Analytical Composite Performance Models for Big Data Applications

Soroush Karimian-Aliabadi[a], Danilo Ardagna[b,*], Reza Entezari-Maleki[c], Eugenio Gianniti[b], Ali Movaghar[a]

[a] *Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.*
[b] *Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy.*
[c] *School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.*

Abstract

In the era of Big Data, whose digital industry is facing the massive growth of data size and development of data intensive software, more and more companies are moving to use new frameworks and paradigms capable of handling data at scale. The outstanding MapReduce (MR) paradigm and its implementation framework, Hadoop are among the most referred ones, and basis for later and more advanced frameworks like Tez and Spark. Accurate prediction of the execution time of a Big Data application helps improving design time decisions, reduces over allocation charges, and assists budget management. In this regard, we propose analytical models based on the Stochastic Activity Networks (SANs) to accurately model the execution of MR, Tez and Spark applications in Hadoop environments governed by the YARN Capacity scheduler. We evaluate the accuracy of the proposed models over the TPC-DS industry benchmark across different configurations. Results obtained by numerically solving analytical SAN models show an average error of 6% in estimating the execution time of an application compared to the data gathered from experiments and moreover the model evaluation time is lower than simulation time of state of the art solutions.

Keywords: Big Data, MapReduce, Apache Spark, Performance Evaluation, Stochastic Activity Network

*Corresponding author
Email addresses: skarimian@ce.sharif.edu (Soroush Karimian-Aliabadi), danilo.ardagna@polimi.it (Danilo Ardagna), entezari@iust.ac.ir (Reza Entezari-Maleki), eugenio.gianniti@polimi.it (Eugenio Gianniti), movaghar@sharif.edu (Ali Movaghar)

1. Introduction

Data have become a dominant phenomenon of today's digital world. Sheer volume of data produced in high velocity and great variety is one the main challenges of the information technology nowadays. Companies have to capture, process, store, and manage Big Data of their customers, suppliers, operations, and transactions, which is beyond the ability of typical database systems and conventional programming paradigms [1]. The amount of digital data is massively increasing in effect of emerging IT services, and especially with the advent of IoT technologies [2]. The 1.5EB digital world data volume estimation in 1999 has increased to 16.1ZB in 2016, and is predicted to raise up to 163ZB in 2025 [3]. The Big Data economy is increasing as well from about $0.7 trillion in 2016 to about $4 trillion in 2025 [3].

From the technical point of view, new applications, frameworks, and platforms are needed in order to facilitate Big Data processing. In this regard, Google introduced the MapReduce (MR) paradigm [4], which is capable of processing large amount of data, efficiently. Its open source implementation, Apache Hadoop became the most popular Big Data framework [5]. Both the MR paradigm and the Hadoop framework have gone through several improvements. For example, Apache Tez introduced the concept of Reduce/Reduce stage that significantly reduced the I/O by storing intermediate results on local disks instead of HDFS allowing the execution of applications with multiple map and reduce phases within a Directed Acyclic Graph (DAG) [6]. Also, YARN (Yet Another Resource Negotiator) layer has been dedicated to resource management tasks such as dynamic resource allocation and scheduling, with more sophisticated algorithms like Capacity and Fair scheduling [7]. A significant speedup in Big Data applications originated from the introduction of the Spark framework [8]. Similar to Tez, Spark's workflow is a DAG of stages, but the main advantage of the Spark framework is in its memory data processing model.

With the Hadoop framework growing in complexity, one of the main challenges is to estimate the execution time of a Big Data application [10]. An accurate estimation helps improving the cluster and application configuration both at design and run-time. It also

assists cloud providers to meet Service Level Agreements (SLAs) and facilitates resource planning for the user. Usually, execution time is empirically measured through costly experiments and in a time consuming process [11]. Models can help predicting execution time eliminating the setup and experiment costs.

MR application analytical performance models are mostly focused on the earliest version of Hadoop, where resource allocation is done statically and scheduling scheme is the simple FIFO policy [12]. Other more recent models, presented in this area are simulation-based models for which analysis is time consuming and less scalable [13, 14]. Methods based on machine learning are good for interpolation, but suffer from low generality and insight [15, 16, 17]. Moreover, machine learning needs costly cluster setup to study historical logs of past executions. Analytical models, on the other hand, can be analyzed faster with numerical solutions, and are more general and scalable in this context.

To fulfill this requirement, analytical models based on Stochastic Activity Networks (SANs) [18] are proposed in this paper to evaluate MR, Tez and Spark applications execution time. The proposed models are superior than previous approaches, which are mostly based on simulation [13, 19] or analyzing historical traces [20, 15] and perform accurate enough in the most widely used Big Data frameworks, from simple MR jobs to Spark applications.

An initial SAN model is first proposed to estimate the execution of MR applications on a Hadoop cluster governed by the YARN Capacity scheduler. The model is then extended to support more complex Tez and Spark applications in two ways. The first extension is a monolithic model, the second one is a composite model, which exhibits higher scalability. The execution time of Big Data applications are the target performance metric which is computed in the steady-state. While in [13], simulation models were proposed, the models in this paper are the first analytical contribution which allow to quickly estimate the execution time of a Big Data application with a good accuracy. We validate the proposed models through experiments on the CINECA [21], Italian Supercomputing Center, Flexiant cloud [22], and a private cluster based on Power 8 considering the TPC-DS industry benchmark [23]. Model parameters are first initialized according to the traces from a small pilot execution of the application, and then model is used to predict the application execution

time in the real scale scenario. The results of comparing real system experiments with the models prediction show that the proposed models can appropriately estimate the application execution time with 6% average error. The accuracy and execution time of the proposed models are then compared to previous work [13], and both an increase in the accuracy and a decrease in the model evaluation time are achieved.

The remaining parts of this paper are organized as follows. Section 2 is dedicated to the description of the features of the application frameworks, section 3 overviews the SAN models, and section 4 presents our proposed SAN models for Hadoop MR, Tez, and Spark applications in the both monolithic and composite forms. The results obtained by the proposed models and their validation against the real systems and the previous work [13] are reported in section 5. In section 6, we introduce related proposals available in the literature. Finally, in section 7, we conclude the paper with some directions for future work.

2. System Architecture and Application Structure

In this section, to provide a background for modeling activities, the architecture of the Big Data application frameworks considered in this paper are introduced. Hadoop started as the first industrial implementation of the MR paradigm and, despite the simple architecture, was effective enough to be widely used. Current Hadoop architecture is however, more mature and well improved. The resource management is separated from MR thanks to the new layer. Decoupling resource management and application logic enables Hadoop to execute not only MR applications but also any other application framework. As shown in Figure 1, Spark [8], Flink [24], and Tez are examples of more recent framework that can be run on top of YARN. Moreover, users can build data flows spanning over multiple stages whose dependencies can be modeled as a DAG and not limited to the map and reduce phases.

Each MR job consists of three phases called map, shuffle, and reduce. In the first phase, mappers start working on the input data chunks to produce the intermediate data. The number of the map tasks is proportional to both input data and chunk sizes. The shuffle phase starts, once the first mapper task is accomplished. There are the same numbers of

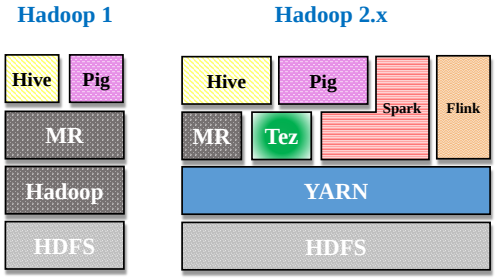


Figure 1: Hadoop evolution

shuffle and reduce tasks on a same thread. Shuffle tasks sort and partition key-value pairs, and assign each partition to a reduce task. Afterwards, reduce tasks start working, and the result of the reduce phase is written to the Hadoop Distributed File System (HDFS). Since each reduce task follows a shuffle task on a same thread, we decide to aggregate shuffle and reduce phases as reduce phase, and hereafter, reduce phase stands for the shuffle and reduce tasks together.

Individual map, shuffle and reduce tasks run on Data Nodes (DN) while the Master Node (MN) is responsible for scheduling the application. Although, jobs were scheduled in earlier versions of the Hadoop framework by FIFO policy, better schemes are available today. Hadoop 2.x and Hadoop 3 let more complex schedulers (i.e. capacity and fair schedulers) to be plugged into the framework. A cluster is a resource pool in YARN enabling dynamic allocation of resources (containers) to the ready tasks. Once a job is submitted to the cluster, an Application Master (AM) is assigned to the job and runs on a DN. The AM schedules tasks of a job, and handles the job execution. YARN layer enables multi-tenancy by sharing resources among multiple AMs running in a single cluster. Moreover, AM negotiates with Resource Manager (RM) to acquire containers. The RM, which is cluster specific and placed in the MN, is responsible for allocating resources to applications and scheduling jobs. Monitoring the containers and resources available in a single node is carried by the Node Manager (NM). The NM sends a heartbeat containing the node status to RM every once in a while. Tez is an application on top of the YARN, as shown in Figure 1, which enables users to execute their queries in the form of a DAG with vertices representing map or reduce phases. In other words, Tez extends the simple MR paradigm

Figure 2: DAG execution model in (a) Tez and (b) Spark applications

by introducing Reduce/Reduce (RR) stages and avoiding in this way to store intermediate results on HDFS. This is the main advantage of Tez, which reduces the synchronization delay of the phases. Note that, it is still possible to run the conventional MR jobs on top of Tez with a simple two-vertex graph. When a Tez application is submitted, the framework receives a DAG with each vertex representing a phase. The directed edges of the graph represent data dependencies between phases. A simple Tez application graph is shown in Figure 2a.

Nevertheless, the today's cutting edge Big Data framework is Spark, which was introduced in 2010. Spark's programming model is similar to MR/Tez but extends it with a data-sharing in memory abstraction called Resilient Distributed Datasets (RDD). RDDs can be cached and can be created from a file, by parallelizing an in-memory collection, or by mapping an existing RDD. Spark evaluates RDDs lazily, and postpones their creation until an action needs the actual evaluation of an RDD. Actions return a value after executing calculations on datasets.

RDDs achieve fault tolerance through the notion of lineage. Each RDD tracks the graph of transformations that was used to build it and reruns these operations on base data to reconstruct any lost partitions [25]. The other key concept in Spark is its DAG execution engine, which is similar to Tez and is our basis for extending the Tez model to Spark. A simple Spark application graph is shown in Figure 2b.

In this paper, we consider target Hadoop clusters running on a set of homogenous resources [26], including MR, Spark, and Tez execution engines on top of the YARN Capacity scheduler [13]. This implies that the cluster capacity is partitioned into multiple queues and within a queue, multiple applciations are scheduled in a FIFO manner.

We assume that multiple users can run the same query, which is submitted to a specific queue. Moreover, after obtaining results, end users can submit the same query again (possibly changing interactively some parameters [16, 27]) after a think time. This implies that the DAG is constant among different executions of a same Tez or Spark application; therefore, we do not need to consider changing DAGs. In other words, we consider a multi-class closed performance model [28] .

3. Overview of SANs

Stochastic Activity Networks (SANs) [18] are stochastic extensions of Petri Nets (PNs) mostly used for modeling and analysis of distributed systems [29, 30, 31]. In comparison with other stochastic generalizations of PNs, e.g. Stochastic Petri Nets (SPNs) and Generalized Stochastic Petri Nets (GSPNs), SANs benefit from more flexibility and tool support. In General, SANs are probabilistic extensions of Activity Networks (ANs) which have been equipped by a set of activity time distribution functions, reactivation predicates and enabling rate functions. In the following, informal description of the basic elements of SANs is presented:

- *Place*: Places are similar to the places in PNs and graphically are represented by circles.

- *Timed activity*: Timed activities are used for modeling actions of the system whose duration affects performance of the system under study noticeably. Graphically, timed activities are represented by thick vertical bars or boxes. Any timed activity can have several inputs and outputs. An input of a timed activity can be a place or an input gate, and similarly an output can be a place or an output gate. An activity distribution

function, an enabling rate function, and a computable predicate called the reactivation predicate are associated to each timed activity.

- *Instantaneous activity*: Instantaneous activities are used for modeling actions of the system which are done in a negligible amount of time compared to the other actions which can be modeled using timed activities. Graphically, instantaneous activities are represented by thin vertical bars. An instantaneous activity can have several inputs and outputs.

- *Input gate*: Gates provide higher flexibility in defining enabling and completion rules. An input gate has a finite set of inputs and one output. A computable predicate called enabling predicate and a computable function called input function are associated to each input gate.

- *Output gate*: An output gate has a finite set of outputs and one input. A computable function called the output function is associated to each output gate.

Above a general overview of the SAN formalism is provided, while the detailed presentation is out of the scope of this paper. A formal definition of SAN formalism, its structure, and behavior are given in [33, 18, 34]. SAN formalism is widely used in other areas of computer science such as cloud computing [29, 30, 31] and Computational Grids [32], for performance evaluation, and is proved to be effective to predict complex IT systems performance.

4. Proposed Models

In this section, the proposed SAN models for the MR, Tez, and Spark applications are presented. To help the better understanding of the proposed models, the SAN model of a MR is presented in subsection 4.1, and then, it is extended to capture a DAG-based Tez or Spark application in subsection 4.2. In order to further improve the scalability of the proposed model for the DAG-based application, the model presented in subsection 4.2 is changed from the monolithic form to the composite form in subsection 4.3.

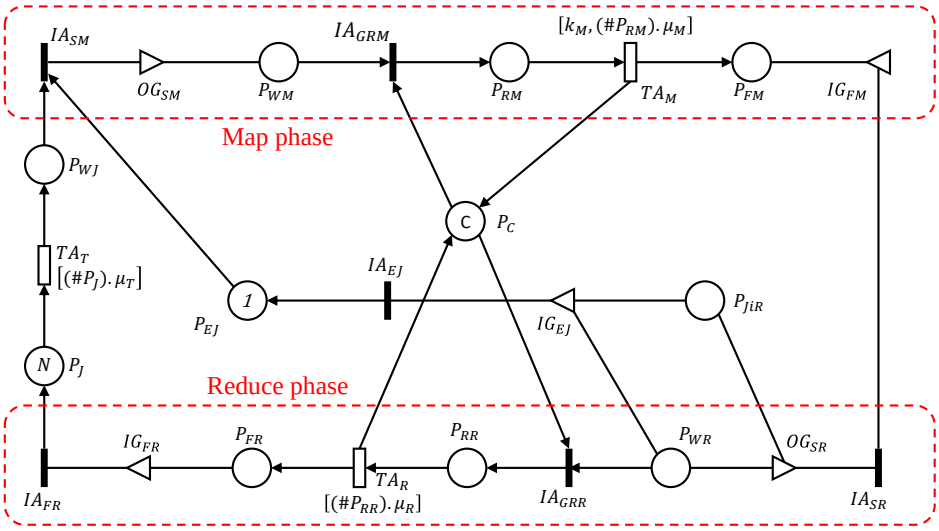


Figure 3: The SAN model proposed for MR applications

4.1. MR Model

The SAN model proposed for MR application is shown in Figure 3. The model includes map and reduce phases, shared resources, think time and the scheduling mechanism. Recalling from section 2 hereafter reduce phase stands for an aggregate of both shuffle and reduce tasks. The detailed description of the places and activities of Figure 3 is provided in Table 1.

Initially, there are N tokens in place P_J showing the jobs waiting to start execution. The timed activity TA_T models the think time of a waiting job. Upon completion of this activity, a token from place P_J is moved to place P_{WJ} with rate $(\#P_J).\mu_T$, where $\#P_J$ is the number of tokens in place P_J showing the waiting jobs, and μ_T is the rate of the exponential distribution considered for activity TA_T. Activity TA_T models the think time and is assumed to be exponentially distributed according to our experiments. Existence of a token in place P_{WJ} triggers instantaneous activity IA_{SM} to start the job if the place P_{EJ} has a token to consume. The place P_{EJ} initially contains a token modeling the possibility of starting a waiting job according to the Capacity scheduler policy. At the start of a job, the output gate OG_{SM} will produce M tokens in place P_{WM}, each one representing a map task. The output function of gate OG_{SM} is given in Table 2. Allocating an available resource

Table 1: Gate predicates/functions of the SAN model represented in Figure 3

Gate	**Predicate**	**Function**
OG_{SM}		`#`P_{WM} `= #`P_{WM} `+ M;`
IG_{FM}	`#`P_{FM} `>= M`	`#`P_{FM} `= #`P_{FM} `- M;`
OG_{SR}		`#`P_{WR} `= #`P_{WR} `+ R;` `#`P_{JiR} `= #`P_{JiR} `+ 1;`
IG_{FR}	`#`P_{FR} `>= R`	`#`P_{FR} `= #`P_{FR} `- R;`
IG_{EJ}	`(#`P_{JiR} `> 0) &&` `(#`P_{WR} `== 0)`	`#`P_{JiR} `= #`P_{JiR} `- 1;`

to a map task is done by instantaneous activity IA_{GRM}, which removes one token from place P_C and one from P_{WM}, and adds a token to place P_{RM}. Place P_C is modeling the pool of containers, which is initially set to contain C tokens representing the total number of containers. The execution of a map task is modeled by the timed activity TA_M, which returns the resource to the pool of available resources whenever a map is done. This activity is characterized by the Erlang distribution with shape k_M and a marking dependent rate $(\#P_{RM}).\mu_M$, where $\#P_{RM}$ is the number of running map tasks and μ_M denotes the execution rate of a single map task. According to our experiments, the exponential distribution is not the case for the map task execution time, and map task execution time fits better with more general distributions like Erlang. On the other hand, for the SAN model to be analytically solvable, all timed activities have to be exponentially distributed [18]. Fortunately, an Erlang distribution can be simulated with a set of continuous exponential activities [35] helping us to use the analytically solvable SAN models, when some actions of the system follow Erlang distribution. Parameters of the distributions are being calculated from the experiment logs.

Once the number of tokens in place P_{FM} reaches the total number of the map tasks, the map phase is finished and the instantaneous activity IA_{SR} starts the reduce phase. The

input gate IG_{FM} consumes M tokens from place P_{FM}, and the output gate OG_{SR} produces R tokens in place P_{WR} representing reduce tasks, whose M and R denote the number of map and reduce tasks, respectively. The input predicate and input function of gate IG_{FM} together with the output function of gate OG_{SR} are shown in Table 2. The completion of this activity IA_{SR} also results in adding a token to place P_{JiR}, which indicates that a job is performing its reduce phase. To the map phase, instantaneous activity IA_{GRR} allocates a free resource to a ready reduce task by removing a token from place P_C and another one from place P_{WR}, and depositing a token into place P_{PR}. The timed activity TA_R models the execution of a reduce task, which moves a token from place P_{RR} to P_{FR} to show that a reduce task is finished. Moreover, a resource is returned to the pool of available resources by completing activity TA_R. The time assigned to the timed activity TA_R modeling the reduce action in the system is assumed to be exponentially distributed with the marking dependent rate $(\#P_{RR}).\mu_R$, where $\#P_{RR}$ and μ_R denote the number of running reduce tasks and the rate of executing a reduce task, respectively. Recalling from section 2, the capacity scheduler implies that the next job can start executing only when the previous job has received all of the necessary resources for completing the reduce phase. Similarly, in our model, the input gate IG_{EJ} enables the instantaneous activity IA_{EJ}, whenever there is a token in place P_{JiR} and there is no token left in place P_{WR}. This condition is checked by the input gate IG_{EJ}. Afterwards, activity IA_{EJ} removes a token from place P_{JiR}, and puts a token into place P_{EJ} enabling instantaneous activity IA_{SM} to start the next job.

4.1.1. Performance Measure

The performance measure we wish to assess by the proposed model of Figure 3 is the steady-state mean execution time of jobs, which is the average time a token needs to move from place P_{WJ} to place P_J. In order to compute the mean execution time, the reward shown in Equation 1 is defined.

$$r = \frac{N}{throughput_{IA_{FR}}} - \frac{1}{\mu_T} \tag{1}$$

Table 2: Elements of the SAN model represented in Figure 3

Name	Description	Rate/Initial no. of tokens		Name	Description	Rate/Initial no. of tokens
P_J	Initial jobs	N		P_{FR}	Finished reduce tasks	0
P_{WJ}	Waiting jobs	0		P_C	Available containers	C
P_{WM}	Waiting map tasks	0		P_{JiR}	Job in reduce phase	0
P_{RM}	Running map tasks	0		P_{EJ}	Enabled subsequent jobs	1
P_{FM}	Finished maps tasks	0		TA_T	Think	$(\#P_J).\mu_T$
P_{WR}	Waiting reduce tasks	0		TA_M	Map	$[k_M\ , (\#P_{RM}).\mu_M]$
P_{RR}	Running reduce tasks	0		TA_R	Reduce	$(\#P_{RR}).\mu_R$

where $throughput_{IA_{FR}}$ is the throughput of the instantaneous activity IA_{FR} and can be calculated by Equation 2.

$$throughput_{IA_{FR}} = \mathbb{P}(\#P_{FR} = R - 1) \cdot \mu_R \tag{2}$$

where $\mathbb{P}(\#P_{FR} = R-1)$ is the probability of being in a state where all but one reduce tasks are finished, so there are $R-1$ tokens in place P_{FR} and one token left to finish the entire job. This probability is multiplied by μ_R, which is the rate of executing a reduce task.

4.2. Monolithic DAG Model

Both Tez and Spark applications consist of multiple phases/stages ordered in a DAG. Therefore, the SAN model of a Tez or Spark application can be achieved by introducing new intermidiate stages to the MR model of Figure 3. Since the DAG differs from one application to another, the DAG model would be application specific. In this paper, for the sake of simplicity, we study the sample DAG represented in Figure 2 (a), and propose a SAN model to evaluate it. The model can be easily extended. The proposed SAN model is shown in Figure 4.

Similar to the SAN modeling a MR application shown in Figure 3, the SAN model of a DAG is also composed of map, reduce and reduce/reduce phases, scheduling mechanism, shared resources, and the think time. Model elements (places, gates, and activities) are

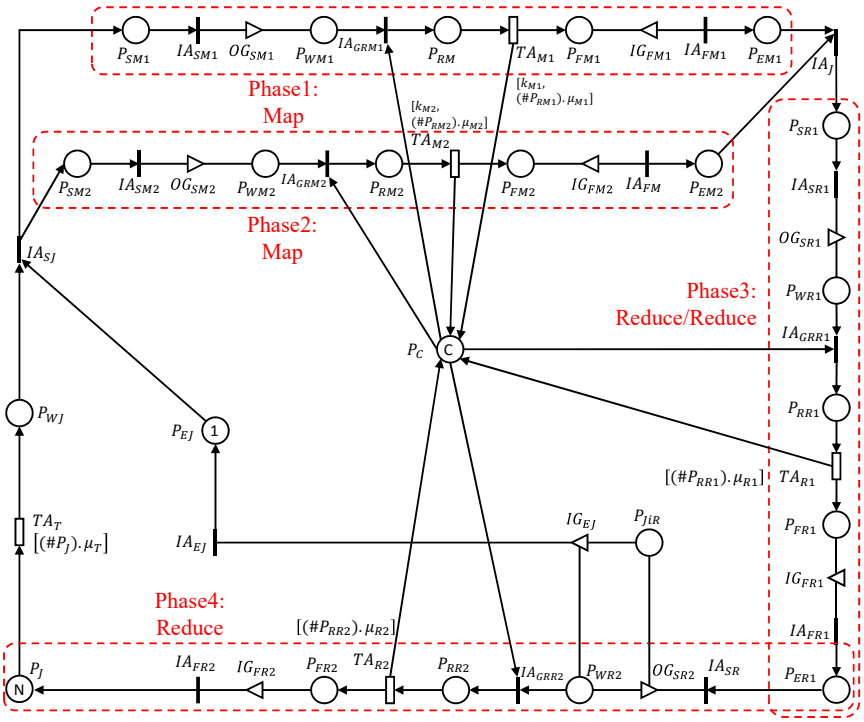


Figure 4: The SAN model proposed for a sample DAG

further numbered in Figure 4 to distinguish between multiple phases. Here, we introduce new instantaneous activities IA_{SJ} and IA_J, which respectively enable parallelism and synchronization among phases. More precisely, the instantaneous activity IA_{SJ} takes a token from each of places P_{WJ} and P_{EJ} and puts a token into both places P_{SM1} and P_{SM2}, which triggers the execution of two map phases in parallel. On the other hand, the instantaneous activity IA_J enables when the previous two parallel map phases are finished and a token is available in each of places P_{EM1} and P_{EM2}. It consumes these two tokens, and puts a token into place P_{SR1}, which starts the execution of the subsequent reduce phase.

Since the model depends on the query graph, a specific model should be built for each query which can cause difficulties especially in modeling complex queries with large graphs tending to be error prone and time consuming. To overcome this difficulty, the composite model is proposed in the next section where similar sub-models are extracted as a single general building block, which can be instantiated multiple times and capture a complete model of a query. With the composite approach, modeling can be automated and gain more

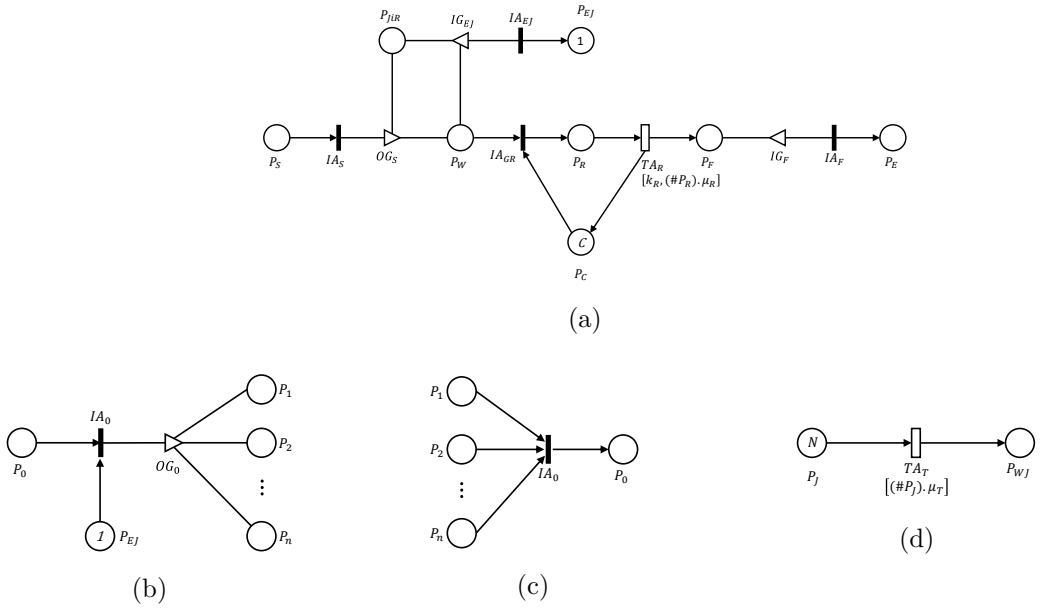


Figure 5: The SAN sub-model for (a) map/reduce phases, (b) parallel fork, (c) synchronization join, and (d) thinking

scalability when the complexity of the query graph grows.

4.3. Composite DAG Model

Looking at the previous models, a structural similarity can be identified among different phases of a MR, or more generally a DAG. For each phase, the model is taking same steps in the same order: generating tasks, allocating resources to them, running tasks, and collecting them at the end. The differences among the phases are in the number of the tasks, and the distribution of the task duration, which are captured as parameters of the model. Therefore, different phases of a DAG model can be regarded as different instances of a general phase sub-model which is represented in Figure 5a, and can be used as a subset of a DAG model. The general phase sub-model of Figure 5a represents the behavior of both map and reduce phases.

In the sub-model shown in Figure 5a, a general phase is modeled, which starts whenever a token is available in P_S, the starting place of the phase. Similar to previous models, instantaneous activity IA_S generates as much token in place P_W as the number of tasks of the current phase. Tokens in place P_W represent the number of tasks waiting for a container

to run, and activity IA_{GR} models the resource allocation from the pool of available containers in place P_C. Running tasks are in place P_R, and the activity TA_R runs tasks with parameters specified for this specific phase. Unlike the MR model described in subsection 4.1, where two different timed activities are considered for map and reduce phases, herein, they are unified in one phase timed activity named TA_R. Since exponential distribution is a specific form of the Erlang distribution, the distribution of activity TA_R is chosen to be Erlang with two parameters shape and rate. For the phases with exponential task execution time, we can simply set the shape parameter to 1. Finished tasks are stored in P_F and whenever all of the phase tasks are finished, activity IA_F removes them from place P_F and inserts a token in place P_E, which indicates that the phase is completed. The scheduling mechanism on top of the model is similar to the previous models and is necessary only for the last phase. Recalling from YARN's Capacity scheduling mechanism, execution of the next job should be informed when the last phase has acquired enough resources to accomplish. A token in place P_{JiR} indicates that a job is performing this phase and activity IA_{EJ} enables whenever a token is in place P_{JiR} and place P_W is empty, which means that all tasks of the current phase have received their containers. After firing IA_{EJ}, a token is put in place P_{EJ}, which triggers the execution of the next job. Gates perform similar to the model of Figure 3.

Sub-models can be connected with sharing places or activities among them [36]. Since sub-models of Figure 5 are asynchronously operating on common resources, we use to share places as the method for connecting sub-models. For example, in order to have a map phase followed by a reduce phase, place P_E of the map sub-model have to be shared with place P_S of the reduce sub-model. Furthermore, place P_C representing available containers is shared between all phases.

For a complete composite model of a DAG, we need a few more sub-models, other than the phase model. A parallel fork is a sub-model that enables the execution of one or more parallel phases as done by instantaneous activity IA_{SJ} in Figure 4. This sub-model is also generalized in Figure 5b, with optional parts drawn with dashed lines. The place P_0 is the start point of the sub-model shown in Figure 5b, which activates instantaneous activity IA_0 when it contains a token. Upon completion of activity IA_0, output places P_1 to P_n,

each will receive a token. Output places are shared with the start places of the subsequent phases. Also, the place P_{EJ} is necessary for the first fork, which is right after the think time, to enable the scheduling mechanism. This happens with sharing P_{EJ} with the same place of the last phase to enable the capacity scheduling to work appropriately.

Another sub-model is a synchronization join shown in Figure 5c., which continues the job execution if the previous phases are accomplished. Again, the input places P_1 to P_n are shared with the output places of the previous phases, and whenever previous phases are accomplished, i.e. a token is available in each of places P_1 to P_n, activity IA_0 is enabled and along with removing a token from each of input places, inserts a token in place P_0. This sub-model is actually modeling a synchronization point where ensures that execution of the next phase starts after finishing all of the previous phases.

The final sub-model is the think time shown in Figure 5d, which is a simple sub-model with places P_J and P_{WJ} shared with P_E of the last phase and P_0 of the Figure 5b sub-model, respectively. Similar to the model of Figure 3, the think activity TA_T is assumed to be exponentially distributed, and moves a token from place P_J to place P_{WJ} upon firing. Place P_{WJ} models the jobs that are ready to start their execution.

Instantiating the same sub-model for different phases and building a composite model using few sub-models, the complexity of the model building is moved to the model instantiation, and thus, the autonomous modeling is facilitated. Figure 6 presents a composite SAN model for the DAG given in Figure 2. In Figure 6, phase instances are drawn in rectangles, and think sub-model is shown by a circle. Moreover, fork and join sub-models are represented by triangles. Sub-models Ph_1 to Ph_4 are different instances of the model represented in Figure 5a named phase model, which are connected directly or through parallel fork and synchronization join, named PJ and SJ, respectively. The think sub-model is also placed at the start of the model. In order to compute the performance measure mentioned in subsubsection 4.1.1, we need to apply Equation 1 and Equation 2 to the last phase of the DAG.

Since Spark's DAG execution engine is similar to Tez from the modeling perspective, we can extend our Tez models to the Spark applications with no specific changes. The phases

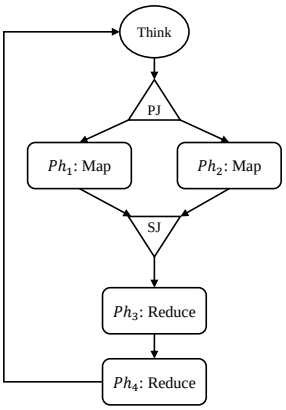


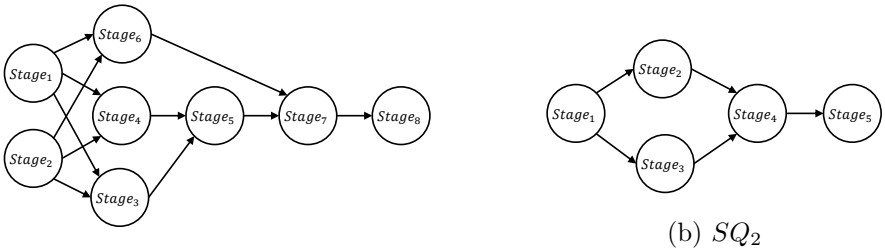
Figure 6: Composite model of the DAG represented in Figure 2

in our Tez model represent the stages in a Spark application and starting tokens represent Spark drivers.

5. Numerical Results

In order to evaluate the accuracy of the proposed models, we have conducted several experiments on different environments ranging from a private cluster to public clouds. More specifically CINECA [21], Flexiant [22], and a private cluster are examined as our test environment. PICO, the Big Data cluster available at CINECA, is composed of 74 nodes, each of them boasting two Intel Xeon 2670v2 2.5GHz 10-core processors, with 128 GB RAM per node. Out of this 74 nodes, up to 66 are available for computation. In our experiments on PICO, we used several configurations ranging from 40 to 120 cores and set up the scheduler to provide one container per core. The Flexiant cluster contains two master nodes and five slave nodes each equipped with four cores and 8GB of memory. Our IBM Power8 (P8) private cluster includes four VMs with 11 cores and 60GB of RAM. Fiber channel disks up to 12TB of storage are available. Spark executors are configured with two cores and four GB of RAM while 8GBof memory are allocated to the driver. The numbers of cores and executors are varied between 6 and 44 and between 3 and 22, respectively.

The dataset used for running the experiments was generated with the TPC-DS benchmark data generator [23], which is the industry standard for benchmarking data warehouses.



(a) SQ_1 (b) SQ_2

Figure 7: Graphs of Spark queries (SQ_1 and SQ_2)

Datasets are in the form of external tables for the Hive [37] queries and their size varies from 250GB to 1TB. Different queries are considered to be executed on datasets as shown in Figure 7 to Figure 9. Queries MQ_1 to MQ_5 represented in Figure 8 are run using MR as Hive underlying engine, while queries TQ_1 to TQ_3 and queries SQ_1 to SQ_2 shown in Figure 9 and Figure 7 are run on Tez and Spark, respectively. Spark queries SQ_1 to SQ_2 are the queries 26 and 52 of the TPC-DS benchmark, respectively; therefore, only their graphs are shown herein for the sake of space limitation. Queries are performed on different dataset sizes to cover a wide range of configurations. For example, in the case of MR queries, the number of map tasks varies between 4 and 1560, while the reduce tasks number varies between 1 and 1009. Moreover, Tez and Spark queries are based on different DAGs, and the number of phases/stages varies from 2 to 8.

For what concerns queries profiling, a pilot execution of the query on a small cluster configuration is considered to properly estimate the parameters of the fitted Erlang distribution for tasks execution time. In particular, the parameters obtained from two configurations (the one corresponding to fewest cores in the considered experiments for each data size) are then used as the model input for all executions of that query across the remaining configurations. This idea of profiling is also used in [16, 17]. Task durations are obtained from execution logs, and on average, each query runs 20 times. According to the information obtained from the above-mentioned experiments, a profile can be created for each query containing the task execution rate for each of its phases, which can be used as input parameters of the proposed SAN models.

```
select avg(ws_quantity), avg(ws_ext_sales_price),
       avg(ws_ext_wholesale_cost), sum(ws_ext_wholesale_cost)
from web_sales
where (web_sales.ws_sales_price between 100.00 and 150.00) or
      (web_sales.ws_net_profit between 100 and 200)
group by ws_web_page_sk
limit 100;
```

(a) MQ_1

```
select inv_item_sk, inv_warehouse_sk
from inventory
where inv_quantity_on_hand > 10
group by inv_item_sk, inv_warehouse_sk
having sum(inv_quantity_on_hand)>20
limit 100;
```

(b) MQ_2

```
select avg(ss_quantity), avg(ss_net_profit)
from store_sales
where ss_quantity > 10 and ss_net_profit > 0
group by ss_store_sk
having avg(ss_quantity) > 20
limit 100;
```

(c) MQ_3

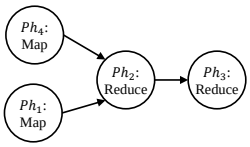
```
select cs_item_sk,
       avg(cs_quantity) as aq
from catalog_sales
where cs_quantity > 2
group by cs_item_sk;
```

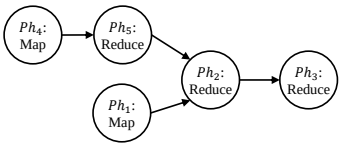
(d) MQ_4

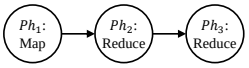
```
select inv_warehouse_sk,
       sum(inv_quantity_on_hand)
from inventory
group by inv_warehouse_sk
having sum(inv_quantity_on_hand) > 5
limit 100;
```

(e) MQ_5

Figure 8: MR queries (MQ_1 to MQ_5)








```
select avg(ss_quantity),
       avg(ss_net_profit)
from store_sales,
     catalog_sales
where cs_bill_customer_sk =
      ss_customer_sk and
      ss_quantity > 10 and
      ss_net_profit > 0
limit 100;
```

(a) TQ_1

```
select a.aq
from (select cs_item_sk,
      avg(cs_quantity) as aq
      from catalog_sales
      where cs_quantity > 2
      group by cs_item_sk) a
  join
     (select i_item_sk,
             i_current_price
      from item
      where
         i_current_price > 2
      order by
          i_current_price) b
  on a.cs_item_sk =
     b.i_item_sk
order by a.aq
limit 100;
```

(b) TQ_2

```
select inv_item_sk,
       inv_warehouse_sk
from inventory
where
 inv_quantity_on_hand > 10
group by inv_item_sk,
         inv_warehouse_sk
having
sum(inv_quantity_on_hand)>20
order by inv_warehouse_sk
limit 100;
```

(c) TQ_3

Figure 9: Tez queries (TQ_1 to TQ_3)

The Mobius tool [36] is used to analytically solve proposed SAN models with iterative steady-state solver. Composite modeling is provided in Mobius tool under hierarchical models. The formalism for composite modeling in Mobius allows modelers to connect sub-models via shared places, which is also used in our proposed model as mentioned in subsection 4.3 for building composite Tez and Spark models.

Results of the proposed models are compared to the SWN model proposed in [13]. This model has been proposed to evaluate the execution time of a MR job via simulation. The GreatSPN 2.0 [38] tool was used to evaluate the SWN model with the same values of accuracy and confidence interval as the SAN model. The results obtained from the experiments, proposed SAN model, and the SWN model in [13] are shown in Table 3 for MR applications.

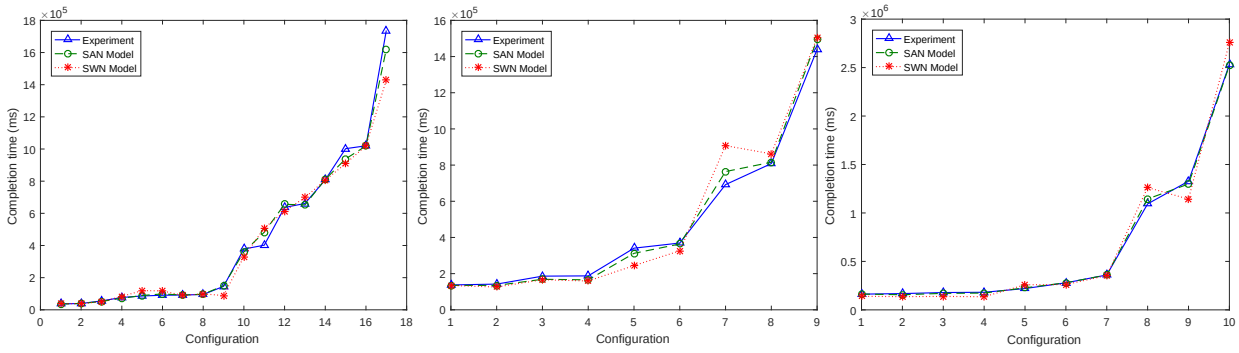


Figure 10: Model validation against experiments and the state of the art model [13] for (a) MR, (b) Tez, and (c) Spark

T denotes the execution time of a MR job measured from the experiment on the system under test, n^M and n^R represent the number of map and reduce tasks respectively, and τ_{SAN} and τ_{SWN} denote the execution times obtained by the proposed SAN model and the SWN model, respectively. The relative error between the SAN model and experiments called θ_{SAN} is computed by Equation 3.

$$\theta_{SAN} = |\frac{\tau_{SAN} - T}{T}| \tag{3}$$

Similarly, the relative error of the SWN model, θ_{SWN}, can be computed by Equation 3 by replacing τ_{SAN} with τ_{SWN}. As can be concluded from Table 3, the average error of the SAN model for estimating MR execution time is 7.1% where this error for the SWN model proposed in [13] is about 12.4% which shows the superiority of our proposed model.

The SAN model accuracy evaluation for Tez applications is reported in Table 4. Here, the SWN model is extended with extra phases in a monolithic way for each of the Tez queries to simulate the graph of that query, and then, the results from the SWN model are compared with the results from analytically solving the proposed SAN model. Column n^{Tasks} denotes the the number of tasks in the successive phases of the reference queries represented in Figure 9.

Table 3: Results obtained from the proposed SAN model represented in Figure 3 and their comparison with the results of SWN model for CINECA system

Query	Users	Cores	Scale [GB]	n^M	n^R	T [ms]	τ_{SAN} [ms]	ϑ_{SAN} [%]	τ_{SWN} [ms]	ϑ_{SWN} [%]
MQ_2	1	240	250	65	5	36 881	35 837	2.83	37 976	2.97
MQ_5	1	80	1 000	64	68	39 206	39 605	1.02	38 796	1.04
MQ_1	1	240	250	500	1	55 410	52 032	6.09	50 629	8.63
MQ_3	1	240	250	750	1	76 806	71 852	6.44	83 317	8.48
MQ_2	3	40	250	4	4	86 023	89 048	3.52	119 712	17.81
MQ_2	5	40	250	4	4	90 674	98 916	9.09	117 582	29.68
MQ_4	1	240	250	524	384	92 141	91 360	0.84	89 426	3.01
MQ_2	3	20	250	4	4	95 403	97 259	1.95	99 219	4.00
MQ_2	5	20	250	4	4	145 646	149 665	2.76	88 683	3.09
MQ_1	1	60	500	287	300	378 127	360 603	4.63	330 149	12.69
MQ_3	1	100	500	757	793	401 827	482 473	20.07	507 758	26.36
MQ_1	5	40	250	144	151	636 694	655 985	3.03	613 577	3.63
MQ_3	1	120	750	1 148	1 009	661 214	652 368	1.33	698 276	5.61
MQ_4	1	60	750	868	910	808 490	811 238	0.34	806 366	0.26
MQ_1	3	20	250	144	151	1 002 160	938 748	6.33	909 217	9.27
MQ_3	1	80	1 000	1 560	1 009	1 019 973	1 018 749	0.12	1 020 294	0.03
MQ_1	5	20	250	144	151	1 736 949	1 620 820	6.69	1 428 894	17.74

Table 4: Results obtained from the proposed composite SAN model for Tez applications and their comparison with the results of SWN model for Flexiant system

Query	Users	Cores	Scale [GB]	n^{Tasks}	T [ms]	τ_{SAN} [ms]	ϑ_{SAN} [%]	τ_{SWN} [ms]	ϑ_{SWN} [%]
TQ_2	1	10	30	<40,6,1,1,1>	137 502	135 192	1.68	134 246	2.37
TQ_3	1	10	30	<30,8,1>	142 071	135 018	4.96	126 154	11.20
TQ_3	1	5	30	<15,8,1>	186 111	168 658	9.38	166 217	10.69
TQ_2	1	5	30	<16,6,1,1,1>	187 628	165 775	11.65	159 383	15.05
TQ_2	1	16	50	<25,10,1,1,1>	340 759	312 276	8.36	244 603	28.22
TQ_3	1	16	50	<25,12,1>	369 215	363 861	1.45	325 938	11.72
TQ_1	1	15	30	<90,20,1,62>	692 141	764 262	10.42	907 535	31.12
TQ_1	1	10	30	<41,20,1,40>	808 359	815 808	0.92	861 997	6.64
TQ_1	1	5	30	<15,20,1,16>	1 439 656	1 495 071	3.85	1 505 288	4.56

According to Table 4, the proposed composite SAN can predict the execution time of Tez applications with an average 6.4% error, which improves the 16.8% average error of the SWN model. Furthermore, there is a significant reduction in Tez modeling time, moving from the monolithic model to the composite model. For example, solving the SAN model for a configuration with 50 users, 1000 tasks, and 300 containers, takes 16s, while the runtime for the SWN model exceeds 350s.

Finally, the accuracy of the proposed model for Spark applications is evaluated in Table 5. Similarly, SWN model is extended with extra stages and its result is compared to the proposed composite SAN model. The average error of 2.7% for the SAN model compared to the 14.4% error of SWN model implies the superior accuracy of the former.

Comparing error values obtained from our proposed models against those obtained from the state-of-the-art machine learning (ML) methods, we can conclude that SAN models perform better than ML-based methods. For example, the error values reported for the ML-based methods proposed in [15] and [16] are 11% and 12%, on average respectively, while it is 6% for the proposed models. SAN and SWN models accuracy across all experiments is summarized in Figure 10, which report the completion time measured on the real systems and estimated by the SAN and SWN models, for different configurations. It is worth mentioning

Table 5: Results obtained from the proposed composite SAN model for Spark applications and their comparison with the results of monolithic SWN model for P8 system

Query	Users	Cores	Scale [GB]	n^{Tasks}	T [ms]	τ_{SAN} [ms]	ϑ_{SAN} [%]	τ_{SWN} [ms]	ϑ_{SWN} [%]
SQ_2	1	32	250	<2,250,2,200,200>	162 232	163 223	0.61	140 180	13.59
SQ_1	1	32	250	<2,2,250,250,200,2,200,200>	168 041	156 211	7.04	136 512	18.76
SQ_1	1	24	250	<2,2,250,250,200,2,200,200>	178 714	170 372	4.67	138 891	22.28
SQ_2	1	24	250	<2,250,2,200,200>	181 496	173 584	4.36	135 536	25.32
SQ_1	1	52	500	<2,2,500,500,500,2,200,200>	220 247	227 526	3.30	258 849	17.53
SQ_2	1	48	750	<2,750,2,200,200>	279 243	279 363	0.04	261 540	6.34
SQ_2	1	48	1000	<2,1000,2,200,200>	359 987	360 096	0.03	354 217	1.60
SQ_1	1	20	500	<2,2,500,500,500,2,200,200>	1 093 593	1 144 559	4.66	1 262 163	15.41
SQ_2	1	10	500	<2,500,2,200,200>	1 327 441	1 298 755	2.16	1 142 832	13.91
SQ_1	1	6	500	<2,2,500,500,500,2,200,200>	2 532 250	2 521 542	0.42	2 757 520	8.90

that configurations in Figures 10a, 10b, and 10c are ordered same as the Table 3, Table 4, and Table 5, respectively.

6. Related Work

In order to evaluate the performance of the MR paradigm and its implementation framework, Hadoop, different approaches have been employed. Some approaches have performed experiments, and then studied historical traces and logs of MR jobs execution in order to assess performance measures and use the insight to predict MR job execution time in future runs [39]. Hadoop execution monitoring is also useful to help the administrator to decide on the configuration parameters and to fine-tune the cluster as proposed in [10]. In [20] for example, authors have proposed the Starfish framework which collects run-time monitoring traces in fine granularity, and uses this detailed job profile to further enhance the configuration and predict job completion time. A sophisticated synthetic workload generator for MR applications over cloud architectures is introduced in [40], which is used to evaluate performance trade-offs. Similarly, the work in [11] used eight benchmarks to evaluate MR performance. The configuration parameters of Hadoop cluster are investigated in [41] and [9] to find the optimal configuration for different types of jobs. In particular, [9] focuses

on optimizing Hadoop parameters by running two copies of the same task with different parameter configurations to empirically identify the best configuration.

Machine learning is another tool used in literature to predict MR performance. Learning techniques from regression [42] to more sophisticated methods, e.g. SVR [15] and KCCA [43], have been used for this regard. Authors in [16] proposed Ernest, to predict Spark job execution time in large scale based on the behavior of the job on small samples of data. Ernest identifies a small set of experiments by relying on optimal experiment design [44] and then applies Non-negative Least Squares (NNLS) to fit the model. The idea of learning a model with small representive experiments has also been used in [17] where multiple polinomial regression models are applied on Spark stages. Stage predictions are then aggregated through the critical path of the execution DAG to estimate the whole job runtime. Combining models with learning methods may improve accuracy. For example, In [43], authors considered static models along with classification techniques to classify current job and estimate its completion time. The authors also compared some clustering and feature elimination techniques, then proposed to use Kernel Canonical Correlation Analysis (KCCA) statistic model to find out the correlation between the features and job execution times. Ataie et al. [15] have combined queueing network model with SVR technique to further increase the accuracy and reduce the number of experiments to be performed on the operational system to train the model.

Simulation as another approach for performance evaluation has been used in several research work [19, 26, 45]. In this regard, great efforts have been made to build a comprehensive simulator for MR and Hadoop [19, 26]. In [26], authors analyzed the application performance on MR cluster through studying the language syntax, logical dataflow, data storage and MR implementations. This simulation is not light weight and may take long time to complete. Ardagna et al. [45] have proposed DAGSim, a novel ad-hoc and fast discrete event simulator, to model the execution of complex DAGs, which can be used to predict Spark application runtime.

In line with simulation approaches, and instead of building simulators, some have introduced simulation models. Petri Nets have been used to study the fault tolerance mecha-

nisms in [46]. Another approach, presented by Barbierato et al. [14], exploits Generalized Stochastic Petri Nets (GSPNs) alongside other formalisms such as process algebras and Markov chains to develop multi-formalism models and capture Hive queries performance behavior. More recently, Ruiz et al. [47] formalized the MR paradigm using Prioritized Timed Colored Petri Nets (PTCPNs). They validated the model and carried out a performance cost trade-off analysis. In [13], queueing network and Stochastic Well-formed Nets (SWN) simulation models have been proposed and validated for MR applications, considering YARN as resource manager. Requeno et al. [48] have proposed a UML profile for Apache Tez and transformed the stereotypes of the profile into Stochastic Petri Nets (SPNs) which can be used as a simulation model. Stochastic Time Petri Nets (STPN) are also among multiple PN-based formalisms which are used to model performance of parallel computing environments [49].

Analytical approaches have also been applied to evaluate the performance of MR paradigm. Analytical models were proposed in [50], for representing a MR system and quantifying the throughput of a particular resource (i.e., network, hardware disk, or CPU). Bardhan and Menascé [51] applied QN models for predicting the completion time of the map phase of MR jobs. Upper and lower bounds were analyticaly derived for MR job execution time in shared Hadoop clusters by authors in [52] SPNs have been used by [12] for performance prediction of adaptive Big Data architecture. Mean Field Analysis was applied by authors in [12] to obtain average performance metrics. In order to cope with the state space explosion problem, authors in [53] used Fluid Petri Nets to simplify the actual model. They proposed fluid models to predict the execution time of the MR and Spark application.

7. Conclusion and Future Work

In this paper, we studied the performance of the main open source Big Data frameworks, i.e., Hadoop, Tez, and Spark. Such frameworks have been widely adopted by the industry and introduce complex software stacks that burden the performance prediction and capacity planning.

Since performance evaluation through experiments is costly and simulation is time-consuming, we have proposed analytical models, which can be generated and solved in a reasonable time, and at the same time, conforms with the real experiments by an acceptable accuracy.

In this paper, we proposed SAN models to analytically evaluate the execution time of MR, Tez, and Spark applications. We cross-validated the accuracy of models, comparing their predictions to the measurements gained from running TPC-DS benchmark over different query and cluster configurations, obtaining across all the experiments an average error of 6%, which is adequate to support capacity planning decisions and what-if analyses [28]. Our proposed models in this paper improve previously presented methods [13] in supporting Tez and Spark frameworks, providing higher accuracy, decreasing model runtime, and supporting larger DAG models.

Future work will extend the models to support multiple simultaneous jobs running different queries on a shared cluster. By considering multiplicity in the query level, scalability problems due to increase in the number of model states arises. Moreover, additional scenarios of interest like execution with faulty nodes, data placement, and speculative execution will be analyzed.

Acknowledgments

The results of this work have been partially funded by the European DICE H2020 research project (grant agreement no. 644869).

[1] M. A. Beyer, D. Laney, The Importance of 'Big Data': A Definition, Tech. rep., Gartner, [Online]. Available: https://www.gartner.com/doc/2057415/importance-big-data-definition. [Accessed on Jul. 2018] (2012). doi:G00235055.

[2] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, A. Hung Byers, Big Data: The next frontier for innovation, competition, and Productivity, 1st Edition, McKinsey Global Institute, 2011.

[3] D. Reinsel, J. Gantz, J. Rydning, Data Age 2025: The Evolution of Data to Life-Critical, [Online]. Available: https://www.seagate.com/de/de/our-story/data-age-2025/. [Accessed on Jul. 2018] (2017).

[4] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Communications of the ACM 51 (1) (2008) 107–113. doi:10.1145/1327452.1327492.

[5] IDC, Worldwide semiannual Big Data and analytics spending guide, [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=IDC_P33195. [Accessed on Jul. 2018].

[6] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, C. Curino, Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications, in: Proceedings of the 2015 ACM International Conference on Management of Data, SIGMOD 2015, ACM Press, Melbourne, Victoria, Australia, 2015, pp. 1357–1369. doi:10.1145/2723372.2742790.

[7] V. K. Vavilapalli, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, Apache Hadoop YARN: Yet Another Resource Negotiator, in: Proceedings of the 4th annual Symposium on Cloud Computing, SOCC 2013, ACM Press, Santa Clara, California, USA, 2013, pp. 1–16. doi:10.1145/2523616.2523633.

[8] Spark, Apache Spark, [Online]. Available: http://spark.apache.org/. [Accessed on Jul. 2018].

[9] S. Babu, Shivnath, Towards automatic optimization of MapReduce programs, in: Proceedings of the 1st ACM symposium on Cloud computing, SoCC 2010, ACM Press, Indianapolis, Indiana, USA, 2010, pp. 137–142. doi:10.1145/1807128.1807150.

[10] J. Dai, J. Huang, S. Huang, B. Huang, Y. Liu, HiTune: dataflow-based performance analysis for big data cloud, in: Proceedings of the USENIX annual technical conference, USENIX Association, Portland, OR, USA, 2011, pp. 87–100.

[11] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis, Evaluating MapReduce for Multi-core and Multiprocessor Systems, in: Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture, IEEE, Scottsdale, AZ, USA, 2007, pp. 13–24. doi:10.1109/HPCA.2007.346181.

[12] A. Castiglione, M. Gribaudo, M. Iacono, F. Palmieri, Exploiting Mean Field Analysis to Model Performances of Big Data Architectures, Future Generation Computer Systems 37 (2014) 203–211. doi:10.1016/j.future.2013.07.016.

[13] D. Ardagna, S. Bernardi, E. Gianniti, S. Karimian Aliabadi, D. Perez-Palacin, J. I. Requeno, Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets, in: Proceedings of the 16th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP 2016, Springer International Publishing, Granada, Spain, 2016, pp. 599–613. doi:10.1007/978-3-319-49583-5_47.

[14] M. Gribaudo, E. Barbierato, M. Iacono, Modeling Apache Hive Based Applications in Big Data Architectures, in: Proceedings of the 7th International Conference on Performance Evaluation Methodologies

and Tools, ValueTools 2013, ICST, Torino, Italy, 2013, pp. 30–38. doi:10.4108/icst.valuetools.2013.254398.

[15] E. Ataie, E. Gianniti, D. Ardagna, A. Movaghar, A Combined Analytical Modeling Machine Learning Approach for Performance Prediction of MapReduce Jobs in Cloud Environment, in: Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2016, IEEE, Timisoara, Romania, 2016, pp. 431–439. doi:10.1109/SYNASC.2016.072.

[16] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, I. Stoica, Ernest: Efficient Performance Prediction for Large-scale Advanced Analytics, in: Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation, USENIX Association, Santa Clara, CA, USA, 2016, pp. 363–378.

[17] G. P. Gibilisco, M. Li, L. Zhang, D. Ardagna, Stage Aware Performance Modeling of DAG Based in Memory Analytic Platforms, in: Proceedings of the 9th International Conference on Cloud Computing, CLOUD 2016, IEEE, San Francisco, CA, USA, 2016, pp. 188–195. doi:10.1109/CLOUD.2016.0034.

[18] J. F. Meyer, A. Movaghar, W. H. Sanders, Stochastic Activity Networks: Structure, Behavior, and Application, in: Proceedings of the International Workshop on Timed Petri Nets, Torino, Italy, 1985, pp. 106–115.

[19] Y. Liu, M. Li, N. K. Alham, S. Hammoud, HSim: A MapReduce simulator in enabling Cloud Computing, Future Generation Computer Systems 29 (1) (2013) 300–308. doi:10.1016/j.future.2011.05.007.

[20] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, S. Babu, Starfish: A Self-tuning System for Big Data Analytics, in: Proceedings of the 5th Biennial Conference on Innovative Data Systems Research, CIDR 2011, Vol. 11, Asilomar, California, USA, 2011, pp. 261–272.

[21] Cineca, Cineca computing center, [Online]. Available: http://www.cineca.it/. [Accessed on Jul. 2018].

[22] Flexiant, Flexiant Cloud Management Software & Cloud Orchestration, [Online]. Available: https://www.flexiant.com/. [Accessed on Jul. 2018].

[23] M. Poess, B. Smith, L. Kollar, P. Larson, TPC-DS, Taking Decision Support Benchmarking to the Next Level, in: Proceedings of the 2002 ACM international conference on Management of data, SIGMOD 2002, ACM Press, Madison, Wisconsin, 2002, pp. 582–587. doi:10.1145/564691.564759.

[24] Flink, Apache Flink, [Online]. Available: https://flink.apache.org/. [Accessed on Jul. 2018].

[25] R. Nachiappan, B. Javadi, R. N. Calheiros, K. M. Matawie, Cloud storage reliability for Big Data applications: A state of the art survey, Network and Computer Applications 97 (2017) 35–47. doi:10.1016/J.JNCA.2017.08.011.

[26] F. Teng, L. Yu, F. Magoulès, SimMapReduce: A Simulator for Modeling MapReduce Framework, in: Proceedings of the Fifth FTRA International Conference on Multimedia and Ubiquitous Engineering,

IEEE, Loutraki, Greece, 2011, pp. 277–282. doi:10.1109/MUE.2011.56.

[27] O. Alipourfard, H. Harry Liu, J. Chen, S. Venkataraman, M. Yu, M. Zhang, CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics, in: Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17), Boston, MA, USA, 2017, pp. 469–482.

[28] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, Quantitative system performance : computer system analysis using queueing network models, 1st Edition, Prentice-Hall, 1984.

[29] E. Ataie, R. Entezari-Maleki, S. E. Etesami, B. Egger, D. Ardagna, A. Movaghar, Power-aware Performance Analysis of Self-adaptive Resource Management in IaaS Clouds, To appear in Future Generation Computer Systems 86 (2018) 134–144. doi:10.1016/J.FUTURE.2018.02.042.

[30] E. Ataie, R. Entezari-Maleki, L. Rashidi, K. S. Trivedi, D. Ardagna, A. Movaghar, Hierarchical Stochastic Models for Performance, Availability, and Power Consumption Analysis of IaaS Clouds, To appear in IEEE Transactions on Cloud Computingdoi:10.1109/TCC.2017.2760836.

[31] R. Entezari-Maleki, L. Sousa, A. Movaghar, Performance and Power Modeling and Evaluation of Virtualized Servers in IaaS Clouds, Information Sciences 394-395 (2017) 106–122. doi:10.1016/J.INS.2017.02.024.

[32] R. Entezari-Maleki, K. S. Trivedi, A. Movaghar, Performability Evaluation of Grid Environments Using Stochastic Reward Nets, IEEE Transactions on Dependable and Secure Computing 12 (2) (2015) 204–216. doi:10.1109/TDSC.2014.2320741.

[33] A. Movaghar, J. F. Meyer, Performability Modeling with Stochastic Activity Networks, in: Proceedings of the 1984 Real-Time Systems Symposium, Austin, TX, USA, 1984, pp. 215–224.

[34] W. H. Sanders, J. F. Meyer, Stochastic Activity Networks: Formal Definitions and Concepts, in: Proceedings of the 1st EEF/Euro Summer School on Formal Methods and Performance Analysis, FMPA 2001, Berg en Dal, Netherlands, 2001, pp. 315–343.

[35] P. Reinecke, L. Bodrog, A. Danilkina, Phase-Type Distributions, Resilience Assessment and Evaluation of Computing Systems (2012) 85–113doi:10.1007/978-3-642-29032-9_5.

[36] T. Courtney, S. Gaonkar, K. Keefe, E. W. D. Rozier, W. H. Sanders, Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models, in: Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, DSN 2009, IEEE, Lisbon, Portugal, 2009, pp. 353–358. doi:10.1109/DSN.2009.5270318.

[37] Hive, Apache Hive, [Online]. Available: https://hive.apache.org/. [Accessed on Jul. 2018].

[38] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, G. Franceschinis, The GreatSPN Tool: Recent Enhancements, ACM SIGMETRICS Performance Evaluation Review 36 (4) (2009) 4–9. doi:10.1145/1530873.1530876.

[39] Z. Zhang, L. Cherkasova, B. T. Loo, Benchmarking approach for designing a mapreduce performance model, in: Proceedings of the ACM/SPEC international conference on International conference on performance engineering, ICPE 2013, ACM Press, Prague, Czech Republic, 2013, pp. 253–258. doi:10.1145/2479871.2479906.

[40] Y. Chen, A. S. Ganapathi, R. Griffith, R. H. Katz, Towards Understanding Cloud Performance Tradeoffs Using Statistical Workload Analysis and Replay, Tech. Rep. UCB/EECS-2010-81, EECS Department, University of California, Berkeley (may 2010).

[41] D. Jiang, B. C. Ooi, L. Shi, S. Wu, The performance of MapReduce: an in-depth study, Proceedings of the VLDB Endowment 3 (1-2) (2010) 472–483. doi:10.14778/1920841.1920903.

[42] N. Yigitbasi, T. L. Willke, G. Liao, D. Epema, Towards Machine Learning-Based Auto-tuning of MapReduce, in: Proceedings of the IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE, San Francisco, CA, USA, 2013, pp. 11–20. doi:10.1109/MASCOTS.2013.9.

[43] A. Ganapathi, Y. Chen, A. Fox, R. Katz, D. Patterson, Statistics-driven workload modeling for the Cloud, in: Proceedings of the IEEE 26th International Conference on Data Engineering Workshops, ICDEW 2010, IEEE, Long Beach, CA, USA, 2010, pp. 87–92. doi:10.1109/ICDEW.2010.5452742.

[44] V. Fedorov, Optimal Experimental Design, Wiley Interdisciplinary Reviews: Computational Statistics 2 (5) (2010) 581–589.

[45] D. Ardagna, E. Barbierato, A. Evangelinou, E. Gianniti, M. Gribaudo, T. B. M. Pinto, A. Guimarães, A. P. Couto da Silva, J. M. Almeida, Performance Prediction of Cloud-Based Big Data Applications, in: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, ACM Press, Berlin, Germany, 2018, pp. 192–199. doi:10.1145/3184407.3184420.

[46] J. E. Marynowski, A. O. Santin, A. R. Pimentel, Method for Testing the Fault Tolerance of MapReduce Frameworks, Computer Networks 86 (2015) 1–13. doi:10.1016/j.comnet.2015.04.009.

[47] M. C. Ruiz, J. Calleja, D. Cazorla, Petri Nets Formalization of Map/Reduce Paradigm to Optimise the Performance-Cost Tradeoff, in: Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, Vol. 3, IEEE, Helsinki, Finland, 2015, pp. 92–99. doi:10.1109/Trustcom.2015.617.

[48] J. I. Requeno, I. Gascón, J. Merseguer, Towards the Performance Analysis of Apache Tez Applications, in: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, ACM Press, Berlin, Germany, 2018, pp. 147–152. doi:10.1145/3185768.3186284.

[49] F. Cicirelli, A. Forestiero, A. Giordano, C. Mastroianni, Parallelization of space-aware applications: Modeling and performance analysis, Network and Computer Applications 122 (2018) 115–127. doi:10.1016/J.JNCA.2018.08.015.

[50] X. Yu, W. Li, Performance modelling and analysis of mapreduce/hadoop workloads, in: Proceedings

of the 21st IEEE International Workshop on Local and Metropolitan Area Networks, IEEE, Beijing, China, 2015, pp. 1–6. doi:10.1109/LANMAN.2015.7114723.

[51] S. Bardhan, D. A. Menascé, Queuing Network Models to Predict the Completion Time of the Map Phase of MapReduce Jobs, in: Proceedings of the 2012 International Computer Measurement Group Conference, CMG 2012, Las Vegas, Nevada, USA, 2012, pp. 1–9.

[52] M. Malekimajd, D. Ardagna, M. Ciavotta, A. M. Rizzi, M. Passacantando, Optimal Map Reduce Job Capacity Allocation in Cloud Systems, ACM SIGMETRICS Performance Evaluation Review 42 (4) (2015) 51–61. doi:10.1145/2788402.2788410.

[53] E. Gianniti, A. M. Rizzi, E. Barbierato, M. Gribaudo, D. Ardagna, Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications, ACM SIGMETRICS Performance Evaluation Review 44 (4) (2017) 23–36. doi:10.1145/3092819.3092824.