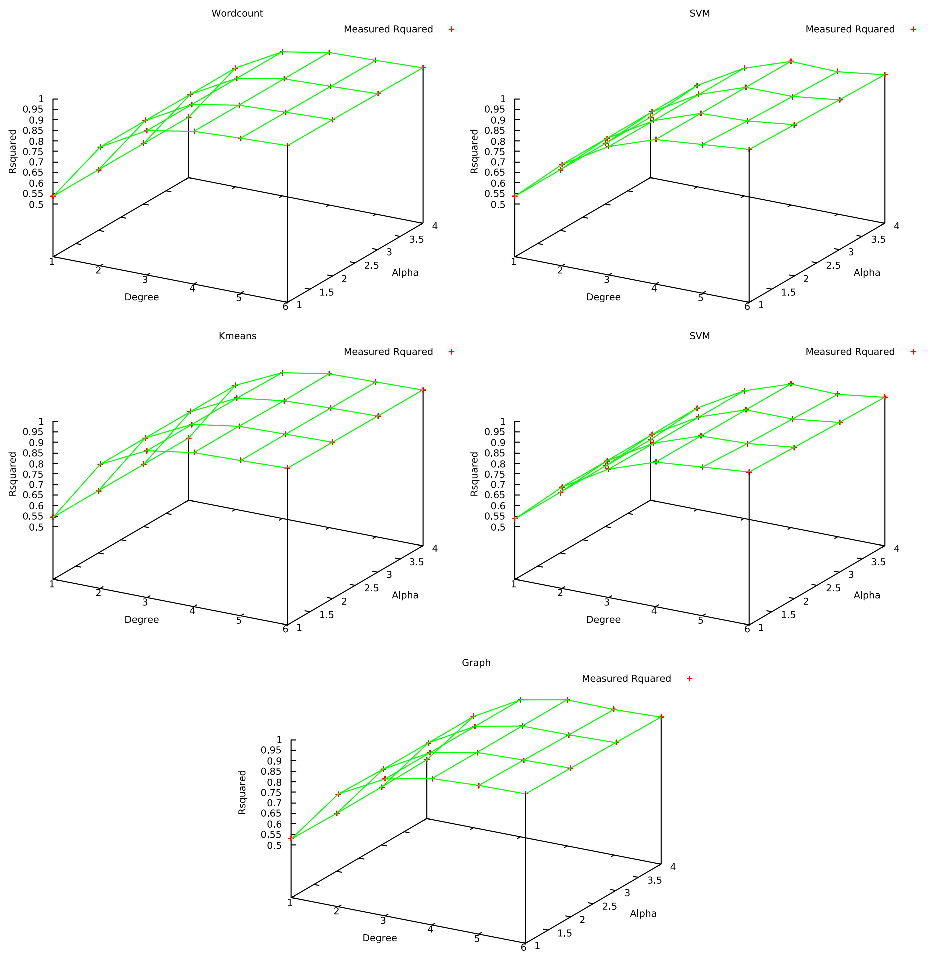

Fig. 4 Performance comparison of KRR algorithm across different degrees and alphas on the performance of selected HiBench workloads

Gradient boost models

GBR algorithm is a popular technique used for building prediction models. GBR uses the forward stage-wise fashion technique with the optimization of arbitrary differentiable loss functions. In this experiment, we used GBR from scikit-learn [48] with the default parameters. We obtained the best results with default random state 1 for all five workloads. However, we further investigated the model by increasing the random state, but the results were unsatisfactory and out of scope for inclusion in this study. In Table 7, the statistical results are shown based on R^2, standard deviation, and RRSE scores.

Performance comparison of ML and analytical models

This section illustrates the proposed model accuracy in terms of R^2, standard deviation and the RRSE values. It shows the performance comparison results between well-known parallelisation models (Amdahl's and Gustafson, ERNEST) and ML algorithms such as KRR and GBR where KRR algorithm parameters were optimised. The KRR optimised parameters assist the model to maximise the performance accuracy of the model. The k-fold cross-validation technique was applied with all the models to achieve highest prediction accuracy. We obtained all the table results and figures using cross-validation with $k = 3$. The individual workload's best results obtained by the proposed models against the ML models are described in the following section.

Wordcount

The wordcount workload comparative statistical results between ML and analytical models with cross validation, are presented in Table 8. From this analysis we can see that except ERNEST, all analytical models' accuracy is 0.995, which is better than the KRR algorithm of 0.974. The GBR algorithm shows better performance as compared to others, where the accuracy is 0.998. The best analytical results and GBR results are presented in Fig. 6. The RRSE results in Table 9 show very low accuracy of GBR algorithm presenting the best fit in the results.

SVM

The comparison between ML and analytical models with cross validation for SVM workloads runtime prediction results are shown in Table 8. In this workload, both ML algorithms show significantly better results than the analytical models where the GBR algorithm completely outperforms the KRR algorithm. It may be noted that the GBR accuracy and RRSE is 0.995 and 0.064 respectively with corresponding standard deviations of 0.001 and 0.010. The comparative best results are plotted in Fig. 7.

Pagerank

The Pagerank performance evaluated in terms of ML algorithms and analytical models with cross-validation approach in Tables 8 and 9. Our results revealed that the model (Eq. 4) either outperforms or equal to all analytical models and KRR algorithms, but the

GBR algorithm achieves the best results among models. In Fig. 8, the best results are plotted from GBR and Eq. (4).

Kmeans

The comparative performance measurement results of ML algorithm and analytical model for kmeans workload are presented in Table 8. The GBR algorithm is the most effective model that shows the best accuracy and produces low RRSE among all the models. It can be noted that analytical models outperform KRR algorithms where the accuracy is 0.981, but analytical models are better with a higher margin of accuracy of 0.992. Among the analytical models, their performance is either equal or slightly different. For example, the Gustafson accuracy is equal among analytical models, but the RRSE is slightly better, as shown in Table 9. The best-obtained results among models are shown in Fig. 9.

Graph

The ML algorithms (KRR and GBR) show excellent results using Graph workload. From the statistical results shown in the Tables 8 and 9, the GBR algorithm records the best results among analytical models and outperforms KRR. Equation (4) indicates a significant performance improvement among the analytical models where other equations previously proposed by us clearly defeat ERNEST, Amdahl and Gustafson models. The best results from the ML model and an analytical model is shown in Fig. 10.

In summary, the above results demonstrate model performances for the selected workloads. The GBR algorithm achieves an excellent performance in comparison to all models. On the other side, Eqs. (4) and (5) show excellent results among analytical models and both equations are better than KRR for WordCount, SVM, PageRank and Kmeans workload. For the graph workload, both ML algorithms demonstrated the best results. The above analysis shows the effectiveness of the analytical models over ML approaches.

Performance analysis using extrapolation

Data obtained from the results in "Performance comparison of ML and analytical models" section is used for the performance analysis using extrapolation in this section. However, rather than carrying out cross validation on the entire dataset, we reserved part of the data set to test how well each model can deal with extrapolation. This is a very important aspect of the prediction models, as extrapolation would allow practitioners to make accurate predictions with a very small number of experiments when using a specific cluster and workload. Our hypothesis was that despite the better results for the models using ML presented in "Performance comparison of ML and analytical models" section, equations that can represent the cluster behaviours could be more accurate for extrapolation. In other words, ML is an effective approach for predictions that fall within the range of values captured in the existing data, but can yield poor results if not enough data is available to describe runtimes beyond a given range. In these scenarios, ML methods may not be able to predict the actual pattern of communication that drives the runtime.

Due to limited data points, we employed the linear extrapolation approach to estimate the data values that are close to the existing data. Generally, it has been proven that nonlinear model accuracy is higher than the linear models because it is more likely to overfit the training data set, which shows the poor performance of the models [58]. In contrast, the linear models fit the data more accurately; thus, better fitting can be achieved from the unseen data. We observe from the presented results in Table 10 that the performance of the linear workloads is better than the quadratic workloads. Two extrapolation scenarios were considered: extrapolation by size, and extrapolation by number of executors.

Extrapolation by size

Wordcount

Figure 11 presents the Wordcount workload results using the extrapolation approach. In this case, the extrapolation by size yielded the best results among analytical and ML models. The model accuracy is measured by finding the accuracy of the models. The comparative results are presented in Table 10, and it can be noted that Eq. (1) shows the best fit of the data compared to other models. By using the ML algorithm, the data fitting accuracy decreases. Our analysis concludes that the performance of the proposed equations are better than ML when the data are extrapolated.

SVM

For SVM workloads, the extrapolation by size found the best results using Eq. (5), where the model accuracy is 0.981, but the KRR and GBR accuracy is lower than that of all the analytical equations. We used 3D charts to illustrate the relationship between size, number of executors and runtime. We chose the 3D charts because it is easier to show their relationship more accurately. We select the best analytical and ML results that are plotted in Fig. 12. In this case, we found that KRR shows better performance as compared to the GBR algorithm.

Pagerank

In the case of Pagerank workload, analytical models completely outperform KRR and GBR. Equations (5) and (6) have low accuracy and prove the models' effectiveness over ML when the data is extrapolated by size. The maximum accuracy of the analytical models is 0.996, whereas ML is 0.875. The influence of the extrapolation is less effective for KRR and GBR. For the comparison, Fig. 13 shows the fitting for Eq. (5) and for GBR.

Kmeans

In the case of the Kmeans workload, when the data is extrapolated by size, Eq. (6) and ERNEST show an equally excellent performance, and other analytical models demonstrate better performance over ML models. As shown in Table 10, Eq. (6) achieved notable accuracy at 0.998, but KRR and GBR accuracy proved to be relatively poor at 0.836 and 0.875 respectively. We can conclude from these results that Eq. (6) is a very effective model on the unseen data points. For comparison, Fig. 14 shows the fitting for Eq. (6) and for GBR.

Graph

In the case of Graph workload when data is extrapolated by size, Eq. (4) is either at par or higher among the analytical models, and both ML algorithms appear to be less effective. We found that equation 4 and KRR performance is better than other models where the Eq. (4) and KRR accuracy is 0.940 and 0.904, respectively. For comparison, Fig. 15 shows the fitting for Eq. (4) and for KRR.

Extrapolation by number of executors

Wordcount

The results of extrapolation by a number of executors based on Wordcount workload are presented in Table 11. As expected, ML performance is considerably lower than that of the analytical model. The performances of the Eqs. (4), (6), and (1) (Amdahl) are equal or the same as the other models. The best accuracy achieved by both the equations is 0.997, whereas the accuracies for ML algorithms KRR and GBR are 0.786 and 0.535, respectively. We compared all the results and plotted the best results in Fig. 16 and (6). These results demonstrate that the extrapolation by a number of executors yielded the best results using equations while the results for ML algorithms were less accurate.

SVM

In the case of SVM, the extrapolation by the number of executors yielded the best results using Eq. (4). The KRR and GBR regression performance is poor as expected. The effectiveness of Eq. (4) is excellent, where the accuracy is 0.893. The best comparative performance in Fig. 17 shows the fitting for Eq. (4) and for KRR. These results demonstrate that the extrapolation by a number of executors yielded the best results using equations while the results were less accurate for the ML algorithms..

Pagerank

Equations (4), (5), (6) and Amdahl (1) show remarkable performance improvement for the Pagerank workload where the number of executors extrapolates the data. All four equations obtained the accuracy of 0.994 while KRR and GBR regression showed relatively demonstrated a poor accuracy which was 0.619 and 0.408 respectively. We examined (5) and compared the models' performance and plotted the best performance in Fig. 18.

Kmeans

In the case of the Kmeans workload, Eqs. (5) and (6) produce better fit than all the models and show a significant performance improvement when the data is extrapolated considering the number of executors. As shown in Table 11 both equations performed

Table 6 Kernel ridge regression algorithm statistical parameters using set of different workloads

Workload	Degree	Alpha	R-squared	RRSE
WordCount	70	1	0.999 ± 0.000	0.097 ± 0.000
	3 (default)		0.935 ± 0.002	2.039 ± 0.046
SVM	70	1	0.999 ± 0.000	0.083 ± 0.003
	3 (default)		0.860 ± 0.001	2.434 ± 0.010
Pagerank	70	1	0.999 ± 0.000	6.507 ± 0.025
	3 (default)		0.932 ± 0.001	53.885 ± 0.376
Kmeans	30	1	0.999 ± 0.000	7.868 ± 0.290
	3 (default)		0.947 ± 0.008	119.551 ± 8.875
Graph (NWeight)	30	1	0.996 ± 0.000	0.239 ± 0.010
	3 (default)		0.900 ± 0.015	1.242 ± 0.0411

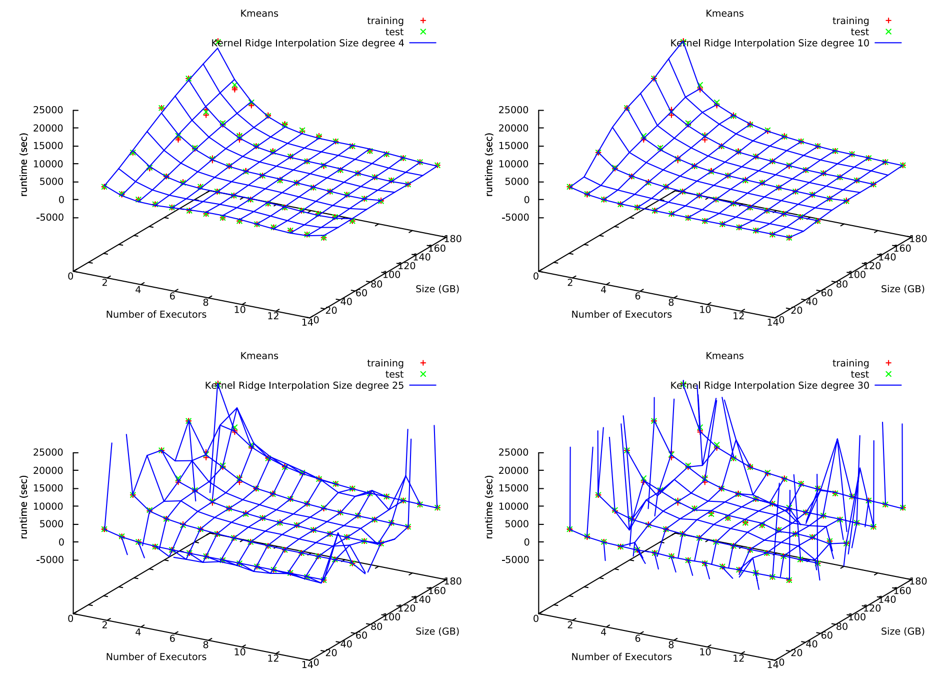

Fig. 5 Kernel ridge regression models for Kmeans with increasing degrees. Despite better R-squared, interpolation is very inaccurate for higher degrees

Table 7 GBR algorithm statistical parameters using set of different workloads

Workload	Random state	R-squared	RRSE
WordCount	1	0.998 ± 0.000	0.348 ± 0.018
SVM	1	0.994 ± 0.001	0.465 ± 0.075
Pagerank	1	0.999 ± 0.000	6.132 ± 0.550
Kmeans	1	0.997 ± 0.000	28.086 ± 3.760
Graph (NWeight)	1	0.986 ± 0.003	0.452 ± 0.015

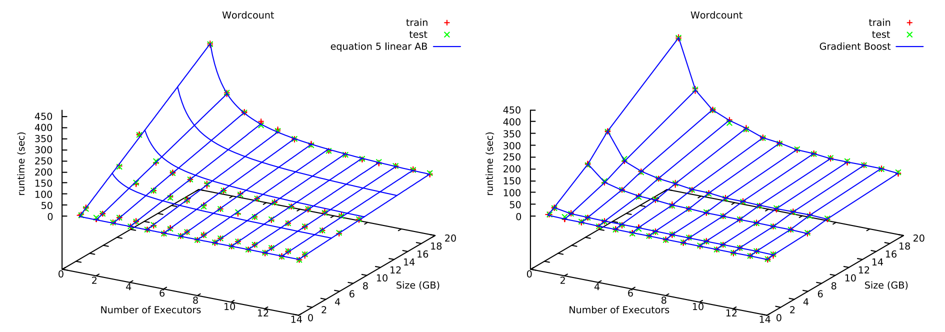

Fig. 6 Comparison of ML (GBR) and analytical models Eq. (5) showing best R-Squared for Wordcount

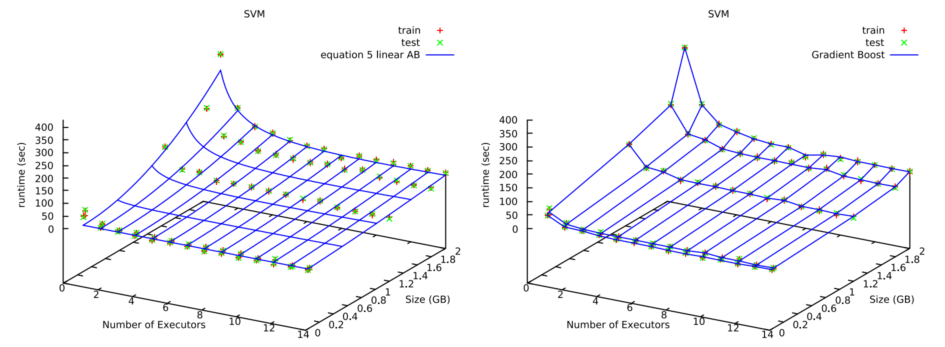

Fig. 7 Comparison of ML (GBR) and analytical models Eq. (5) showing best R-Squared for SVM

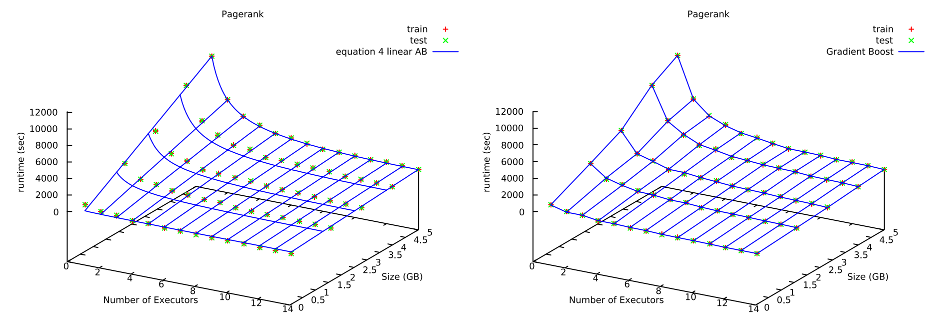

Fig. 8 Comparison of ML (GBR) and analytical models Eq. (4) showing best R-Squared for Pagerank

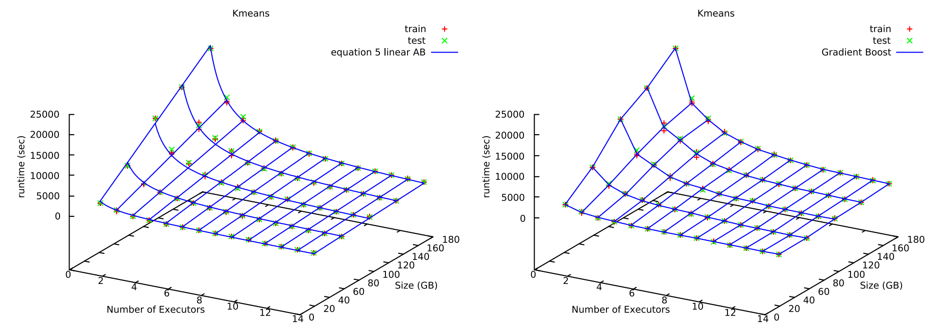

Fig. 9 Comparison of ML (GBR) and analytical models Eq. (5) showing best R-Squared for Kmeans

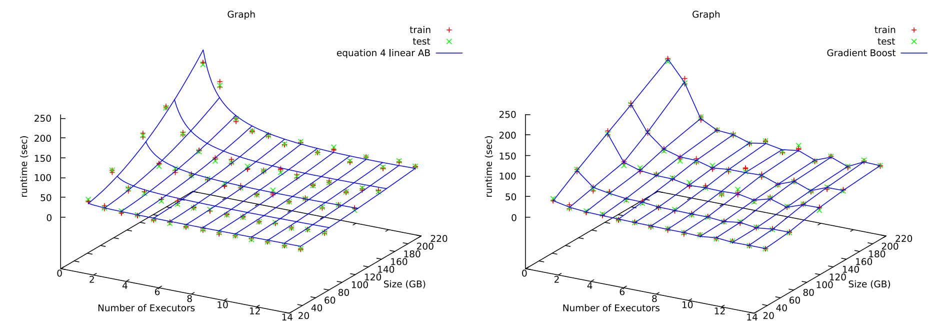

Fig. 10 Comparison of ML (GBR) and analytical models Eq. (4) showing best R-Squared for Graph

Table 8 R-squared values for a different set of workloads and models

Workload *f*(*S*)	Wordcount linear	SVM quadratic	Pagerank linear	Kmeans linear	Graph (NWeight) quadratic
Amdhal equation (1)	0.995 ± 0.000	0.908 ± 0.005	0.990 ± 0.000	0.992 ± 0.002	0.901 ± 0.012
Gustafson equation (2)	0.995 ± 0.000	0.888 ± 0.002	0.988 ± 0.000	0.992 ± 0.000	0.898 ± 0.013
ERNEST equation (3)	0.994 ± 0.000	0.848 ± 0.001	0.987 ± 0.000	0.992 ± 0.002	0.916 ± 0.003
2D plate equation (4)	0.995 ± 0.000	0.916 ± 0.005	0.990 ± 0.000	0.992 ± 0.002	0.918 ± 0.009
Connected graph equation (5)	0.995 ± 0.001	0.918 ± 0.005	0.989 ± 0.000	0.992 ± 0.002	0.911 ± 0.005
Con. graph $c = 1$ equation (6)	0.995 ± 0.001	0.914 ± 0.005	0.989 ± 0.000	0.992 ± 0.002	0.911 ± 0.005
Kernel ridge regression	0.974 ± 0.002	0.934 ± 0.001	0.977 ± 0.000	0.981 ± 0.005	0.945 ± 0.009
Gradient boost regression	**0.998 ± 0.000**	**0.995 ± 0.001**	**0.999 ± 0.000**	**0.997 ± 0.001**	**0.986 ± 0.003**

The bold data in each column indicates the largest R-squared value in the corresponding column

Table 9 Relative mean standard error (RRSE) values for a different set of workloads and models

Workload *f*(*S*)	Wordcount linear	SVM quadratic	Pagerank linear	Kmeans linear	Graph (NWeight) quadratic
Amdhal equation (1)	0.085 ± 0.005	0.282 ± 0.009	0.113 ± 0.000	0.140 ± 0.019	0.252 ± 0.009
Gustafson equation (2)	0.091 ± 0.004	0.311 ± 0.003	0.120 ± 0.000	0.134 ± 0.009	0.254 ± 0.008
ERNEST equation (3)	0.100 ± 0.006	0.366 ± 0.001	0.127 ± 0.000	0.142 ± 0.017	0.231 ± 0.009
2D plate equation (4)	0.086 ± 0.005	0.272 ± 0.009	0.113 ± 0.000	0.141 ± 0.019	0.226 ± 0.007
Connected graph equation (5)	0.091 ± 0.009	0.268 ± 0.009	0.116 ± 0.000	0.141 ± 0.018	0.244 ± 0.008
Con. graph $c = 1$ equation (6)	0.091 ± 0.009	0.273 ± 0.008	0.118 ± 0.000	0.140 ± 0.018	0.243 ± 0.008
Kernel ridge regression	0.203 ± 0.009	0.234 ± 0.002	0.173 ± 0.001	0.207 ± 0.024	0.178 ± 0.003
Gradient boost regression	**0.058 ± 0.003**	**0.064 ± 0.010**	**0.033 ± 0.002**	**0.062 ± 0.038**	**0.087 ± 0.004**

The bold data in each column indicates the smallest RRSE value in the corresponding column

Table 10 R-squared values for extrapolation on size

Workload *f*(*S*)	Wordcount linear	SVM quadratic	Pagerank linear	Kmeans linear	Graph (NWeight) quadratic
Amdhal equation (1)	**0.998 ± 0.000**	0.965 ± 0.001	0.994 ± 0.000	0.997 ± 0.001	0.937 ± 0.006
Gustafson equation (2)	0.996 ± 0.001	0.949 ± 0.004	0.994 ± 0.000	0.996 ± 0.001	0.913 ± 0.008
ERNEST equation (3)	0.996 ± 0.001	0.958 ± 0.002	0.990 ± 0.000	0.998 ± 0.001	0.921 ± 0.008
2D plate equation (4)	0.997 ± 0.001	0.951 ± 0.003	0.993 ± 0.000	0.997 ± 0.001	**0.940 ± 0.005**
Connected graph equation (5)	0.257 ± 0.061	**0.981 ± 0.001**	**0.996 ± 0.000**	0.996 ± 0.001	0.940 ± 0.006
Con. graph $c = 1$ equation (6)	0.997 ± 0.001	0.978 ± 0.001	**0.996 ± 0.000**	**0.998 ± 0.001**	0.940 ± 0.006
Kernel ridge regression	0.836 ± 0.011	0.745 ± 0.004	0.836 ± 0.011	0.836 ± 0.011	0.904 ± 0.043
Gradient boost regression	0.875 ± 0.005	0.690 ± 0.003	0.875 ± 0.005	0.875 ± 0.005	0.775 ± 0.009

The bold data in each column indicates the largest R-squared values in the corresponding column

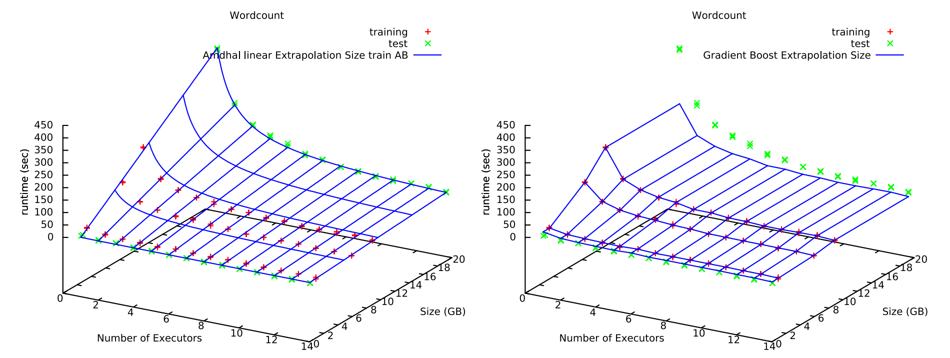

Fig. 11 Extrapolation relationship showing Wordcount workload by size using Eq. (1) and GBR

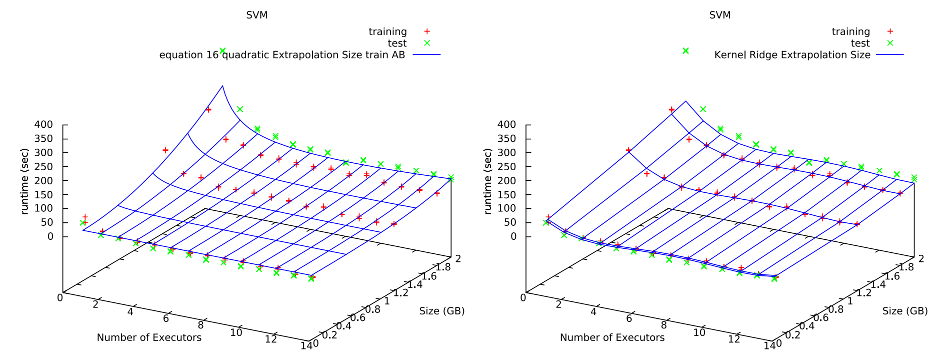

Fig. 12 Extrapolation relationship showing SVM workload by Size using Eq. (5) and KRR

well where the equation achieved the accuracy of 0.995. The KRR and GBR achieved poor accuracy of 0.917 and 0.510 respectively. These results indicate that the ML performance is less accurate when training on limited data and when having to extrapolate

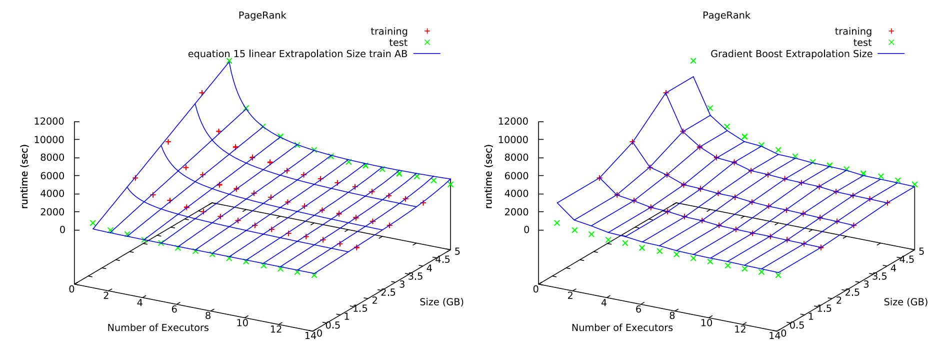

Fig. 13 Extrapolation relationship showing Pagerank workload by Size using Eq. (5) and GBR

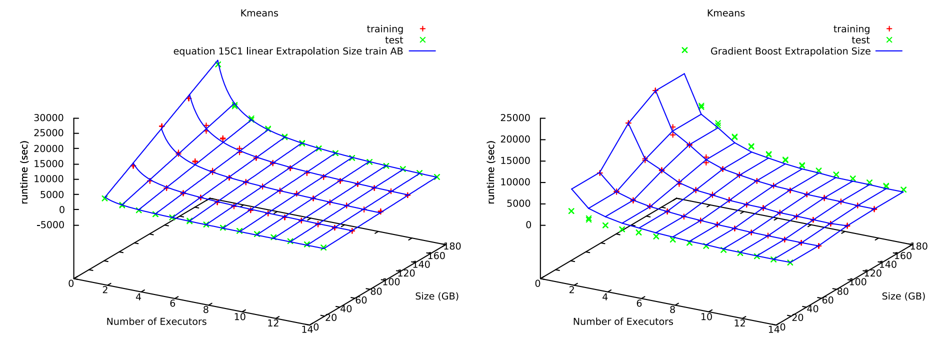

Fig. 14 Extrapolation relationship showing Kmeans workload by Size using Eq. (6) and GBR

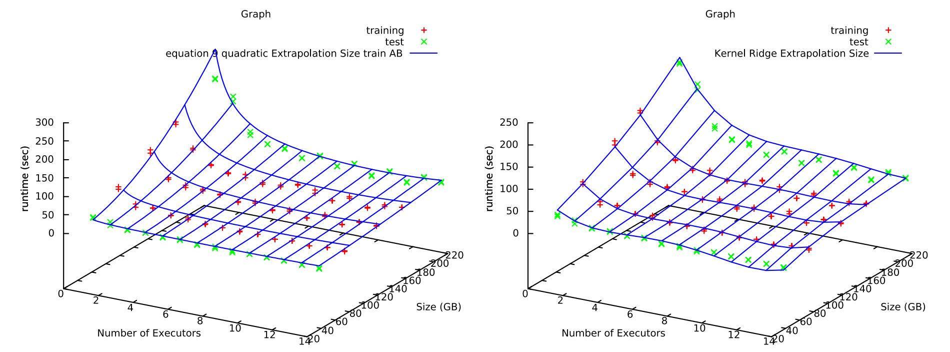

Fig. 15 Extrapolation relationship showing Graph workload by Size using Eq. (4) and Kernel Ridge

results beyond the values seen in the training data. We examined and compared the performance of the models and plotted the best performance in Fig. 19 and (6).

Graph

In the case of the Graph workload, the data were extrapolated by size. Equation (4) shows the best performance among all the models. In this case, the KRR and GBR performances were inferior. As shown in Table 11, the Eq. (4) accuracy is 0.945, while

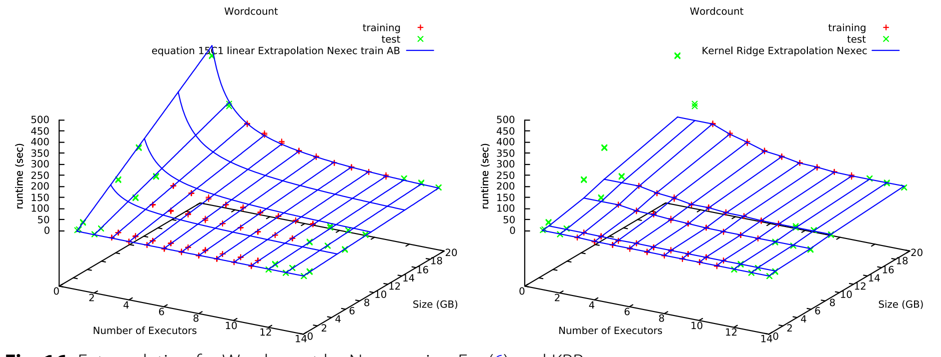

Fig. 16 Extrapolation for Wordcount by Nexec using Eq. (6) and KRR

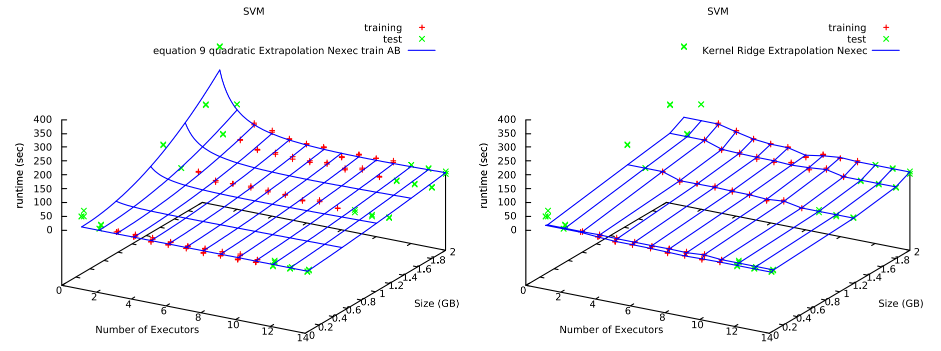

Fig. 17 Extrapolation for SVM by Nexec using Eq. (4) and KRR

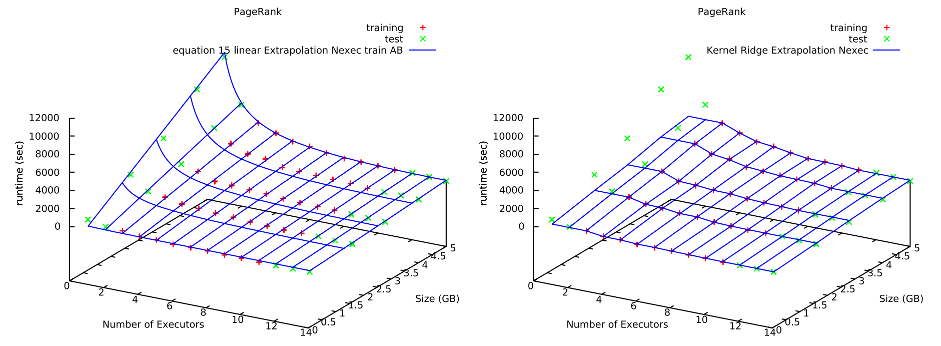

Fig. 18 Extrapolation for Pagerank by Nexec using Eq. (5) and KRR

the KRR and GBR accuracies are 0.150 and 0.371 respectively. These results indicate that the selected ML algorithms are not suitable when the extrapolation is required. Fig. 20 shows this performance comparison results for Eq. (4) and GBR.

Discussion

To evaluate the performance of the prediction models, this paper used the analytical and ML models and depicted the comparative analysis between them. To predict Spark runtime performance, several experiments have been performed to evaluate the

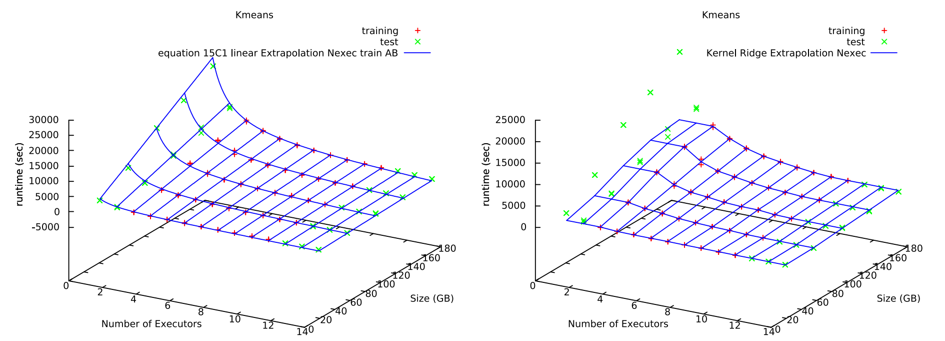

Fig. 19 Extrapolation for Kmeans by Nexec using Eq. (6) and KRR

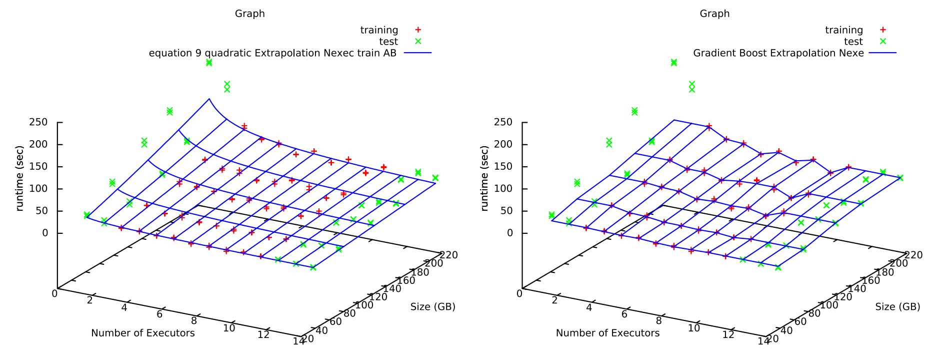

Fig. 20 Extrapolation for Graph by Nexec using Eq. (4) and GBR

Table 11 R-squared values for Extrapolation on Number of Executors

Workload *f*(*S*)	Wordcount linear	SVM quadratic	Pagerank linear	Kmeans linear	Graph (NWeight) quadratic
Amdhal equation (1)	**0.997 ± 0.000**	0.878 ± 0.002	**0.994 ± 0.000**	0.994 ± 0.001	0.932 ± 0.001
Gustafson equation (2)	0.995 ± 0.000	0.728 ± 0.000	0.988 ± 0.000	0.921 ± 0.001	0.922 ± 0.000
ERNEST equation (3)	0.996 ± 0.000	0.822 ± 0.002	0.991 ± 0.000	0.992 ± 0.001	0.930 ± 0.007
2D plate equation (4)	**0.997 ± 0.000**	**0.893 ± 0.004**	**0.994 ± 0.000**	0.992 ± 0.002	**0.945 ± 0.004**
Connected graph equation (5)	0.996 ± 0.000	0.853 ± 0.072	**0.994 ± 0.000**	**0.995 ± 0.001**	0.917 ± 0.002
Con. graph $c = 1$ equation (6)	**0.997 ± 0.000**	0.850 ± 0.072	**0.994 ± 0.000**	**0.995 ± 0.001**	0.916 ± 0.002
Kernel ridge regression	0.786 ± 0.012	0.619 ± 0.012	0.917 ± 0.014	0.917 ± 0.014	0.150 ± 0.060
Gradient boost regression	0.535 ± 0.001	0.408 ± 0.007	0.510 ± 0.011	0.510 ± 0.011	0.371 ± 0.015

The bold data in each column indicates the largest R-squared values in the corresponding column

performance. The comprehensive comparative study of the results presented in Tables 6, 7, 8, 9, 10, and 11 inspires us to the following three steps analysis.

Firstly, we present the KRR regression parameter relationship between alpha and degree. This study found, for most of the workloads, the best R-squared can be achieved by selecting the small degree with alpha. Our analysis found higher degree can produce best R-squared but the data overfitting can be a major limitation. For the GBR, we kept all the parameters default. However, we have examined different random states, but there are no effects on the accuracy improvement. The detailed results are presented in "Performance evaluations and analysis" and "Performance analysis using extrapolation" sections.

Secondly, interpolation experiments were carried out. The data split for the cross-validation used data points for all the available sizes and number of executors for both test and training sets. The presented results showed that analytical models are better than KRR regression and produce similar accuracy as ERNEST, Amdahl or Gustafson. However, the GBR model was the most accurate when compared to all other models.

Finally, we used the extrapolation method with the cross-validation technique, and the analysis was carried out using size and executors. We noticed the performance of ML models are poor in both cases, but the analytical models are more accurate and effective. The presented results in Tables 10 and 11 showed that the linear workloads are more accurate among the ML models than the quadratic workloads; in fact, the accuracy is significantly poorer. The KRR, GBR, 2D-plate (Eq. 4) or fully-connected (Eq. 5) models average accuracies are 0.466, 0.677 and 0.950. These results indicate that both 2D-plate model and the fully-connected models are more effective and accurate when using extrapolation of data, either over size or number of executors.

Conclusion

This work aimed to compare five analytical models against ML regression algorithms for Spark performance prediction. We investigated two ML algorithms, namely KRR and GBR algorithms, and five analytical models, namely, 2D-plate, fully-connected, ERNEST, Amdahl, Gustafson models. The key challenges were how to use limited data points for generating models that fit the data accurately and generalise well when extrapolating.

To address these challenges, we used interpolation and extrapolation methods with k-fold cross-validation technique for both ML and analytical models. Using the interpolation method, 2D-plate and fully-connected models outperformed the KRR algorithm, ERNEST, Amdahl and Gustafson, but the GBR showed a better fitting accuracy (R^2) than all other models.

Due to the limited available input data when using the extrapolation method, ML algorithms proved not to be as accurate as 2D-plate, and fully-connected models. Our experimental findings confirm that both 2D-plate and fully-connected models reduce the percentage error significantly and can accurately fit the data for prediction purposes. For this reason, 2D-plate and fully-connected models stand out as very effective approaches in the presence of limited input data for predicting Spark performance as well as parallel system performance.

For future work, we plan to study different ML algorithms for comparative analysis as well as perform more robust experimentations with different HiBench workloads in order to yield further conclusive findings.

Abbreviations
SVM: Support vector machines; SVR: Support vector regression; GBR: Gradient boost regression; GBM: Gradient boost machine; DT: Decision tree; LR: Logistic regression; KRR: Kernel ridge regression; MF: Matrix factorization; NB: Naıve Bayes; NN: Neural networks; MLR: Multi linear regression; TSt: Two-stage tree; LS-SVM: Least square support vector machine; API: Application programming interface; SQL: Structured query language; HDFS: Hadoop distributed file system; RDD: Resilient distributed datasets; ML: Machine learning; MLlib: Machine learning library; AMPLab: Algorithms, machines and people lab; CPU: Central processing unit; I/O: Input/output; UC: University of California; AMP: Algorithms, machines and people; DAG: Directed acyclic graph; YARN: Yet another resource negotiator; NEXEC: Number of executor; SQRT: Square root; RRSE: Root relative squared error; 2D: Two dimensional; GHz: Gigahertz; TB: Terabyte; RAM: Random access memory; DDR: Double data rate; GB: Gigabyte; MB: Megabyte; WC: Wordcount; Exec: Executor; Amazon EC2: Amazon elastic computing.

Acknowledgements
This work was supported in part by the Massey University Doctoral Scholarship.

Author contributions
NA and ALCB were the main contributors of this work. NA has done an initial literature review and data collection, run experiments, prepare results, and drafted the manuscript. Both NA and ALCB fitted the data into the models and analysed the results. ALCB and TS deployed and configured the physical Hadoop cluster. TS and MAR helped to improve the final paper. All authors read and approved the final manuscript.

Funding
Not applicable. This research received no fund from any agency.

Availability of data and materials
Data are contained within the article. However, the correspondence author can be contacted for more details.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]School of Mathematical and Computational Sciences, Massey University, Auckland 0745, New Zealand. [2]Department of Mechanical and Electrical Engineering, Massey University, Auckland 0745, New Zealand.

Received: 22 December 2021 Accepted: 2 May 2022
Published online: 19 May 2022

References
1. Ghani NA, Hamid S, Hashem IAT, Ahmed E. Social media big data analytics: a survey. Comput Hum Behav. 2019;101:417–28.
2. Fang R, Pouyanfar S, Yang Y, Chen S-C, Iyengar S. Computational health informatics in the big data age: a survey. ACM Comput Surv. 2016;49(1):1–36.
3. Hirschberg J, Manning CD. Advances in natural language processing. Science. 2015;349(6245):261–6.
4. Maros A, Murai F, da Silva APC, Almeida JM, Lattuada M, Gianniti E, Hosseini M, Ardagna D. Machine learning for performance prediction of spark cloud applications. In: 2019 IEEE 12th international conference on cloud computing (CLOUD). New York: IEEE; 2019. p. 99–106.
5. Salloum S, Dautov R, Chen X, Peng PX, Huang JZ. Big data analytics on apache spark. Int J Data Sci Anal. 2016;1(3):145–64.
6. Awan MJ, Khan RA, Nobanee H, Yasin A, Anwar SM, Naseem U, Singh VP. A recommendation engine for predicting movie ratings using a big data approach. Electronics. 2021;10(10):1215.
7. Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, et al. Mllib: machine learning in apache spark. J Mach Learn Res. 2016;17(1):1235–41.
8. Petridis P, Gounaris A, Torres J. Spark parameter tuning via trial-and-error. In: INNS conference on big data. Berlin: Springer; 2016. p. 226–37.
9. Ahmed N, Barczak AL, Susnjak T, Rashid MA. A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. J Big Data. 2020;7(1):1–18.
10. Herodotou H, Babu S. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. Proc VLDB Endow. 2011;4(11):1111–22.
11. Mustafa S, Elghandour I, Ismail MA. A machine learning approach for predicting execution time of spark jobs. Alex Eng J. 2018;57(4):3767–78.
12. Cheng G, Ying S, Wang B, Li Y. Efficient performance prediction for apache spark. J Parallel Distrib Comput. 2021;149:40–51.

13. Cheng G, Ying S, Wang B. Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model. J Syst Softw. 2021;180:111028.
14. Luo N, Yu Z, Bei Z, Xu C, Jiang C, Lin L. Performance modeling for spark using svm. In: 2016 7th international conference on cloud computing and big data (CCBD). New York: IEEE; 2016. p. 127–31.
15. Ahmed N, Barczak AL, Rashid MA, Susnjak T. An enhanced parallelisation model for performance prediction of apache spark on a multinode hadoop cluster. Big Data Cogn Comput. 2021;5(4):65.
16. Ahmed N, Barczak ALC, Susnjak T, Rashid MA. A parallelization model for performance characterization of spark big data jobs on Hadoop clusters. J Big Data. 2021;8(107):1–28. https://doi.org/10.1186/s40537-021-00499-7.
17. Dogan A, Birant D. Machine learning and data mining in manufacturing. Expert Syst Appl. 2021;166:114060.
18. Mavridis I, Karatza H. Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark. J Syst Softw. 2017;125:133–51.
19. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauly M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: 9th {USENIX} symposium on networked systems design and implementation ({NSDI} 12), 2012. p. 15–28.
20. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I, et al. Spark: cluster computing with working sets. HotCloud. 2010;10(10–10):95.
21. Shahul A. Spark architecture: Apache Spark tutorial. 2021. https://www.learntospark.com/2020/02/spark-architecture.html.
22. Chen J, Li K, Tang Z, Bilal K, Yu S, Weng C, Li K. A parallel random forest algorithm for big data in a spark cloud computing environment. IEEE Trans Parallel Distrib Syst. 2016;28(4):919–33.
23. Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S et al. Apache hadoop yarn: yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing, 2013. p. 1–16.
24. Ousterhout K, Rasti R, Ratnasamy S, Shenker S, Chun B-G. Making sense of performance in data analytics frameworks. In: 12th {USENIX} symposium on networked systems design and implementation ({NSDI} 15), 2015. p. 293–307.
25. Al-Sayeh H, Hagedorn S, Sattler K-U. A gray-box modeling methodology for runtime prediction of apache spark jobs. Distrib Parallel Databases. 2020;38(4):819–39.
26. Chao Z, Shi S, Gao H, Luo J, Wang H. A gray-box performance model for apache spark. Future Gener Comput Syst. 2018;89:58–67.
27. Lattuada M, Gianniti E, Hosseini M, Ardagna D, Alexandre M, Fabricio M, COUTO da SILVA AP, Jussara MA. Gray-box models for performance assessment of spark applications. In: 9th international conference on cloud computing and services science, SciTePress; 2019. p. 609–18.
28. Prats DB, Portella FA, Costa CH, Berral JL. You only run once: spark auto-tuning from a single run. IEEE Trans Netw Serv Manag. 2020;17(4):2039–51.
29. Jia Z, Xue C, Chen G, Zhan J, Zhang L, Lin Y, Hofstee P. Auto-tuning spark big data workloads on power8: Prediction-based dynamic smt threading. In: 2016 international conference on parallel architecture and compilation techniques (pact), New York: IEEE; 2016. p. 387–400.
30. Nikitopoulou D, Masouros D, Xydis S, Soudris D. Performance analysis and auto-tuning for spark in-memory analytics. In: 2021 design, automation & test in Europe conference & exhibition (DATE). New York: IEEE; 2021. p. 76–81.
31. de Oliveira D, Porto F, Boeres C, de Oliveira D. Towards optimizing the execution of spark scientific workflows using machine learning-based parameter tuning. Concurr Comput Pract Exp. 2021;33(5):5972.
32. Boden C, Spina A, Rabl T, Markl V. Benchmarking data flow systems for scalable machine learning. In: Proceedings of the 4th ACM SIGMOD workshop on algorithms and systems for mapreduce and beyond. 2017. p. 1–10.
33. Boden C, Rabl T, Schelter S, Markl V. Benchmarking distributed data processing systems for machine learning workloads. In: Technology conference on performance evaluation and benchmarking. Berlin: Springer; 2018. p. 42–57.
34. Mostafaeipour A, Jahangard Rafsanjani A, Ahmadi M, Arockia Dhanraj J. Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. J Supercomput. 2021;77(2):1273–300.
35. Assefi M, Behravesh E, Liu G, Tafti AP. Big data machine learning using apache spark MLlib. In: 2017 IEEE international conference on big data (big Data). New York: IEEE; 2017. p. 3492–8.
36. Javaid MU, Kanoun AA, Demesmaeker F, Ghrab A, Skhiri S. A performance prediction model for spark applications. In: International conference on big data. Berlin: Springer; 2020. p. 13–22.
37. Singhal R, Phalak C, Singh P. Spark job performance analysis and prediction tool. In: Companion of the 2018 ACM/SPEC international conference on performance engineering. 2018. p. 49–50.
38. Yigitbasi N, Willke TL, Liao G, Epema D. Towards machine learning-based auto-tuning of mapreduce. In: 2013 IEEE 21st international symposium on modelling, analysis and simulation of computer and telecommunication systems. New York: IEEE. 2013. p. 11–20.
39. Cristianini N, Shawe-Taylor J, et al. An introduction to support vector machines and other kernel-based learning methods. Cambridge: Cambridge University Press; 2000.
40. Friedman JH. Stochastic gradient boosting. Comput Stat Data Anal. 2002;38(4):367–78.
41. Venkataraman S, Yang Z, Franklin M, Recht B, Stoica I. Ernest: Efficient performance prediction for large-scale advanced analytics. In: 13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16), 2016. p. 363–78.
42. Amdahl GM. Validity of the single processor approach to achieving large scale computing capabilities. In: AFIPS '67 (spring): spring joint computer conference. 1967. p. 483–5. https://doi.org/10.1145/1465482.1465560.
43. Gustafson JL. Reevaluating Amdahl's law. Commun ACM. 1988;31(5):532–3. https://doi.org/10.1145/42411.42415.
44. Boden C, Rabl T, Markl V. Distributed machine learning-but at what cost. In: Machine learning systems workshop at the 2017 conference on neural information processing systems. 2017.
45. Cawley GC, Talbot NL, Chapelle O. Estimating predictive variances with kernel ridge regression. In: Machine learning challenges workshop. Berlin: Springer; 2005. p. 56–77.
46. Kernel Ridge Regression. https://scikit-learn.org/stable/modules/generated/sklearn.kernel_ridge.K ernelRidge.html.

47. Ashcroft M. Advanced machine learning: basics and kernel regression. https://www.futurelearn.com/info/courses/advanced-machine-learning/0/st eps/49560.
48. sklearn.ensemble.GradientBoostingRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.
49. Wilkinson B, Allen M. Parallel programming. New Jersey: Prentice Hall; 1999.
50. Barczak ALC, Messom CH, Johnson MJ. Performance characteristics of a cost-effective medium-sized Beowulf cluster supercomputer. In: LNCS 2660. (2003). SpringerLink; 2003. p. 1050–9.
51. Vazquez C, Krishnan R, John E. Cloud computing benchmarking: a survey. In: Proceedings of the international conference on grid, cloud, and cluster computing (GCC), 2014. p. 1.
52. Sobel W, Subramanyam S, Sucharitakul A, Nguyen J, Wong H, Klepchukov A, Patil S, Fox A, Patterson D. Cloudstone: multi-platform, multi-language benchmark and measurement tools for web 2.0. In: Proc. of CCA, Vol. 8. 2008. p. 228.
53. Intel-bigdata: HiBench benchmark suit. https://github.com/Intel-bigdata/HiBench.
54. Han R, John LK, Zhan J. Benchmarking big data systems: a review. IEEE Trans Serv Comput. 2017;11(3):580–97.
55. Zhao Y, Hu F, Chen H. An adaptive tuning strategy on spark based on in-memory computation characteristics. In: 2016 18th international conference on advanced communication technology (ICACT). New York: IEEE; 2016. p. 484–8.
56. Marcu O-C, Costan A, Antoniu G, Pérez-Hernández MS. Spark versus flink: understanding performance in big data analytics frameworks. In: 2016 IEEE international conference on cluster computing (CLUSTER). New York: IEEE; 2016. p. 433–42.
57. NIST/SEMATECH e-handbook of statistical methods. National Institute of Standards and Technology (NIST). 2018. https://www.itl.nist.gov/div898/handbook/pmd/section8/pmd811.htm
58. Ding Y, Pervaiz A, Carbin M, Hoffmann H. Generalizable and interpretable learning for configuration extrapolation. In: Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering. 2021. p. 728–40.

Publisher's Note