Future Generation Computer Systems 87 (2018) 580–590

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A multi-layered performance analysis for cloud-based topic detection and tracking in Big Data applications

Meisong Wang [a], Prem Prakash Jayaraman [b], Ellis Solaiman [c,*], Lydia Y. Chen [d], Zheng Li [e], Song Jun [f], Dimitrios Georgakopoulos [b], Rajiv Ranjan [f,c]

[a] *School of Computer Science, Australian National University, ACT, Australia*
[b] *Faculty of Science, Engineering and Technology, Swinburne University of Technology, Melbourne, Australia*
[c] *School of Computer Science, Newcastle University, Newcastle Upon Tyne, UK*
[d] *Zurich Research Laboratory, IBM, Zurich, Switzerland*
[e] *Department of Electrical and Information Technology, Lund University, Sweden*
[f] *Department of Computer Science, Chinese University of Geosciences, Wuhan, China*

HIGHLIGHTS

- A novel end-to-end performance analysis that identifies key metrics impacting CTDT applications.
- A novel analysis that captures dependencies between metrics across the cloud layers.
- Identified metrics are validated via real-world data sets obtained from Twitter.

ARTICLE INFO

Article history:
Received 19 July 2017
Received in revised form 21 November 2017
Accepted 22 January 2018
Available online 3 March 2018

Keywords:
Cloud-based TDT
Big Data
Performance analysis
Cloud computing

ABSTRACT

In the era of the Internet of Things and social media; communities, governments, and corporations are increasingly eager to exploit new technological innovations in order to track and keep up to date with important new events. Examples of such events include the news, health related incidents, and other major occurrences such as earthquakes and landslides. This area of research commonly referred to as Topic Detection and Tracking (TDT) is proving to be an important component of the current generation of Internet-based applications, where it is of critical importance to have early detection and timely response to important incidents such as those mentioned above. The advent of Big data though beneficial to TDT applications also brings about the enormous challenge of dealing with data variety, velocity and volume (3Vs). A promising solution is to employ Cloud Computing, which enables users to access powerful and scalable computational and storage resources in a "pay-as-you-go" fashion. However, the efficient use of Cloud resources to boost the performance of mission critical applications employing TDT is still an open topic that has not been fully and effectively investigated. An important prerequisite is to build a performance analysis capable of capturing and explaining specific factors (for example; CPU, Memory, I/O, Network, Cloud Platform Service, and Workload) that influence the performances of TDT applications in the cloud. Within this paper, our main contribution, is that we present a multi-layered performance analysis for big data TDT applications deployed in a cloud environment. Our analysis captures factors that have an important effect on the performance of TDT applications. The novelty of our work is that it is a first kind of vertical analysis on infrastructure, platform and software layers. We identify key parameters and metrics in each cloud layer (including Infrastructure, Software, and Platform layers), and establish the dependencies between these metrics across the layers. We demonstrate the effectiveness of the proposed analysis via experimental evaluations using real-world datasets obtained from Twitter.

* Corresponding author.
E-mail addresses: deanmeisong@gmail.com (M. Wang), pjayaraman@swin.edu.au (P.P. Jayaraman), ellis.solaiman@ncl.ac.uk (E. Solaiman), yic@zurich.ibm.com (L.Y. Chen), zheng.li@eit.lth.se (Z. Li), songjun@cug.edu.cn (S. Jun), dgeorgakopoulos@swin.edu.au (D. Georgakopoulos), rranjans@gmail.com (R. Ranjan).

https://doi.org/10.1016/j.future.2018.01.047

1. Introduction

The advent of Big Data applications that are fuelled by numerous data sources such as social media and the Internet of Things, has created new opportunities for individuals, communities, governments, and corporations to make use of this new and potentially important data that is continuously being generated. This area of research commonly referred to as Topic Detection and Tracking (TDT) is becoming a critical component of the current generation of Internet-based applications. An example where TDT research is of critical importance is in developing the capability to provide early detection and then timely response to potential landslides using data obtained from sensors, and from social media outlets such as Twitter. The Achilles heel for TDT applications thus far has been limited access to real-time data which has an impeding effect on the accuracy of the application. The Big Data era has the potential to enhance the development of TDT applications by satisfying the requirement of acquiring large volumes of data from variety of sources at high velocity. Traditional TDT techniques are incapable of coping with Big Data challenges best characterized by the 3V features, which are Variety, Velocity, and Volume. Volume means that the amount of data is so large that traditional storage devices cannot store it (e.g. Every day, around 2.5 quintillion bytes of data is created, which means that 90% of the data in the world was created in the last two years [1]). Variety refers to the many sources and types of data, which creates problems for storing, mining and analysing the data. Velocity means being able to deal with the massive and continuous speed at which data flows from sources like sensors, social media, and various networks to the cloud to be processed and stored.

Recently, cloud computing techniques have emerged as reliable, effective and practicable means for tackling the problems confronting TDT in the Big Data era. For instance, there are a number of cloud storage frameworks both commercial and free such as Amazon S3 that can be used to store large amounts of data *(Volume)*. Some NO-SQL databases can be used to store, process and analyse various types of data *(Variety)*. In addition, parallel computing frameworks such as Apache Spark can be effectively used to significantly enhance the speed of processing Big Data, and even to meet real-time analysis requirements, which consequently addresses the *"Velocity"* problem. Another benefit that Cloud computing can offer is the scalability that can satisfy the requirement of processing data which is rapidly increasing in volume.

1.1. Motivation and research problem

Although Cloud computing creates clear advantages for TDT applications (for processing and analysing Big Data) such as those identified above, it also generates new challenges, and one of the most important challenges is how to optimize the cloud resources to support mission critical TDT applications. An important first step is to study and analyse the performance of cloud-based TDT (CTDT) applications. Developing analysis capabilities that can capture the performance of CTDT applications is not a trivial task given (1) the multi-layered nature of cloud computing (IaaS, SaaS, and PaaS), (2) different metrics required to capture the performance of TDT applications when compared with other cloud-based applications such as e-commerce and customer relationship management systems, and (3) dependencies between each of the metrics across cloud layers. Existing TDT analysis techniques [2–4], capture the performance of processing and analysing Big Data in clouds, but cannot be applied accurately to model the performance of CTDT applications due to the lack of consideration for all layers (end-to-end) that constitute a typical CTDT application (i.e. IaaS, SaaS, PaaS, etc.).

1.2. Overview of methods and contributions

In this paper, we present a first kind of vertical multi-layered (infrastructure, platform, and software) performance analysis which captures and analyses the key metrics that have an important effect on the performance of CTDT big data applications. The main contributions of this paper are:

- We clearly identify the key performance metrics that impact the performance of CTDT applications with respect to each cloud layer (i.e. IaaS, PaaS and SaaS).
- We then analyse and establish the dependencies between these metrics. The aim of the analysis is to be able to capture the performance of CTDT applications in order to be able to effectively optimize resources for such applications deployed in clouds.
- We conduct comprehensive experimental evaluations using real-world datasets obtained from Twitter to validate the effectiveness of the identified metrics and their dependencies.

The paper is organized as follows: Section 2 summarizes a comprehensive survey of existing work related to the optimization of CTDT applications, and also existing work related to performance analysis for Cloud resource optimization; Section 3 illustrates our performance analysis framework in detail; To evaluate our performance analysis, we apply it to a specific case, which is a Naïve Bayesian based CTDT application in Section 4; In Section 5 we present experimental results based on a CTDT applications that we implement; Conclusions and future directions are in Section 6.

2. Related work

Studies that are related to our work can be divided into: (1) development and implementation of cloud-based TDT applications using machine learning techniques; and (2) performance analysis for platform-as-a-service TDT applications running on clouds using frameworks such as MapReduce. As we shall see, none of these studies can be used to efficiently analyse the end-to-end performance of CTDT applications. The first class of studies mainly focuses on how to develop and implement a CTDT application using various machine learning algorithms (e.g. state vector machine, Naive Bayes etc.). However, these works lack an analysis of factors that influence the performance of the CTDT application. On the other hand, the second set of studies are heavily focused on PaaS-based approaches such as Map Reduce and lack consideration for metrics such as performance of the distributed machine learning algorithms and related dependencies across the cloud layers (IaaS, PaaS and SaaS). These factors are important, and when not considered often lead to inaccuracy of performance modelling results. This will have significant consequences on mission critical CTDT applications that are dependent on fast, scalable and accurate analysis of events. For example, consider a landslide scenario. Under normal conditions, the sensors deployed in the field monitoring the activity of the land (e.g. movement of earth) produce data at a constant rate, and data coming from social media streams is relatively less constant. However, in case of an abnormal situation, the sensor data rate and social media data increases significantly resulting in increased volume. The challenge here is that a CTDT application running in the cloud needs to be able to optimize the cloud resources to cater for such diverse situations (normal and abnormal). Failing to do so will result in mission critical applications failing to meet their goals; i.e. detecting and alerting their users to important events [5,6]. In cloud computing terminology, this is generally referred to as Quality of Service (QoS) guarantees enforced by service level agreements (SLA) [7].

Table 1 presents a summary of CTDT applications focusing on development and implementation. As described earlier, the first

Table 1
Characteristics of cloud-based TDT applications.

Study	Performance model	Performance guarantee	Performance metrics	IaaS	PaaS	SaaS
[8]	No	No	No	Yes	No	No
[9]	No	No	No	Yes	Yes	No
[10]	No	No	No	Yes	No	No
[11]	No	No	No	Yes	Yes	No
[12]	No	No	No	Yes	No	No

Table 2
Characteristics of related performance models.

Study	HDFS	Memory	ML	Task scheduler	Real environment	Simulator	Greedy algorithm	Network
[2]	Yes	Yes	No	No	Yes	No	No	Yes
[3]	Yes	No	No	Yes	No	No	No	No
[13]	No	No	No	Yes	No	Yes	Yes	No
[4]	No	No	No	Yes	No	No	Yes	No
[14]	Yes	No	No	No	Yes	No	No	No
[15]	Yes	No	No	Yes	Yes	Yes	No	No
[16]	No	No	No	Yes	No	Yes	Yes	No
[17]	Yes	Yes	No	Yes	Yes	No	No	Yes
[18]	No	No	Yes	No	No	Yes	No	No
[19]	No	No	Yes	No	No	No	No	No
[20]	Yes	No	No	No	Yes	No	No	Yes

class of CTDT applications lack performance analysis and evaluation capabilities, and provide no performance guarantees (QoS or SLA). This means that they cannot be used to develop QoS guarantees for mission critical CTDT big data applications.

A summary of platform-as-a-service CTDT applications is shown in Table 2. As stated earlier, the focus of this related work is to develop a performance model for map-reduce or similar distributed framework-based TDT applications. We compare these approaches by using the taxonomy presented below:

(1) HDFS: Are factors of HDFS taken into consideration?
(2) Memory: Whether this work considers effects of memory.
(3) Task Scheduler: Whether this work consists of scheduling mechanisms of MapReduce tasks.
(4) Real Environment: Whether this work is based on a real environment or another approach such as simulator.
(5) Simulator: Whether this work is based on a simulator.
(6) Greedy Algorithm: Whether this work uses greedy algorithms to calculate or estimate the execution time of MapReduce tasks. This is a separate research problem as different Map/Reduce scheduling strategies will lead to varying runtime performance (e.g., Mapper/Reducer response time). However, analysing how different scheduling strategies affect run-time performance is not the focus of this paper. In our model this is an input parameter available through workload benchmarking.
(7) Network: Whether this works considers the impact of the network.

In summary, both classes of CTDT applications surveyed, lack the ability to represent the key metrics that influence the performance of CTDT applications across cloud layers. In order to support QoS guarantees (which we believe will be an essential part of future CTDT applications), it is essential to understand the impact of the application's components on each layer in order to optimize and orchestrate cloud resources. To the best of our knowledge, we are the first to present a performance analysis that considers the performance metrics within all end-to-end layers of a typical CTDT application, as well as the dependencies between each of those metrics.

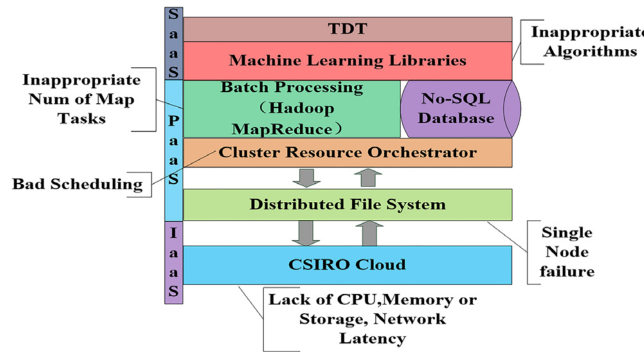


Fig. 1. Factors which affect the performance in different layers.

3. Multi-layered performance model for CTDT Big Data applications

3.1. Background

Let us consider a disease detection CTDT system. Such a system could potentially use a combination of MapReduce, HDFS and Amazon or Spark Streaming, HDFS and Windows Azure or Storm, HDFS and Google Compute Engine. The goal of such a CTDT application is to provide timely and accurate notification to its users allowing them to respond to adverse events such as earthquakes or diseases outbreak. Current CTDT approaches depend on QoS guarantees provided by the cloud provider, which are limited and restrictive. For instance, it limits QoS to IaaS resources such as CPU, Memory and Storage [7]. However, to support CTDT applications such as the ones described earlier, there is a need to go beyond a simple QoS guarantee strategy to a more end-to-end approach, i.e., the QoS must satisfy constraints such as *events detected within x minutes of occurrence and notification delivered with y minutes.* We need to acknowledge that factors exerting substantial effects on the performance of a CTDT application come from different layers (SaaS, PaaS, and IaaS). For example, consider a typical Batch Processing architecture (e.g., MapReduce) presented in Fig. 1. From the figure, we can see that several factors from different layers can affect the performance of a system. In the *machine learning libraries* layer, the accuracy and precision of the classification techniques such as the *Support Vector Machine (SVM)* and the *Naive Bayesian* model depends on the underlying input data sets (e.g., Tweets). However, in this work we validate the performance analysis technique in context of *Naive Bayesian* classification algorithm. Moreover in a MapReduce-based TDT application, the optimal number of Map Tasks is also essential for achieving the highest speed of a system. In addition, an appropriate scheduling method equally has a pivotal role to play in the speed of a system. For a CTDT application using a master–slave distributed file system (e.g., HDFS), single failure is obviously a catastrophe in terms of speed. In IaaS layer, for example, whether the applied memory is sufficient has a significant influence on the speed of a Spark-based TDT application.

In summary, we cannot ignore factors from any layer. Also in addition to considering factors from all layers we need to identify

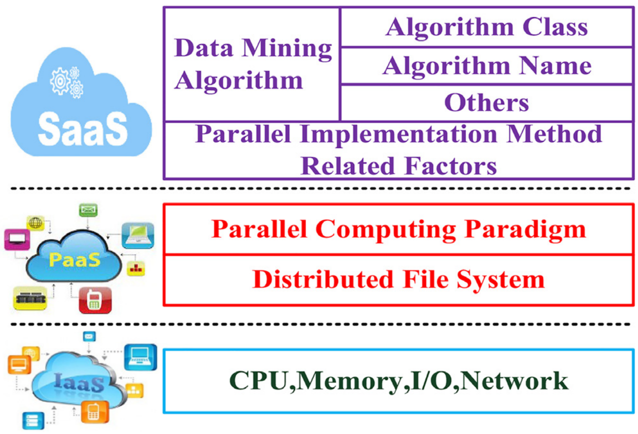


Fig. 2. Architecture of a performance analysis framework for CTDT applications.

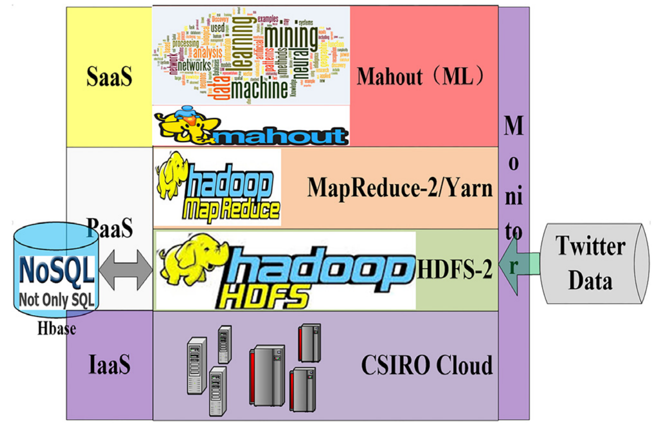


Fig. 3. Architecture of distributed disease detection system.

dependencies between these factors and how they can influence the performance of big data applications. Because, commonly, the cooperative effect of more than one metric has more effect or at least has equal effect on performance. Finally, the developed analysis needs to cater to a range of CTDT big data applications rather than being constrained to a specific class.

3.2. Metrics influencing the performance of CTDT applications

To capture the performance of CTDT applications, the first step would be to identify and determine which metrics should be used to measure the performance of a CTDT application at each layer. There are different performance metrics in terms of different practical needs. Be that as it may, there can be certain common important metrics that can be applied to most TDT applications such as speed, accuracy, price (for commercial applications), etc. Regardless of economic terms, speed is the factor of first-rate importance in a CTDT application particularly for mission critical disaster detection systems such as epidemic detection, earthquake detection, fire detection, etc. Consider earthquake detection for instance. Detecting the earthquake and warning citizens even a fraction of a minute earlier may save many lives. Furthermore, accuracy is another important metric for CTDT applications. A speedy but inaccurate traffic congestion detection system aiming to inform travellers about traffic jams or even suggest alternative routes, for example, would mean nothing because it provides outdated or fraudulent information that misleads travellers, and could even lead to more traffic jams. We select speed and accuracy as two key metrics in our performance analysis. Speed can be measured by calculating the execution time of a CTDT application while accuracy differs in different kinds of CTDT applications in terms of different data mining algorithms adopted. Within this paper, and for the purposes of our experiments, “precision” is used to describe the accuracy of classification algorithms whereas “perplexity” is used to measure the accuracy of clustering algorithms.

3.3. CTDT Big Data applications: Performance analysis framework

Fig. 2 illustrates our proposed performance analysis framework. We develop a generic framework that could be easily adopted to model a range of CTDT big data applications that could include several technologies at each of the IAAS, PAAS and SAAS layer.

Data Mining Algorithm means the group of factors related to the data mining algorithm adopted. Different kinds of data mining algorithms [21] have different effects on both accuracy and speed. For instance, as we discussed before, measuring the accuracy of a clustering algorithm based CTDT application requires the calculation of perplexity. In contrast, for a classification based one, we need to compute the precision. *Algorithm Class* means the type of data mining algorithm (e.g., Classification or Clustering) while *Algorithm Name* means the exact algorithm used (e.g., K-means, LDA, Naive Bayesian, etc.). Even in the same class, different algorithms might influence the performance of a system in different ways. For example, K-means and Canopy are both clustering algorithms, yet their influences on the speed of the system are substantially different, as K-means can be executed in more than one iteration whilst Canopy has only one iteration. *Others* refers to factors that might be important but beyond the scope of our existing work (providing scope for improvement). *Parallel Implementation Method* represents factors related to different parallelling methods of conversion of sequential data mining algorithms into parallel ones, such as MapReduce or MPI. *Parallel Computing Paradigm* means the different kinds of parallel computing frameworks adopted and relevant factors such as MapReduce (e.g., the factor of *Number of Mappers or Reducers*), Storm, Spark, etc. *Distributed File System* refers to factors related to the distributed system such as Hadoop Distributed File System. In the *IaaS* layer, we consider CPU, memory, I/O and Network related factors.

From the above architecture, it is obvious that our performance analysis defines several groups of factors rather than specific factors. Because different CTDT applications might adopt different implementation methods, such as different parallel computing paradigms (MapReduce or Storm). Our performance analysis can now be applied to almost all MapReduce-based TDT applications. In the next step, we will illustrate how to use it for a MapReduce-based Flu Detection system.

4. Using the multi-layered performance analysis framework to understand MapReduce-based TDT applications

In this section, we demonstrate how the proposed multi-layered performance analysis framework could study the impact of key identified parameters for MapReduce-based TDT application.

4.1. MapReduce based TDT application architecture

We present the architecture of a MapReduce [22,23] based TDT application in Fig. 3. The disease detection TDT application in this scenario uses data from Twitter to detect Flu-related events by analysing the tweets. First, we store the Twitter data in HDFS (Hadoop Distribute File System). In our work, the data was provided by COSMOS project (https://www.cs.cf.ac.uk/cosmos/).

Table 3
Factors in the IaaS layer.

Name	Explanation
T	The capacity of a single node (the number of floating point operations FLOPs per second).
B	The bandwidth (Mbps).
P	The number of computers.
$Data$	Including data size and information in data.

We run MapReduce-2 and the HBase Database. On top of Hadoop, we employ Mahout [24] which is a distributed and scalable machine learning library. One of the advantages of Mahout is that most of its ML algorithms can be executed as a MapReduce job. The disease detection application (also known as an "epidemic detection" application) [25,26] is built on a combination of clustering, classification and topic detection algorithms.

4.2. Modelling of the disease detection system

As discussed in Section 3, speed is an important performance metric, therefore we will discuss how to model the speed of a MapReduce TDT application (Diseases Detection System).

The execution time of a MapReduce TDT process is actually a MapReduce data mining process consisting of one or more MapReduce jobs. In the MapReduce paradigm most jobs are executed in a sequential way, therefore calculating the execution time of a MapReduce data mining process can be divided into two parts: calculating the execution time of a single MapReduce job and calculating how many MapReduce jobs contained in a MapReduce data mining process. To calculate the execution time of a single MapReduce job, we need to identify the process of a single MapReduce job.

The calculation of a single MapReduce job involves capturing the performance in IaaS, PaaS and SaaS layers. Factors relevant to the IaaS Layer can be seen in Table 3. We explain the details of PaaS and SaaS using the example of using Naive Bayes' classification for predicting disease types.

4.2.1. IaaS layer analysis factors

As discussed earlier, the analysis can be divided into three independent layers, which have dependencies on each other. We will illustrate the practical use of the analysis based on the diseases detection application in terms of the three layers. Factors relevant to the IaaS Layer can be shown in Table 3:

4.2.2. PaaS layer analysis factors

For the PaaS Layer, by adopting Hadoop MapReduce and HDFS, the factors of the performance analysis are listed as shown in Table 4.

In fact, a MapReduce-based Data mining algorithm consists of one or several MapReduce jobs. Now we can calculate the execution time of each MapReduce job. The total execution time of a MapReduce job can be computed according to Eq. (1).

$$T_{total} = T_{map} \times StartPercent + T_{shuffle} + T_{reduce} \tag{1}$$

Execution time of a map task can be computed using Eqs. (2) and (3).

$$T_{map} = T_{umap} \times N_{map}/P \tag{2}$$

$$T_{umap} = W_{map} \times C_{umap}/T \tag{3}$$

C_{umap} depends on several factors, such as the CPU speed, memory size, and available network bandwidth, etc. From the above formula, we can see that by increasing the number of nodes (i.e. number of Map and Reduce instances), end-to-end execution time of a MapReduce job (and the CTDT application) can be reduced. Unfortunately, it is not always the case, due to that C_{umap} will change with the changing of other parameters, such as CPU, Memory, Number of Mapper, etc.

In the MapReduce based Hadoop framework the N_{map} parameter (number of Map Tasks) is determined by setting: "dfs.block.size", "mapred.map.tasks", "mapred.min.split.size", "input data size", "goal number of mapper", and "mapred.max.split.size". How to compute N_{map} will be illustrated in the following equations.

The execution time of a reduce task can be computed by using Eqs. (4), and (5).

$$T_{reduce} = T_{ureduce} \times N_{reduce}/P \tag{4}$$

$$T_{ureduce} = W_{reduce} \times C_{ureduce}/T \tag{5}$$

The coefficient $C_{ureduce}$ is similar to C_{umap}, and the only difference is that $C_{ureduce}$ is for Reduce tasks (Reducer). The formula to compute the execution time of a shuffle task is shown in (6) and (7).

$$T_{shuffle} = W_{uoutmap} \times (N_{map} \bmod P)/B \tag{6}$$

$$W_{uoutmap} = W_{outmap}/N_{map} \tag{7}$$

The number of Map tasks is determined by the following parameters: size of block in HDFS "dfs.block.size", the goal number "mapred.map.tasks", the minimum size of splitting data for each mapper "mapred.min.split.size" and the maximum size of splitting data for each mapper "mapred.max.split.size".

4.2.3. SaaS layer analysis factors

We consider the Naive Bayes' classification ML algorithm [27] to aid our discussion of performance modelling at this layer. Except speed, we will also discuss the accuracy ("Precision" for Classification algorithms). See Table 5 which details features of this ML algorithm.

The total execution time of the whole classification process can be computed as shown below in Eq. (8).

$$T_{bayes} = N_{job} \times T_{total} \tag{8}$$

"RunSequential" is a special parameter which determines if the Naïve Bayesian training process has to be executed in a MapReduce way. If this is set to "true", the training set will be executed in a sequential way. This can be a typical situation for a TDT application where the training data is not large enough to be processed in parallel by exploiting the MapReduce distributed parallel programming abstractions. While the training can be done sequentially on one cluster node, the actual classifying (testing) phase can be implemented in the MapReduce way. In other words, the performance analysis has to capture such complex configuration decisions if it has to guarantee end-to-end execution times.

As noted earlier, Precision depends on the underlying ML algorithms and the type of data sets under consideration. Hence, we need to undertake various experimental studies to verify which ML algorithm leads to best possible precision for a given dataset. Even for a given classification (ML) algorithm, the precision can change due to the changing of other parameters. For instance, the parameter "complement" determined if the MapReduce Naïve Bayesian

Table 4
Factors in the PaaS layer.

Name	Explanation
T_{total}	The execution time (seconds) of a single MapReduce job.
T_{map}	The execution time (seconds) of a mapper task.
$T_{shuffle}$	The execution time (seconds) of a shuffle task.
T_{reduce}	The execution time (seconds) of a reducer task.
P_{start}	The percentage of the finished mapper tasks when the "shuffle" starts.
W_{map}	The product of the amount of workload for a single mapper task and it is related to the set of the blocksize of the HDFS, the split of the MapReduce, the data size (IaaS).
N_{map}	The number of the mapper tasks.
W	The workload of the whole input data size (MB or GB).
C_{umap}	Coefficient describing the relationship between the node (TaskTracker) and mapper.
$C_{ureduce}$	Coefficient describing the relationship between the node (TaskTracker) and Reducer.
$T_{ureduce}$	The execution time (seconds) of a single Reduce task.
N_{reduce}	The number of reducer tasks.
W_{reduce}	The workload for a single reducer task.
T_{umap}	The execution time (seconds) of a single mapper task.
$W_{uoutmap}$	The workload of the single mapper task.
W_{outmap}	The workload of all the mapper task.
B_{HDFS}	Blocksize of HDFS.
$N_{replication}$	The number of replication of data in HDFS.
MaxMemory of Map & Reduce task	Maximum Memory allocated to mapper or reducer Task can use, it can affect the execution time of a single task.

Table 5
Factors in the SaaS layer.

Name	Explanation
Execution Time	The whole execution time of Bayes' classification includes time taken for training, testing and learning.
Precision	The accuracy of the classification.
Njob	Depends on the PaaS level the number of Mapper and Reducer parameter/factor.
Class of ML Algorithm	Classification.
Name of ML Algorithm	Naïve Bayes'.
Complement (Boolean value)	Training process is based on C Naïve Bayesian or Standard Naïve Bayesian.
RunSequential (Boolean value)	MapReduce way or sequential way.

classifier is trained by using "Complementary Naïve Bayesian". This could lead to different precision as compared to standard Naïve Bayesian.

Some parameters in this layer might have influence on factors from other layers or require the assistance from factors of other layers to cooperate in order to affect the speed, or precision of the system. For instance, the "RunPartial" (MapReduce-based Random Forest) will determine if the MapReduce job will be executed in memory. If the MapReduce job is executed in memory, the job will be memory-intensive and more memory (IaaS resources) might lead to less execution time of Random Forest MapReduce-based jobs. One of the advantages of our performance analysis is that it can capture or reveal these inter-layer dependencies. Next, we will discuss such specific dependencies in relation to different algorithms such as Random Forest, Naïve Bayesian.

The basic theory of Naïve Bayesian classifiers is to group an unclassified item into a class where such an item has the highest probability related. For instance, $x = a_1, a_2...a_m$ is an unclassified dataset and each a is a feature of x while $C = y_1, y_2...y_n$ is a set of all classes and each y represents a class. Take the disease detection application for instance, x is an unprocessed tweet and y_i means a sort of known epidemic such as "flu", "measles" or "Ebola".

Traditionally a naïve Bayesian classifier process is sequential, which means it will not scale to processing of large volumes of Big Data. To efficiently process big data, it is better that naive Bayesian classification algorithm should be parallelized. We adopt the MapReduce programming model to parallelize naive Bayesian classification algorithm and explain the key steps and analysing factors in the following.

There are two main steps in the training part of naïve Bayesian classifier: (1) Counting the ClassPrior $P(y_i)$ for each class. (2) Counting the conditional probability for each attribute per class $P(a|y_i)$ (in text classification, the attribute can be the word).

As a consequence, it is necessary for a naïve Bayesian classifier based on MapReduce to have two main MapReduce jobs to undertaking the above 2 steps: first for counting the Classifier and second for computing the conditional probability.

The practical implementation of naïve Bayesian classifier in MapReduce varies, especially for the training part. In this paper, we study Naïve Bayesian implementation based on the MapReduce framework by exploiting Mahout (an open-source scalable machine learning library) as an ML engine. Even in Mahout, the specific implementation of naïve Bayesian classifier has been changed since its initial release.

4.3. Dependency across layers

Here, we use the training process of Naïve Bayes in Mahout to illustrate the dependencies across layers as shown in Fig. 4. Red lines represent dependencies across different layers while purple lines represent dependencies within the same layer.

(1) Dependency1: Dependency between "Complement" and "Data Information". Means that the influence of "Complement" on "Precision" might be affected by "Data Information". For instance, the Complementary Naïve Bayesian method is more effective for classifying unbalanced data than the balanced data.

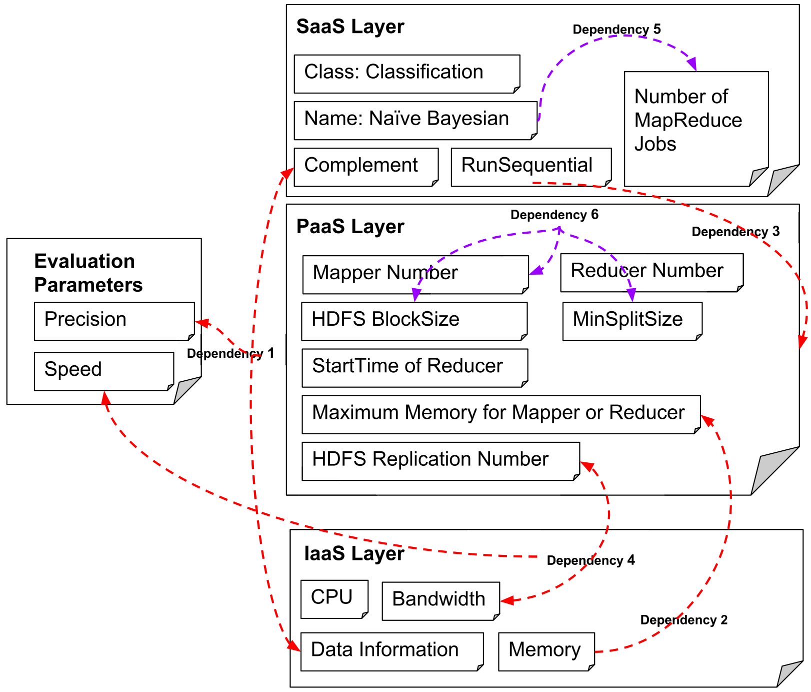


Fig. 4. Dependency across layers. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

(2) Dependency2: Dependency between "Memory" and "Maximum Memory for a Mapper or a Reducer". It means that "Memory" in IaaS layer has to cooperate with "Maximum Memory for a Mapper or a Reducer" in order to manage speed of executing of analytics tasks. Specifically, without tuning the "Maximum Memory for a Mapper or a Reducer", Memory available at the IaaS layer cannot affect the performance of the underlying TDT application. On the other hand "Maximum Memory for a Mapper or a Reducer" has an upper bound. For instance, if the total available Memory (IaaS) is 1000 MB and the total number of Reducers and Mappers are 10, the "Maximum Memory" cannot be over 100 MB, and otherwise the MapReduce job execution will not commence.

(3) Dependency 3: Dependency between "RunSequential" and MapReduce. As we mentioned before, the MapReduce training process can be executed only when this parameter is false.

(4) Dependency 4: Dependency between "Bandwidth" and "HDFS Replication Number". When the replication of data is not enough, the node might need to copy required data from another node to process. In this situation, "Bandwidth" has a more significant role to play in the speed of the system, for the reason that low Bandwidth might lead to the slow speed of copying data from one node to another.

(5) Dependency5: It is an inner Dependency within the SaaS layer. The parameter "Number of MapReduce jobs" is determined by the parameter "Name of Algorithm". In a Naïve Bayesian performance analysis it is 3 for training set (in the new edition of Mahout) and 1 for the testing (classifying) part, and for another classification algorithm (e.g. Random Forest), it might require a different number of tasks at training and testing steps/phases.

(6) Dependency6: An inner Dependency within PaaS layer, the number of mappers is affected by the size of the HDFS block and the min splitting size of input data.

5. Experimentation and evaluation

5.1. Experimental environment

The environment of our experiments is based on a CSIRO ICT Cloud which is built with OpenStack. There are 10 clusters adopted in our experiment, shown in Table 6. As explained previously, the data for our experiments is collected from Twitter in order to detect outbreaks of flu.

We did all the experiments that required maximum 4 nodes at first, then we created snapshots of Clusters 1–4 and we did experiments on Cluster 5, Cluster 6, Cluster 7, Cluster 10. The reason we run our experiment under these different settings is that we will present the effect generated by the IaaS resources upon the speed of the system.

5.2. Experimental results

5.2.1. IaaS experiment (number of VCPU cores)

Description of Experiment: In accordance with our performance analysis, when the CPU resource is enough, increasing of the CPU resource does not affect the execution time of a TDT application significantly. However, when the CPU resource is in shortage, for example, there is only a virtual machine with 1 core CPU in a cluster and the MapReduce-based data mining algorithm in a TDT application requires more than 5 Map tasks, the increasing of CPU resource might lead to the increasing of the speed of such a TDT application.

Because our system is built on CSIRO Cloud where we do not have the highest level of access privilege, we can only change the number of VCPU (Virtual CPU) cores. As we mentioned in this chapter, we built cluster 1–10 (Shown in Table 6). To eliminate the effect of memory, we kept the memory size of each Mapper or Reducer unchanged and the number of Mappers or Reducers unchanged. In this experiment, we chose Naive Bayesian as our algorithm, Fig. 5 shows the result of the experiment.

The first figure shows that the speed increased with the increasing of the CPU core number, but the second figure shows

Table 6
Clusters adopted in experiments in CSIRO ICT Cloud.

Cluster	Specification
Cluster 1	1 node, pseudo-distributed Hadoop 2, HDFS 2, 1 CPU,
Cluster 2	2 nodes, 1 master node (Namenode, ResourceManager, JobTracker), 1 slave node (DataNode, NodeManager, TaskTracker), Hadoop2.4.1, Mahout 1.0, 2 CPU cores (2.40 GHz)
Cluster 3	3 nodes, 1 master node, 2 slave nodes, Hadoop2.4.1, Mahout 1.0, 3 CPU cores (2.40 GHz)
Cluster 4	4 nodes, 1 master node, 3 slave nodes, Hadoop2.4.1, Mahout 1.0, 4 CPU cores (2.40 GHz)
Cluster 5	5 nodes, 1 master node, 4 slave nodes, Hadoop2.4.1, Mahout 1.0, 5 CPU cores (2.40 GHz)
Cluster 6	6 nodes, 1 master node, 5 slave nodes, Hadoop2.4.1, Mahout 1.0, 6 CPU cores (2.40 GHz)
Cluster 7	7 nodes, 1 master node, 6 slave nodes, Hadoop2.4.1, Mahout 1.0, 7 CPU cores (2.40 GHz)
Cluster 8	8 nodes, 1 master node, 7 slave nodes, Hadoop2.4.1, Mahout 1.0, 8 CPU cores (2.40 GHz)
Cluster 9	9 nodes, 1 master node, 8 slave nodes, Hadoop2.4.1, Mahout 1.0, 9 CPU cores (2.40 GHz)
Cluster 10	10 nodes, 1 master node, 9 slave nodes, Hadoop2.4.1, Mahout 1.0, 10 CPU cores (2.40 GHz)

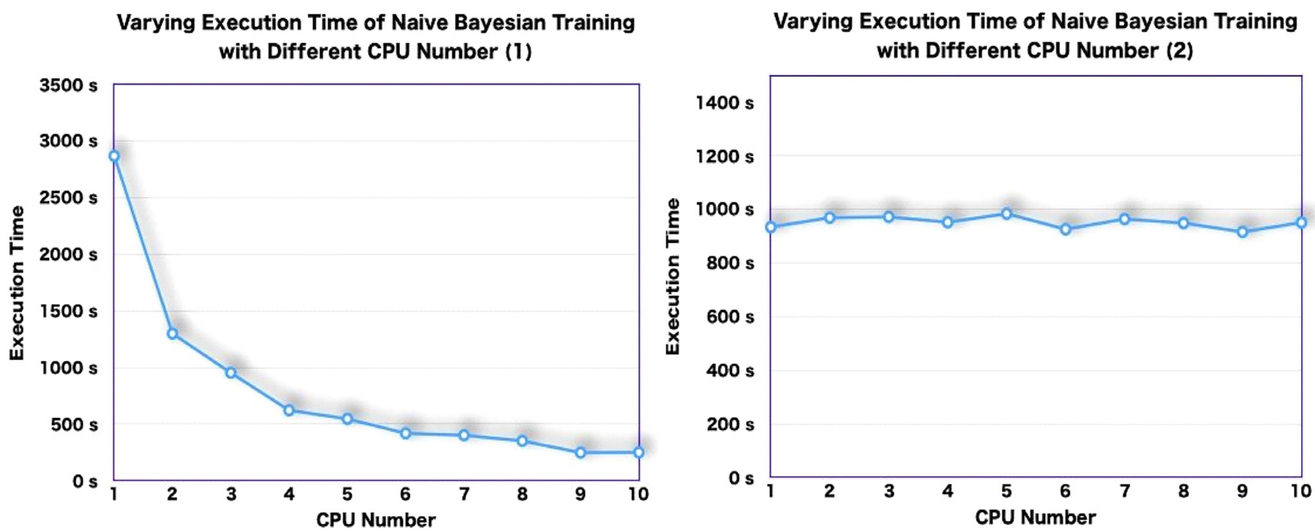


Fig. 5. Result of execution time of Naïve Bayesian trainings with different number of CPU cores.

that the speed was not affected by the increasing of the CPU core number. The first figure shows a group of experiments based on the Mapper number of "18" while the second figure represents a group of experiments based on the Mapper number of "1". Specifically, in the first group from cluster 1 to 10, the CPU resource of each cluster might not have the maximum required CPU resource, which led to the situation that all the Mappers might not be able to start at the same time. Consequently, with the increasing of CPU resources, the number of Mappers which could be open in the meantime increased and this led to the increasing of the speed. The second group of experiments was based on 10% of the data of the first group (for saving time), 1 Mapper and 2 Reducer (the same as with the first group). The Mapper Number is "1" and we set the parameter "mapreduce.map.cpu.vcores" (number of virtual cores to request from the scheduler for each map task) as "1" and "mapreduce.reduce.cpu.vcores" (number of virtual cores to request from the scheduler for each reduce task) as "1".

Conclusion of Experiment: The number of CPU cores will affect the speed when the CPU resource is so little that it cannot start all the Mappers at the same time. When the CPU resource is sufficient, the increasing of CPU numbers cannot affect the speed significantly. The result also shows how the influence of different parameters: "Number of CPU" (IaaS), "Mapper Number" and "mapreduce.map.cpu.vcores" might affect the speed together. This has been identified in our performance analysis.

5.2.2. PaaS experiment (number of mappers and reducers)

Description of Experiment: As mentioned in our performance analysis there is an optimal number of Mappers for the speed of the system, and the number of Mappers might affect the speed significantly. We change the number of Mappers and make other factors fixed. The result can be shown in Fig. 6.

Conclusion of Experiment: From the results, we can conclude that the number of Mappers can affect the speed of Naïve Bayesian training and Random Forest Training. Furthermore, there is an optimal number of Mappers for a MapReduce-based Naïve Bayesian and Random Forest (Training). There is an optimal value for mapper number that can achieve a minimum execution time.

5.2.3. SaaS experiment

Description of Experiment: According to our performance analysis, the other parameter possessing a significant role to play in the performance of a TDT application based on Naïve Bayesian is "trainComplementary" which determines whether the Naïve Bayesian algorithm is executed as "Complementary Naïve Bayesian" or "Standard Naïve Bayesian". Complementary Naïve

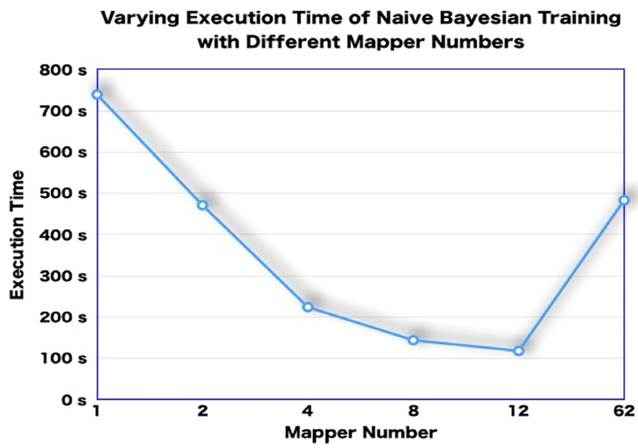


Fig. 6. Execution time of Naïve Bayesian training with different mapper numbers.

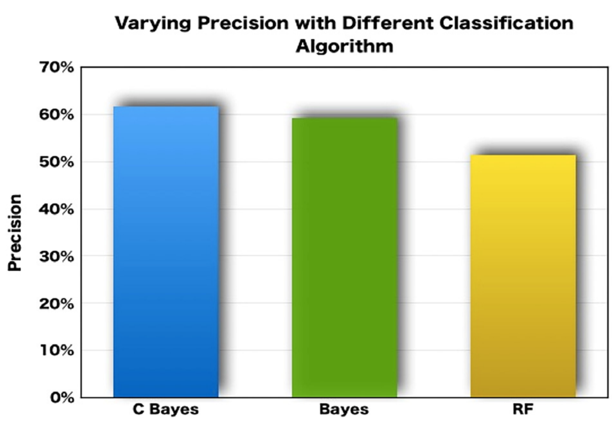


Fig. 7. Varying precision with different classification algorithms.

Bayesian is a Naïve Bayesian variant overcoming some weaknesses of the standard Naïve Bayesian. The Naïve Bayesian classifier tends to classify documents into a category possessing a great number of documents while the complementary uses data from all categories apart from the category that is worked on.

This parameter might affect the precision of the system, in accordance with our performance analysis. To evaluate the effect of this parameter, we conducted the following experiment: keeping other parameters unchanged and seeing the result of precision in terms of different kinds of classification algorithms. In this experiment, we also compare the precision of Random Forest classifier with the same data. The result of this experiment is shown in Fig. 7.

Conclusion of Experiment: The parameter "Complement" can control whether the classification algorithm is based on C Bayes or standard Bayes and indirectly affects the precision of the system. Furthermore, the parameter "name of classification algorithm" can affect the precision of the classification-based system, which means different classification algorithms have a different precision based on the same data.

5.3. Evaluation summary

In conclusion, our experiments show that our performance analysis has achieved the following: (1) Our performance analysis is capable of capturing metrics which affect the performance of a CTDT application across all three layers. (2) Our performance model can illustrate the dependencies between these metrics. (3) Our performance analysis can reflect on how these factors affect performance. (4) Our performance analysis can be used to predict the execution time of a TDT application under various conditions. As discussed in Section 2 to the best of our knowledge we are the first to present a performance analysis that considers the performance metrics within all end-to-end layers of a typical CTDT application, as well as the dependencies between each of those metrics.

6. Conclusions and future work

Cloud computing technology offers a possible solution to tackle new challenges of TDT (Topic Detection and Tracking) techniques in the Big Data era. However, this new combination of Cloud resources and TDT (CTDT) generates a new issue – how to analyse the performance of CTDT to meet the demands posted by big data applications. Our performance analysis framework provides a practical and generic solution to analyse and model the performance of big data-based CTDT applications. We demonstrate the effectiveness of the performance analysis framework using the case study of MapReduce-based TDT applications. Within our analysis, we have identified key parameters in each cloud layer, and established the dependencies between these metrics across the layers. We have also demonstrated and validated the correctness of parameters and their relationship across cloud layers via experimental evaluations using real-world datasets.

There are a number of issues that require further work. For example, we need to apply this performance analysis framework to more MapReduce-based TDT applications using different data mining algorithms, such as Random Forest, LDA, SVM, etc. Moreover, we will also extend this performance analysis framework to other classes of Big Data Applications, which can be based on other types of programming paradigms such as Stream Processing. To achieve such generalization, we will extend the performance analysis framework to capture the limited sets of data flow and analytic patterns generated by different classes of Big Data Applications (e.g., real-time traffic modelling, and real-time energy modelling). For example, in context of the Stream Processing paradigm, we will need to consider real-time analytic latency as the most important performance model parameter (at the PaaS layer), as compared to batch processing response time of the Hadoop programming paradigm. In our view, such extensions will not affect formulations across the other layers of the Big Data stack including SaaS and IaaS.

Acknowledgement

This work is partly funded by SNSF NRP75, Switzerland project 407540_167266.

References

[1] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, S.U. Khan, The rise of "big data" on cloud computing: Review and open research issues, Inf. Syst. 47 (2015) 98–115.
[2] X. Lin, Z. Meng, C. Xu, M. Wang, A practical performance model for Hadoop MapReduce, in: Proceedings of the 2012 IEEE International Conference on Cluster Computing Workshops, CLUSTER WORKSHOPS 2012, IEEE Computer Society, Beijing, China, 2012, pp. 231–239.
[3] M.J. Fischer, X. Su, Y. Yin, Assigning tasks for efficiency in Hadoop: extended abstract, in: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2010, ACM Press, Thira, Santorini, Greece, 2010, pp. 30–39.
[4] J. Berlińska, M. Drozdowski, Scheduling divisible MapReduce computations, J. Parallel Distrib. Comput. 71 (3) (2011) 450–459.
[5] E. Solaiman, R. Ranjan, P.P. Jayaraman, K. Mitra, Monitoring internet of things application ecosystems for failure, IT Prof., IEEE 18 (5) (2016) 8–11.
[6] R. Ranjan, S. Garg, A. Khoskba, E. Solaiman, J. Philip, D. Georgakopoulos, Orchestrating bigdata analysis workflows, IEEE Cloud Comput. 4 (3) (2017) 20–28.
[7] P.P. Jayaraman, K. Mitra, S. Saguna, T. Shah, D. Georgakopoulos, R. Ranjan, Orchestrating quality of service in the cloud of things ecosystem, in: 2015 IEEE International Symposium on Nanoelectronic and Information Systems, 2015, pp. 185–190. http://dx.doi.org/10.1109/iNIS.2015.64.

[8] M. Yun, D. Bragg, A. Arora, H.-A. Choi, Battle event detection using sensor networks and distributed query processing, in: Proceedings of the 1st International Workshop on Cyber-Physical Networking Systems, IEEE Press, Shanghai, China, 2011, pp. 761–766.

[9] C. Kühnert, M. Baruthio, T. Bernard, C. Steinmetz, J.-M. Weber, Cloud-based event detection platform for water distribution networks using machine-learning algorithms, Procedia Eng. 119 (2015) 901–907.

[10] H. Deng, Q.-A. Zeng, D. Agrawal, SVM-based intrusion detection system for wireless ad hoc networks, in: Proceedings of the IEEE 58th Vehicular Technology Conference, VTC2003-Fall, IEEE Press, Orlando, Florida USA, 2003, pp. 2147–2151.

[11] M. Bahrepour, N. Meratnia, M. Poel, Z. Taghikhaki, P.J. Havinga, Distributed event detection in wireless sensor networks for disaster management, in: Proceedings of the 2nd International Conference on Intelligent Networking and Collaborative Systems, INCoS 2010, IEEE Computer Society, Thessaloniki, Greece, 2010, pp. 507–512.

[12] N. Dziengel, G. Wittenburg, J. Schiller, Towards distributed event detection in wireless sensor networks, in: Proceedings of the 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems, DCOSS 2008, 2008, pp. 507–512.

[13] A. Verma, L. Cherkasova, R.H. Campbell, ARIA: automatic resource inference and allocation for MapReduce environments, in: Proceedings of the 8th ACM International Conference on Autonomic Computing, ICAC 2011, ACM Press, Karlsruhe, Germany, 2011, pp. 235–244.

[14] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, N. Koudas, MRShare: sharing across multiple queries in MapReduce, Proc. VLDB Endowment 3 (1–2) (2010) 494–505.

[15] X. Cui, X. Lin, C. Hu, R. Zhang, C. Wang, Modeling the performance of MapReduce under resource contentions and task failures, in: Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, IEEE Computer Society, Bristol, UK, 2013, pp. 158–163.

[16] L. Xu, MapReduce framework optimization via performance modeling, in: Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPSW 2012, IEEE Computer Society, Shanghai, China, 2012, pp. 2506–2509.

[17] H. Herodotou, Hadoop Performance Models, CS-2011-05, Computer Science Department, Duke University, 2011.

[18] M. Khan, Y. Jin, M. Li, Y. Xiang, C. Jiang, Hadoop performance modeling for job estimation and resource provisioning, IEEE Trans. Parallel Distrib. Syst. 27 (2) (2016) 441–454.

[19] H. Tamano, S. Nakadai, T. Araki, Optimizing multiple machine learning jobs on MapReduce, in: Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science, CloudCom 2011, IEEE Computer Society, Athens, Greece, 2011, pp. 59–66.

[20] J. Han, M. Ishii, H. Makino, A Hadoop performance model for multi-rack clusters, in: Proceedings of the 5th International Conference on Computer Science and Information Technology, CSIT 2013, IEEE Press, Amman, Jordan, 2013, pp. 265–274.

[21] L. Zhang, L. Jiang, C. Li, G. Kong, Two feature weighting approaches for naive bayes text classifiers, Know.-Based Syst. 100 (C) (2016) 137–144. http://dx.doi.org/10.1016/j.knosys.2016.02.017.

[22] L. Bing, K.C. Chan, A paralleled big data algorithm with MapReduce framework for mining twitter data, in: Proceedings of the 4th International Conference on Big Data and Cloud Computing, BdCloud 2014, IEEE Computer Society, Sydney, Australia, 2014, pp. 121–128.

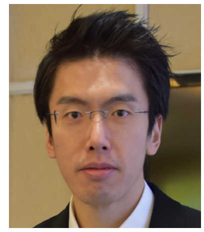
[23] C. Hu, J. Zhao, X. Yan, D. Zeng, S. Guo, A MapReduce based parallel niche genetic algorithm for contaminant source identification in water distribution network, Ad Hoc Netw. 35 (2015) 116–126. http://dx.doi.org/10.1016/j.adhoc.2015.07.011. Special Issue on Big Data Inspired Data Sensing, Processing and Networking Technologies, http://www.sciencedirect.com/science/article/pii/S1570870515001468.

[24] R.M. Esteves, C. Rong, Using mahout for clustering wikipedia's latest articles: A comparison between K-means and fuzzy C-means in the cloud, in: Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science, CloudCom 2011, IEEE Computer Society, Athens, Greece, 2011, pp. 565–569.

[25] J. Huang, H. Zhao, J. Zhang, Detecting flu transmission by social sensor in China, in: Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GREENCOM-ITHINGS-CPSCOM 2013, IEEE Computer Society, Beijing, China, 2013, pp. 1242–1247.

[26] H. Achrekar, A. Gandhe, R. Lazarus, S.-H. Yu, B. Liu, Predicting flu trends using twitter data, in: Proceedings of the 1st International Workshop on Cyber-Physical Networking Systems, IEEE Press, Shanghai, China, 2011, pp. 702–707.

[27] L. Jiang, S. Wang, C. Li, L. Zhang, Structure extended multinomial naive bayes, Inf. Sci. 329 (C) (2016) 346–356. http://dx.doi.org/10.1016/j.ins.2015.09.037.



Meisong Wang holds a Masters in Philosophy (M.Phil.) from the Faculty of Engineering and Computer Science, Australian National University. His research interests are in Big Data, Cloud Computing, and the Internet of Things.

Prem Prakash Jayaraman is currently a Research Fellow at Swinburne University of Technology, Melbourne. His research areas of interest include, Internet of Things, cloud computing, mobile computing, sensor network middleware and semantic internet of things. He has authored/co-authored more than 50 research papers in international Journals and conferences such as IEEE Trans. on Cloud Computing, IEEE Selected areas in Communication, Journal of Computational Science, IEEE Transactions on Emerging Topics in Computing, Future Generation Computing Systems, Springer Computing, ACM Ubiquity Magazine, IEEE Magazine. He is one of the key contributors of the Open Source Internet of Things project OpenIoT that has won the prestigious Black Duck Rookie of the Year Award in 2013. He has been the recipient of several awards including hackathon challenges at the 4th International Conference on IoT (2014) at MIT media lab, Cambridge, MA and IoT Week 2014 in London and best paper award at HICSS 2016/2017 and IEA/AIE-2010. Previously he was a Postdoctoral Research Fellow at CSIRO Digital Productivity Flagship, Australia from 2012 to 2015.

Ellis Solaiman is a Lecturer at the School of Computing, Newcastle University. He previously received his Ph.D. in Computing Science also from Newcastle University, where he subsequently worked as a Research Associate and Teaching Fellow. His research interests are mainly in the areas of Dependability and Trust in Distributed Systems such as the Cloud and the Internet of Things. He is also interested in the automated monitoring of these systems using technologies such as Smart Contracts. He is a Fellow of the UK Higher Education Academy (FHEA) since 2016.

Lydia Y. Chen is a performance analyst at the Energy Management group of IBM Zurich Research Lab. She holds a Ph.D. in Operations Research and Industrial Engineering from Penn State University. She completed her undergraduate studies at National Taiwan University and British Columbia University. Her main research interests are in the areas of performance evaluation, power and workload management, big data and cloud computing and architecture-aware parallel algorithms.

Zheng Li received his Ph.D. degree and M.E. by Research degree from the Australian National University (ANU) and the University of New South Wales (UNSW) respectively. During the same time, he was a graduate researcher with the Software Systems Research Group (SSRG) at National ICT Australia (NICTA). Before studying abroad, he had around four-year industrial experience in China after receiving his M.Sc.Eng. degree from the Beijing University of Chemical Technology and the B.Eng. degree from the Zhengzhou University. His research interests include Cloud computing, performance engineering, empirical software engineering, software cost/effort estimation, and Web service compo-

Jun Song received the Bachelor's and Master's degrees from the China University of Geosciences, Wuhan, China, and the Ph.D. degree from Wuhan University, all in computer science. He is currently an associate professor of computer science with the China University of Geosciences. His area of specialization is cryptography application and information security, and his current research interests include security analysis of cryptography application in wireless networks, applied network security, and cryptography security for big data.

Dimitrios Georgakopoulos is a Prof. in Computer Science and Director of the Key Lab for IoT at Swinburne University of Technology, Melbourne, Australia. Before that was Research Director at CSIRO's ICT Centre and Executive Director of the Information Engineering Laboratory, which was the largest Computer Science program in Australia. Before CSIRO, he held research and management positions in several industrial laboratories in the US, including Telcordia Technologies (where he helped found two of Telcordia's Research Centers in Austin, Texas, and Poznan, Poland); Microelectronics and Computer Corporation (MCC) in Austin, Texas; GTE (currently Verizon) Laboratories in Boston, Massachusetts; and Bell Communications Research in Piscataway, New Jersey. He was also a full Professor at RMIT University, and he is currently an Adjunct Prof. at the Australian National University and a CSIRO Adjunct Fellow. Prof. Georgakopoulos has produced 170+ journal and conference publications in the areas of IoT, process management, and data management, and has 10,500+ lifetime citations.

Rajiv Ranjan is an Associate Professor (Reader) in Computing Science at Newcastle University, United Kingdom. Prior to that, he was a Senior Research and Julius Fellow at CSIRO, Canberra, where he was working on projects related to Cloud and big data computing. He has been conducting leading research in the area of Cloud and big data computing developing techniques for: (i) Quality of Service based management and processing of multimedia and big data analytics applications across multiple âĂÍ Cloud data centers (e.g., CSIRO Cloud, Amazon and GoGrid); and (ii) automated decision support for migrating applications to data centers. He has published about 110 papers that include 60+ journal papers. He serves on the editorial board of IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Cloud Computing, and Future Generation Computer System Journals. He is one of the highly cited authors (top 0.09%) in computer science and software engineering worldwide (h-index=43, g-index=94, and 10,050+ google scholar citations).