This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

Fixed-point Iteration Approach to Spark Scalable Performance Modeling and Evaluation

Soroush Karimian-Aliabadi, Mohammad-Mohsen Aseman-Manzar, Reza Entezari-Maleki, Danilo Ardagna, Bernhard Egger, and Ali Movaghar

Abstract—Companies depend on mining data to grow their business more than ever. To achieve optimal performance of Big Data analytics workloads, a careful configuration of the cluster and the employed software framework is required. The lack of flexible and accurate performance models, however, render this a challenging task. This paper fills this gap by presenting accurate performance prediction models based on Stochastic Activity Networks (SANs). In contrast to existing work, the presented models consider multiple work queues, a critical feature to achieve high accuracy in realistic usage scenarios. We first introduce a monolithic analytical model for a multi-queue YARN cluster running DAG-based Big Data applications that models each queue individually. To overcome the limited scalability of the monolithic model, we then present a fixed-point model that iteratively computes the throughput of a single queue with respect to the rest of the system until a fixed-point is reached. The models are evaluated on a real-world cluster running the widely-used Apache Spark framework and the YARN scheduler. Experiments with the common transaction-based TPC-DS benchmark show that the proposed models achieve an average error of only 5.6% in predicting the execution time of the Spark jobs. The presented models enable businesses to optimize their cluster configuration for a given workload and thus to reduce their expenses and minimize service level agreement (SLA) violations. Makespan minimization and per-stage analysis are examined as representative efforts to further assess the applicability of our proposition.

Index Terms—Apache Spark, Big Data Frameworks, Performance Evaluation, Stochastic Activity Network, State-space Explosion, Approximation Technique, Fixed-point Iteration Method.

1 Introduction

Almost every corner of computer science is touched by the challenges and opportunities of Big Data analytics. A wide variety of fields, from data mining and business intelligence, to the Internet of Things (IoT) and databases are adopting new technologies that are capable of storing and processing huge amount of data [1]. Data volumes growing at an ever increasing pace introduce difficulties as well as new opportunities. These days, over 500 centers around the world are processing COVID-19 test results, looking for patterns and opportunities to find a treatment [2]. Big Data frameworks are, bar none, playing a great role in this industry, and among them Apache Spark [3] is one of the most popular.

Prior to the introduction of Apache Spark, the Hadoop framework [4] was the dominant tool supporting the MapReduce (MR) paradigm for Big Data processing. Both Spark and Hadoop implement their own scheduling mechanisms, however, providers usually employ the more sophisticated YARN scheduler [5]. YARN specializes on cluster resource management and provides two main scheduling algorithms, called the Capacity and the Fair Scheduler.

While these frameworks simplify the task of executing Big Data workloads on a cluster, they also raise the level of abstraction and increase the complexity of the software. This, in turn, leads to a widening configuration space [6], [7]. Since over one third of data intensive businesses are expected to move to the data marketplace by 2024 [2], it is of paramount importance to improve the efficiency and reduce the operating costs of Big Data infrastructures. This has led to a surging interest in breaking down the complexity and understanding the behavior of Big Data workloads. A crucial element to managing and planing Big Data analysis workloads is predicting their performance [8] – both for providers and end-users as both are paying for the resources. The former are able to minimize their total cost of ownership (TCO) and avoid Service Level Agreement (SLA) violations, while the latter can tailor the rented resources to the workload at hand [9].

The many details of the Spark framework, its scheduler, and the other parameters of the cluster require advanced mathematical models to accurately predict the job completion time of a workload. The three main approaches in existing literature on the topic are simulation [10], [11], machine learning [12], [13], [14], [15], and analytical modeling [16], [17], [18]. Efforts to simulate a Spark cluster have resulted in a number of comprehensive simulators that mimic the actual framework's behavior [10]. These tools, however, are often difficult to use and require a lot of simulation time to

- *S. Karimian-Aliabadi, M. M. Aseman-Manzar, and A. Movaghar are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.*
  E-mail: {skarimian, asemanmanzar}@ce.sharif.edu, movaghar@sharif.edu
- *R. Entezari-Maleki is with the School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, and the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.*
  E-mail: entezari@iust.ac.ir
- *D. Ardagna is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Milan, Italy.*
  E-mail: danilo.ardagna@polimi.it
- *B. Egger ✉ is with the Department of Computer Science and Engineering, Seoul National University, Seoul, Korea.*
  E-mail: bernhard@csap.snu.ac.kr (Corresponding author)

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

produce the metrics of interest. Machine learning techniques from regression to neural networks, on the other hand, are effective in estimating the execution time of Spark applications, however, are specifically tuned to a single criterion. Analytical models are capable of offering a greater insight into the system and require less time to produce a result.

Over the past decade, a number of analytical models for Big Data processing have been proposed. Models for MapReduce-based workloads fail to model Spark's more flexible directed acyclic graph (DAG) dependency model and cannot easily be adapted [9], [19], [20]. Models for the Spark architecture exist, however, fail to model important details of the YARN scheduler that have a significant effect on execution time [17], [21], [22]. YARN allows resources to be allocated to a hierarchy of queues that are processed in parallel based on the queue's category or precedence relations. The interference among running YARN queues is not captured by previous performance models. The same multi-tenancy is further supported within the queue itself, and multiple users can simultaneously submit jobs to the different queues. Although considering these details has the potential to increase the accuracy and practicability of the model, this would also introduce serious scalability issues.

In this work, we present an analytical model based on Stochastic Activity Networks (SANs) [23], [24] to predict the execution time of Apache Spark applications running on top of a homogeneous Big Data cluster. We assume that YARN is performing the scheduling task [14] and applications are assigned to multiple parallel YARN queues simultaneously by multiple users. For the first time, our model is considering such interference among applications to reach a realistic performance prediction. A monolithic model is first proposed which reflects the DAG structure of applications, resource allocation, individual queues, and users. Due to serious scalability issues of the monolithic model, we then decompose the model per queue and resolve the dependencies using the fixed-point iterations technique. The existence of a fixed-point is proven mathematically, and the accuracy of the proposed model is assessed through real experiments with the TPC-DS benchmark [25]. The comparison between numerical results obtained from the analytic model and the measurements from real-world experiments show an acceptable average error of 5.6%.

The decomposed model is numerically solvable and thus considerably more scalable than the initial monolithic model. This enables faster decision making for both user and provider sides. To examine this claim, two representative analyses are considered, namely, makespan minimization and per-stage prediction. Results obtained for both scenarios support the applicability of the proposed analytical model in different use cases. In summary, the contributions of this work are:

- To the best of our knowledge, this is the first model that considers YARN queues, user multiplicity, and DAG-based workflows simultaneously. The analytically solvable model is easy to apply and facilitates deployment decisions for both users and providers.
- The presented fixed-point iteration approach is unique in terms of the number of sub-models, the flexibility of the interconnections, and the existence of a proof for convergence.
- Unlike most existing work that rely on simulation, the presented model is validated with results from a real-world cluster.
- The practicality of the proposed model is illustrated by minimizing the makespan of a given set of jobs and per-stage prediction of the execution time.

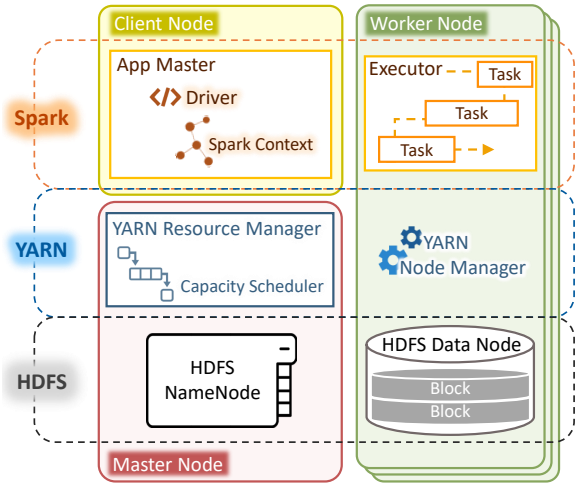

Fig. 1. Target system architecture.

The remainder of this paper is organized as follows. Section 2 describes the architecture of the target system and Section 3 gives a brief introduction to the SAN formalism. Section 4 presents the monolithic form of the proposed model and its scalability issues. These concerns are addressed in the fixed-point iterative approach presented in Section 5. The practicality and accuracy of the model is examined in Section 6. Section 7 discusses related work, and Section 8, finally, concludes this paper.

2 Modeled Target Architecture

This section clarifies the modeled system architecture and briefly describes its components.

2.1 Spark

Spark has emerged as one of the most popular distributed Big Data processing engines [26], particularly thanks to its high speed in-memory calculations. Although Spark is available for standalone usage and is equipped with its own file system, the more popular scenario is the deployment of Spark on top of Hadoop clusters [16]. By doing so, developers can not only take advantage of the mature Hadoop File System (HDFS), but also the comprehensive stack of frameworks intertwined with Hadoop and, most notably, YARN. YARN provides expert means for managing resources, scheduling, and dividing applications into separate queues dedicated to multiple users. In this paper, we assume that Spark is running on top of a Hadoop-YARN cluster with homogeneous resources as depicted in Fig. 1.

A Spark application is mapped to one or more *jobs*. A job is composed of a series of *tasks*, the atomic unit of execution in Spark. One or more tasks are encapsulated in a

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

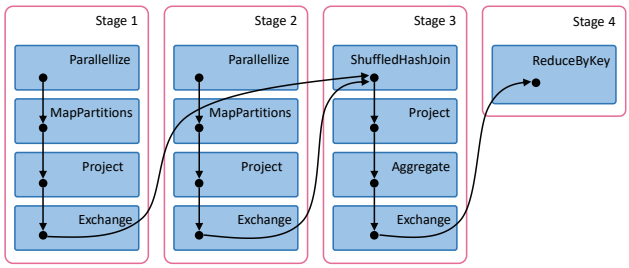

Fig. 2. The DAG including stages for a sample Spark application.

stage which represents the basic unit of parallel execution, i.e., parallelism is achieved by dividing the input data into chunks that are then processed in parallel by multiple identical tasks. Data flow dependencies between the stages, are represented by a Directed Acyclic Graph (DAG). The nodes in the DAG represent the Spark stages, while the edges model data dependencies between them. Fig. 2 shows a visualization of an example Spark application.

The actual execution of a Spark application is orchestrated by a *driver*. The Spark driver encapsulates the *SparkContext*, the object representing the application and responsible for computing the DAG. The driver manages the job flow and schedules tasks on a set of *executors* that run on distributed worker nodes. An executor has a user-configured number of execution threads and amount of memory. One task maps to one thread, i.e., an executor can run multiple tasks simultaneously. The execution schedule and resource management is performed by the cluster manager which we assume to be YARN in this work.

2.2 YARN Capacity Scheduler

The global ResourceManager (RM) is the main component of YARN and manages resources among different applications, no matter whether it is a simple MR or a complex DAG. Each application is assigned to a specific ApplicationMaster (AM) during its life-cycle. There are NodeManagers (NMs) each residing in a node of the cluster to monitor the local resources and to coordinate with the RM. The AM is linked to NMs and the RM to acquire its necessary resources to execute its tasks. The RM has two key sub-components, namely the scheduler and the application manager. The latter keeps track of submitted applications and decides whether to accept another one or to reject the submission. The scheduler, however, is the part that imposes the execution trace and is the subject of a further discussion.

When multi-tenancy was first added to the Hadoop architecture, scheduling was done in first-in-first-out order on a single queue. YARN, on the other hand, supports more flexible scheduling policies implemented by the *Capacity* and the *Fair* schedulers. This work focuses on modeling the Capacity scheduler since it is the more commonly used approach [14].

Instead of a single FIFO queue, the Capacity scheduler allows multiple queues and guarantees a minimum amount of resources for each application. Users can submit their applications to one of the queues according to their own estimate of resource demand. Hierarchical parent-child structures are another feature of the Capacity queues. The parent queue has access to all resources and distributes them to its child queues. This forms a tree structure where the leaves represent the actual computations that are executed with the configuration of their queue. This configuration includes limits for minimal/maximal resource usage relative to the resources of its parent queue. Inside each queue, scheduling is performed in FIFO order and the resources acquired by a user are also controlled by relative upper and lower bounds of the queue's minimum capacity.

To increase overall utilization, available resources are assigned to applications that requested more resources than currently acquired. Under-provisioned applications always have higher priority when assigning free resources. Inside a queue, an application can start executing as soon as the previous application has obtained enough resources to finish its final stage.

2.3 Model Assumptions

The presented models can be deployed in homogeneous Apache Spark clusters (on top of YARN) and can be hosted by an ad-hoc private solution or a public cloud. A key concern here is that our model assumes that the cluster is isolated from the rest of the infrastructure. Therefore, while the proposed model is considering details of inter- and intra-queue relations, it is not aware of inter-cluster interference. The study of such hidden side effects along with resource failures are left to future work and are hereby assumed negligible.

To create a closed queuing network, we assume that user applications are repeatedly submitted to a queue after a short delay called *think time*. This scenario fits real-world execution scenarios well since applications are frequently executed repeatedly after assessing the results of previous runs [27]. Finally, to provide a clearer and easier to understand environment, we assume that each queue is dedicated to a specific application. This assumption is reasonable since one can simulate running different applications in a single queue with multiple queues running each of them.

3 Introduction to SANs

Among the mathematical modeling formalisms employed in computer systems, Petri Nets (PNs) are adopted widely, particularly in evaluating the performance of distributed systems [28], [29]. Petri Nets model a system with a directed bipartite graph. Tokens represent a configuration of the system. Elements of the graph model the flow of the tokens through the system over time. The graphical representation of PNs simplifies the modeling of individual agents and their interconnection and the mathematical background empowers them for explaining probabilistic behaviors. There are a number of stochastic derivations to PNs, among which Stochastic Activity Networks (SANs) have good tool support and are flexible enough to model complex systems compared to other extensions [30], [31], [32], e.g. Stochastic Petri Nets (SPNs) and Generalized Stochastic Petri Nets (GSPNs). SANs that are a probabilistic generalization of Activity Networks (ANs) also benefit from novel features such as activity time distribution functions, reactivation predicates, and enabling rate functions. While a formal definition is given in [24], we include an informal description of the basic elements below.

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

TABLE 1
Summary of key notations.

Notation	Description
C	Total processing cores
Q	YARN queues
μ_T	Rate of exponentially distributed think time
N_i	Users of ith queue
K_i	Stages of the ith queue
S_i	Share of the ith queue as a fraction of C
N_{ij}	Tasks of the jth stage of the ith queue
$\mu_{R_{ij}}$	Rate of exponentially distributed runtime of tasks in the jth stage of the ith queue

- *Places* hold the tokens representing the current system state and are graphically represented by circles.
- *Timed activities* model actions of the system whose duration affects performance of the system. Graphically, timed activities are represented by thick vertical bars or boxes. A timed activity can have several inputs and outputs. An input of a timed activity can be a place or an input gate. 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 with each timed activity.
- *Instantaneous activities* model actions of the system that are instantaneous or consume a negligible amount of time. Graphically, instantaneous activities are represented by thin vertical bars. An instantaneous activity can have several inputs and outputs.
- *Input gates*: 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 the enabling predicate and a computable function called the input function are associated with 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 with each output gate.

The applicability and effectiveness of the SAN formalism covers a wide range of application domains from Cloud Computing [31], to Computational Grids [32], to Computer Networks [30] and has been shown to be effective means to analyze the performance of complex systems [33].

4 Monolithic Model

The monolithic form of the proposed model is presented in this section for the simple scenario of two YARN queues running Spark applications. The rest of the section addresses the definition of the performance metric as well as the scalability issues of the monolithic design.

4.1 SAN Model Description

As stated in Section 2 we have assumed each queue is running a specific application from one or more users; therefore, it is possible to reflect the workflow of the application in the queue itself. Supposing there are two kinds of applications with three and two consecutive stages which are running in two separated queues in parallel by one or more users, the execution model can be proposed as Fig. 3. This model consists of two main parts for each of the queues and a place named P_C which represents the available resources as free cores and is initialized to contain C of them shared among all queues. Inside each queue, place P_{S_i} is the starting point for applications and holds N_i tokens initially equivalent to the number of active users in the queue i and the red dashed boxes indicate stages inside a Spark application. A summary of notations, gate definitions, and elements are given in Table 1, Table 2, and Table 3 respectively.

TABLE 2
Gate predicates/functions of the SAN model represented in Fig. 3.

Name	Feature	Definition
$OG_{S_{ij}}$	Function	$\#P_{W_{ij}} = \#P_{W_{ij}} + N_{ij}$
$IG_{D_{ij}}$	Function	$\#P_{D_{ij}} = \#P_{D_{ij}} - N_{ij}$
	Predicate	$\#P_{D_{ij}} \geq N_{ij}$
IG_{N_i}	Function	$\#P_{L_i} = \#P_{L_i} - 1$
	Predicate	$(\#P_{L_i} > 0)$ && $(\#P_{W_{ik}} = 0)$
$IG_{R_{ij}}$	Function	$\#P_{W_{ij}} = \#P_{W_{ij}} - 1$ $\#P_C = \#P_C - 1$
	Predicate	$(\#P_{W_{ij}} > 0)$ && $(\#P_C > 0)$ && ($(\sum_{j=1}^{K_i} \#P_{R_{ij}} < S_i \cdot C)$ || $for\ each\ queue\ q,\ q \neq i$: $(\sum_{j=1}^{K_q} \#P_{W_{qj}} = 0$ || $\sum_{j=1}^{K_q} \#P_{R_{qj}} \geq S_q \cdot C))$

The SAN sub-model proposed for each stage is the same for all. Every stage starts with populating its tasks which are N_{ij} for the jth stage of the queue i. The instantaneous activity $IA_{S_{ij}}$ and the output gate $OG_{S_{ij}}$ together are responsible for introducing N_{ij} tokens to the place labeled $P_{W_{ij}}$ where tasks wait for acquisition of cores. Afterwards, another instantaneous activity, called $IA_{R_{ij}}$, with the help of an input gate, named $IG_{R_{ij}}$, provides free available cores to the waiting tasks. It wraps the logic for resource allocation performed by the Capacity scheduler and depends on the share of each queue which is denoted by S_i. The detailed behavior of all the input and output gates including their predicates and functions are represented in Table 2. If, according to $IG_{R_{ij}}$, a free core matches a waiting task, $IA_{R_{ij}}$ then removes one token from each of $P_{W_{ij}}$ and P_C places, and adds one to place $P_{R_{ij}}$ where tokens simulate running tasks. The execution time of each task is considered exponentially distributed with rate $\mu_{R_{ij}}$, thus the rate of timed activity $TA_{R_{ij}}$ which models completion of tasks is $\mu_{R_{ij}}$ multiplied by the number of tokens in $P_{R_{ij}}$. Upon completion of a task, $TA_{R_{ij}}$ takes a token from $P_{R_{ij}}$ and inserts a token to $P_{D_{ij}}$ marking it as done. When all tasks of the stage j are completed, i.e. multiplicity of the place $P_{D_{ij}}$ reaches N_{ij}, the stage is finished and $IG_{D_{ij}}$ collects all of the tokens in $P_{D_{ij}}$.

Apart from the elements that are modeling the execution of stages, some others are necessary to complete the

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

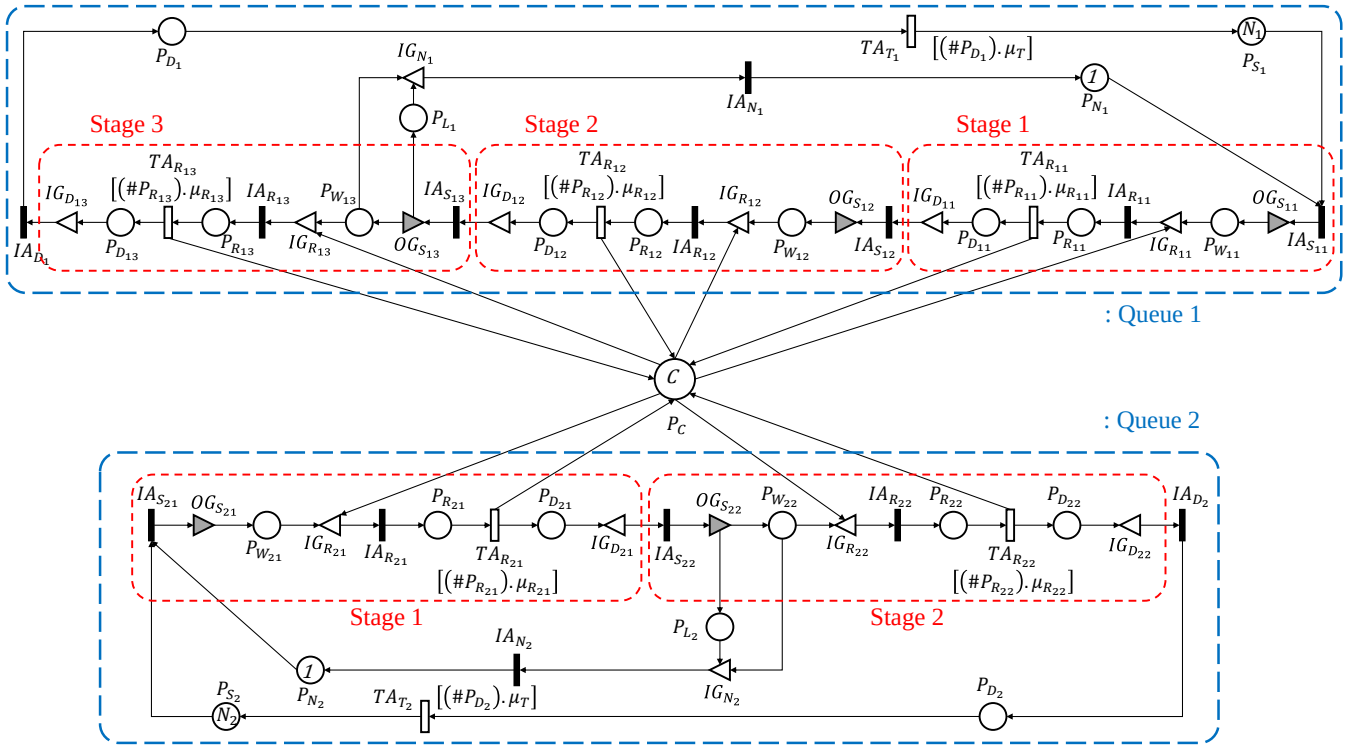

Fig. 3. Monolithic SAN model proposed for a sample scenario of two Capacity queues.

workflow. Just when the last stage finishes, an instantaneous activity dubbed IA_{D_i} puts a token in P_{D_i} determining that the application owned by one of the users in queue i has finished and the user can submit it again after some moments of thinking. This think time is modeled by TA_{T_i}, a timed activity with a marking dependent rate $[(\#P_{D_i}) \cdot \mu_T]$ which is exponentially distributed and moves a token from P_{D_i} to P_{S_i} at every completion.

Recalling from Section 2, the Capacity scheduler only lets an application to start executing in a queue when the last running one is provided with enough cores to finish its finalizing stage. Thus, when $P_{W_{ij}}$ becomes empty, $IA_{S_{ij}}$ had been completed and prompted the starting of the stage some time earlier. Therefore, for the last stage of the ith queue, K_i, the output gate $OG_{S_{ik}}$ adds a token to the place P_{L_i} in addition to its normal function. A token in P_{L_i} means that the last stage is already started. Moreover, IA_{N_i} completes when IG_{N_i} has noticed an empty $P_{W_{ik}}$ and a token in P_{L_i}. This introduces a new token to P_{N_i} which in turn enables the next waiting application to start its first stage via $IA_{S_{i1}}$. As might be expected the marking of the place P_{N_i} is set to one initially.

4.2 Model Output

One of the key characteristics of every stochastic model is the definition of the performance metric for which the model is solved and represents the main concern of the designer. Reward functions are the mechanism devised for some of the formalisms based on Markov Reward Models (MRMs), such as SANs, in order to formulate the performance metric in terms of the state probability distribution. The measure that we wish to assess for the model proposed in Fig. 3 is the steady-state application execution time in each of the queues. This is analogous to the time an application takes to move from place P_{S_i} to the place P_{D_i} in average and is computed using the reward function Eq. 1

$$r_i = \frac{N_i}{throughput_{IA_{D_i}}} - \frac{1}{\mu_T} \tag{1}$$

where, r_i is the reward function for the ith queue and $throughput_{IA_{D_i}}$ is the completion rate of IA_{D_i}, and is computed as Eq. 2.

$$throughput_{IA_{D_i}} = \pi(\#P_{D_{ik}} = N_{ik} - 1) \cdot \mu_{R_{ik}} \tag{2}$$

where $\pi(\#P_{D_{ik}} = N_{ik} - 1)$ is the steady-state probability of being in a state where all but one task of the last stage of the ith queue are finished, so there are $N_{ik} - 1$ tokens in place $P_{D_{ik}}$ and one token left to finish the entire job. This probability is multiplied by $\mu_{R_{ik}}$, the execution rate of a task.

4.3 State-space Explosion

The complexity of the proposed monolithic model is proportional to the number of states of the underlying MRM and the larger its state space become, the more time it will take to solve the model. In the case of the Fig. 3, state space size is reported in Table 4 for different parameter assignments. For the simple configuration of only 2 users in each queue, stages with 32 tasks each, and 20 cores, the state space grows to more than 2M in size which takes more than 42 minutes to generate on a personal computer with an Intel Core i5 2.5 GHz processor, 3 MB of cache, and 6 GB of memory. Provided that in real usage scenarios of Spark applications, stages usually have hundreds of tasks

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

TABLE 3
Elements of the SAN model represented in Fig. 3.

Name	Description	Initial marking	Name	Description	Rate
$P_{D_{ij}}$	Completed tasks	0	$TA_{R_{ij}}$	Task execution time	$[(\#P_{R_{ij}})\cdot\mu_{R_{ij}}]$
$P_{W_{ij}}$	Tasks waiting for resource	0	TA_{T_i}	Thinking time	$[(\#P_{D_i})\cdot\mu_T]$
$P_{R_{ij}}$	Running tasks	0	IA_{D_i}	Completion of an application	
P_{L_i}	Start of the last stage	0	$IA_{R_{ij}}$	Assignment of free cores to waiting tasks	
P_{D_i}	Completed applications	0	IA_{N_i}	Enabling the next application to start	
P_{N_i}	Enabling the next application to start	1	$IA_{S_{ij}}$	Start of the jth stage of ith queue	
P_{S_i}	Users waiting to start their application	N_i			
P_C	Available resources	C			

TABLE 4
State space size of the Fig. 3 for different parameter assignments.

Users in each queue (N_i)	Tasks in each stage (N_{ij})	Cores (C)	State space size	State space generation time (m)
2	40	10	1 333 885	17
2	50	10	1 945 905	18
2	32	20	Out of memory	
3	20	10	939 085	8
3	30	10	2 153 389	39
3	30	20	Out of memory	

and plenty of resources, the monolithic model of Fig. 3 is impractical and its scalability should be improved. In the next section the fixed-point iteration approach is proposed in order to break down the complexity of the model and a composite approximate model is presented instead of the monolithic style.

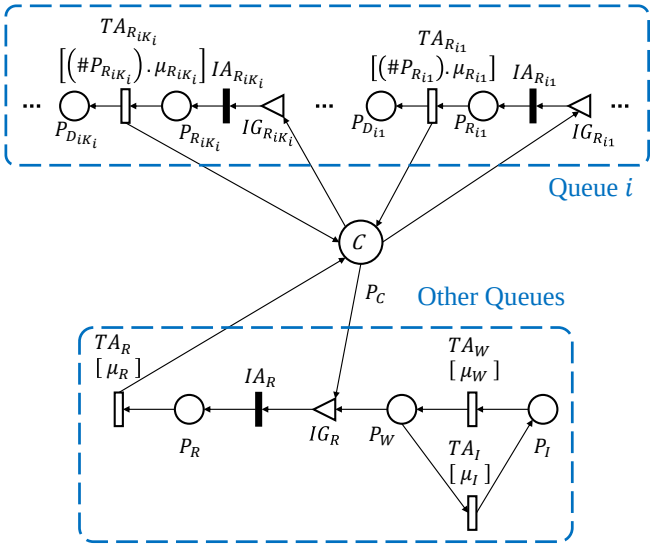

Fig. 4. SAN model proposed for a decomposed queue.

5 Fixed-point Approach

Recalling from Table 4, state space size of the monolithic proposal grows easily with the multiplicity of the parameters and falls short in meeting scalability requirements even for simple scenarios. One of the key reasons is the multiplication of markings in different queue sub-models in order to devise all possibilities of the whole state space. Therefore, knowing that a single queue structure is completely tractable, a decomposition scheme is proposed in this section in which each queue sub-model is being solved individually and the fixed-point theorem [34], [35] is used in order to estimate the final solution. The following section describes the approximate decomposed model as well as the fixed-point iteration approach to solve the proposed models.

5.1 SAN Model Description

Each queue sub-model in the previously proposed monolithic model is decoupled and is presented individually. Thus, instead of a single SAN model for the whole structure, here a system of multiple approximate SAN models are presented, each dedicated to one of the queues. These newly created models are similar in general structure, therefore, a representative example is shown in Fig. 4 which is the decomposed equivalent to the queue i in Fig. 3. The upper part is the exact life-cycle of tasks flowing from stage to stage, while the lower part is an approximate equivalent of resource acquisition/release by the rest of the queues in the system. Some details of the queue i sub-model are hidden to avoid repetition.

It is important for the queue i whether there is any other queue demanding resources or not. In order to model such behavior, two places P_I and P_W are designed which represent the idle and waiting states, respectively. Initially P_I contains one token which means none of the other queues are requesting for cores. At least one of the queues, other than queue i, is asking for resource if a token is encountered in P_W. Timed activities TA_W and TA_I are simulating the transition from idle to waiting state and vice versa, respectively, with relevant rate parameters μ_W and μ_I. The acquisition of resources by other queues is modeled by the input gate IG_R and its appointed instantaneous activity IA_R. It takes a core from P_C and adds to place P_R if there is a token in P_W and if cumulative share of other queues is greater than the current number of acquired resources in P_R. Gate functions and predicates used in the

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

TABLE 5
Gate predicates/functions of the SAN model represented in Fig. 4.

Name	Feature	Definition
IG_R	Function	$\#P_C = \#P_C - 1$
	Predicate	$(\#P_W = 1)$ && $(\#P_C > 0)$ && ($(\#P_R < \sum_{q=1,q\neq i}^{Q} S_q \cdot C)$ || $(\sum_{j=1}^{K_i} \#P_{W_{ij}} = 0))$
$IG_{R_{ij}}$	Function	$\#P_{W_{ij}} = \#P_{W_{ij}} - 1$ $\#P_C = \#P_C - 1$
	Predicate	$(\#P_{W_{ij}} > 0)$ && $(\#P_C > 0)$ && ($(\sum_{j=1}^{K_i} \#P_{R_{ij}} < S_i \cdot C)$ || $(\#P_W = 0))$

TABLE 6
Elements of the SAN model represented in Fig. 4.

Name	Description	Rate/ Initial marking
P_I	All the other queues are in idle state	1
P_W	At least one of the other queues is waiting for resources	0
P_R	Running tasks	0
TA_W	Time from idle to waiting	μ_W
TA_I	Time from waiting to idle	μ_I
TA_R	Task execution time	μ_R
IA_R	Resource allocation	

model of Fig. 4 are defined in Table 5 and a brief description of newly introduced elements is given in Table 6. The timed activity TA_R completes with rate μ_R, upon finishing the tasks and frees resources to P_C.

5.2 Parameter Assignment

SAN model parameter assignment prior to reward computation is another important step in the performance evaluation scheme proposed in this paper. Some parameters of the model are infrastructure related including the number of users in each queue, think time, each queue's capacity, and number of all the available cores which in our proposed SAN model are denoted with N_i, μ_T, S_i, and C respectively. Others are application specific, namely, the DAG of each queue, number of tasks in each stage, and the average time of task execution. Assuming that the DAG is specific to the application and remains constant for different data sizes it can be obtained once the application is submitted and prior to the actual execution of the job since Spark generates the DAG, right after the submission. In order to predict the average task execution time in each stage of the application, and therefore $\mu_{R_{ij}}$, micro-benchmarking approaches can be employed, where the application is tested against a small portion of the actual data and measurements are then extrapolated to estimate that of actual runs. The same approach can be used to realize the number of tasks in each stage, N_{ij}. In this regard, learning-based method developed in [21] is proved practically accurate and is also adopted in this paper.

While assignment of the above-mentioned parameters is feasible, μ_W, μ_I, and μ_R are not known a priori. Rather they can be computed in term of the results obtained from solving individual queue sub-models according to Eq. 3 to Eq. 5.

$$\mu_W = \sum_{q=1,q\neq i}^{Q} \mu_{W_q} \tag{3}$$

$$\mu_R = \sum_{q=1,q\neq i}^{Q} \mu_{R_q} \tag{4}$$

$$\mu_I = \mu_W \cdot \frac{\mathbb{P}_I}{1 - \mathbb{P}_I} \tag{5}$$

where μ_{W_q} is the rate of the transition from idle to waiting state and μ_{R_q} is the rate of task execution, both for the queue q. Rates μ_I and μ_W are proportional to the steady state probability of being in idle or waiting state, therefore, $\mathbb{P}_I$ is introduced in Eq. 5 as the steady state probability of being in idle state which can be calculated by Eq. 6,

$$\mathbb{P}_I = \prod_{q=1,q\neq i}^{Q} \mathbb{P}_{I_q} \tag{6}$$

where $\mathbb{P}_{I_q}$ is the probability of queue q to be in idle state. So far μ_W, μ_R, and μ_I are related to parameters in individual queues for which we have designed reward functions. The reward function for estimating μ_{W_q} is named r_{W_q}. Since each stage waits for resources at its start, Eq. 7 sums the rates of stage activation.

$$r_{W_q} = \sum_{j=1}^{K_q - 1} \Big(\pi(\#P_{D_{qj}} = N_{qj} - 1 \wedge \#P_{R_{qj}} = 1) \cdot \mu_{R_{qj}} \Big) + \pi(\#P_{D_q} > 0) \cdot \mu_T \tag{7}$$

In Eq. 7 for each stage j, $1 \leq j \leq K_q - 1$, the rate of completion of the last task is equivalent to the rate of activating the stage $j + 1$. The first stage is only enabled with rate μ_T if there is any token in the place P_{D_q}. Summing all these terms, the overall rate in which queue q goes to waiting state can be calculated. In order to estimate the value of μ_{R_q} the reward function r_{R_q} is introduced in Eq. 8 where the marking dependent rate of task execution is aggregated for all the stages and $E[\#P_{R_{qj}}]$ denotes the steady state average marking of place $P_{R_{qj}}$.

$$r_{R_q} = \sum_{j=1}^{K_q} E[\#P_{R_{qj}}] \cdot \mu_{R_{qj}} \tag{8}$$

Finally, r_{I_q} is designed for $\mathbb{P}_{I_q}$ estimation in Eq. 9 where the probability that all the tasks of the stage are provisioned or accomplished and none of them is waiting for resources is summed for all stages. Also when jobs are waiting in place P_{D_q} and are not yet started, queue is in idle state.

$$r_{I_q} = \sum_{j=1}^{K_q} \pi(\#P_{R_{qj}} + \#P_{D_{qj}} = N_{qj}) + \pi(\#P_{D_q} > 0) \tag{9}$$

The performance metric designed in Eq. 1 is still applicable in the decomposed model proposed in this section and a

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

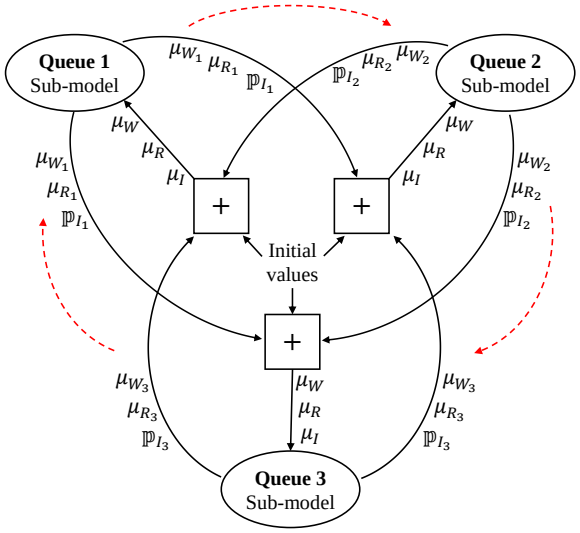

Fig. 5. Interactions among fixed-point sub-models for a sample scenario of three YARN queues.

fixed-point scheme is proposed next in order to circulate parameters μ_{W_q}, μ_{R_q}, and $\mathbb{P}_{I_q}$ among the different sub-models which facilitates obtaining an estimation of this performance metric.

5.3 Fixed-point Iteration

There is a circular dependency between individual queue sub-models and the definite solution for each of them is not possible but a fixed-point iteration scheme can be devised in which each round of model solving increases the accuracy of performance metrics until a threshold is reached. During each iteration the output of one sub-model is used as input to the others while in the first round random initial values are used for circulating parameters. The interactions between the sub-models applying the fixed-point iteration are depicted in Fig. 5 for the sake of clarity and for a sample case including three queues running in parallel where red dashed lines show the order of model solving and central boxes aggregate updated single queue results to compute μ_W, μ_R, and μ_I values. Through successive iterations, performance metric designed in Eq. 1 for each queue sub-model converges to a value which its existence is examined next.

5.4 Fixed-point Existence

Fixed-point variables are depicted in Fig. 5 for a sample case of three queues but for the general case of Q queues the fixed-point equation can be written as below.

$$y = F(y) \tag{10}$$

$$\begin{aligned} y = (&\mu_{W_1}, \mu_{W_2}, ..., \mu_{W_Q}, \\ &\mu_{R_1}, \mu_{R_2}, ..., \mu_{R_Q}, \\ &\mathbb{P}_{I_1}, \mathbb{P}_{I_2}, ..., \mathbb{P}_{I_Q}) \end{aligned} \tag{11}$$

In order to demonstrate the existence of a solution to Eq. 10 we use Brouwers fixed-point theorem [36]:

let $F : \mathcal{C} \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$ be continuous on the compact, convex set $\mathcal{C}$ and suppose that $F(\mathcal{C}) \subseteq \mathcal{C}$. Then, F has a fixed-point in $\mathcal{C}$.

From Eq. 7 we can derive lower and upper bounds of μ_{W_i} values as 0 and $(K-1)\mu_R + \mu_T$, respectively, assuming that $K = \max(K_i), 1 \leq i \leq Q$ is the largest number of stages among all queues and $\mu_R = \max(\mu_{R_{ij}}), 1 \leq i \leq Q, 1 \leq j \leq K_i$ is the greatest task execution rate among all stages of all queues. Similarly, μ_{R_i} is bounded between 0 and $KN\mu_R$ according to Eq. 8, assuming that $N = \max(N_{ij}), 1 \leq i \leq Q, 1 \leq j \leq K_i$ is the biggest number of tasks among all stages of all queues. Finally, knowing that $\mathbb{P}_{I_i}$ is a probability closed in $[0, 1]$, then we can define the set $\mathcal{C}$ as:

$$\begin{aligned} \mathcal{C} = \{y = (&\mu_{W_1}, \mu_{W_2}, ..., \mu_{W_Q}, \\ &\mu_{R_1}, \mu_{R_2}, ..., \mu_{R_Q}, \\ &\mathbb{P}_{I_1}, \mathbb{P}_{I_2}, ..., \mathbb{P}_{I_Q}), \\ \mu_{W_i} \in &[0, (K-1)\mu_R + \mu_T], \\ \mu_{R_i} \in &[0, KN\mu_R], \\ \mathbb{P}_{I_i} \in &[0, 1]\} \end{aligned} \tag{12}$$

According to Heine-Borel theorem, if a subset of the euclidean space $\mathbb{R}^n$ is closed and bounded then it is also compact, therefore, $\mathcal{C}$ defined in Eq. 12 is compact knowing that all of its n-tuples are closed and bounded. Next we show that $\mathcal{C}$ is convex. The set $\mathcal{C}$ is convex if, given two elements $x \in \mathcal{C}$ and $y \in \mathcal{C}$, the element $tx + (1-t)y$, with $t \in [0, 1]$ belongs to $\mathcal{C}$. Since members of $\mathcal{C}$ are n-vectors lets consider the ith component of y, and define $z_i = tx_i + (1-t)y_i$. The maximum value of z_i is obtained when $x_i = y_i = max(y_i)$, so z_i is bounded by $max(y_i)$ and consequently $tx_i + (1-t)y_i \in [0, max(y_i)]$.

A vector function F is continuous over the set $\mathcal{C}$ if its component functions F_i are continuous over $\mathcal{C}$, for each $y \in \mathcal{C}$. That is, for each $\hat{y} \in \mathcal{C}$, $\lim_{y \rightarrow \hat{y}} F_i(y) = F_i(\hat{y})$. The component functions of F are those introduced from Eq. 7 to Eq. 9 for which the limit converges to its (finite) value at $\hat{y}$, and therefore, the component functions are continuous. Hence, the vector function F is continuous over $\mathcal{C}$ and a solution exists for the fixed-point equation of Eq. 10.

6 Analytical and Experimental Results

Extensive experiments are performed in order to asses the accuracy and feasibility of the model proposed in Fig. 5 regarding the performance measure defined in Eq. 1. The results of measurements from real world experiments are compared to those obtained from numerically solving the fixed-point approximate model. Details of the experimental setup and the obtained results are discussed next.

6.1 Model Validation

The errors reported in this paper, denoted by θ, are relative errors computed by Eq. 13.

$$\theta = \left| \frac{\tau - T}{T} \right| \tag{13}$$

where τ stands for the execution time obtained from the proposed SAN model and T is the average of measurements

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

Output: $r_1, r_2, \dots, r_Q$
$\mu_{W_1}, \mu_{W_2}, \dots, \mu_{W_Q} \leftarrow initial\ values$
$\mu_{R_1}, \mu_{R_2}, \dots, \mu_{R_Q} \leftarrow initial\ values$
$\mathbb{P}_{I_1}, \mathbb{P}_{I_2}, \dots, \mathbb{P}_{I_Q} \leftarrow initial\ values$
$r'_1, r'_2, \dots, r'_Q \leftarrow initial\ values$
$converge \leftarrow false$
while $converge = false$ **do**
  $converge \leftarrow true$
  for $q = 1$ **to** Q **do**
    compute $\mu_W, \mu_R, \mathbb{P}_I$ for q**th** queue
    solve the sub-model of q**th** queue
    update $\mu_{W_q}, \mu_{R_q}, \mathbb{P}_{I_q}, r_q$
    if $|r_q - r'_q| / r'_q \geq \delta$ **then**
      $converge \leftarrow false$
    end if
  end for
  $r'_1, r'_2, \dots, r'_Q \leftarrow r_1, r_2, \dots, r_Q$
end while
return $r_1, r_2, \dots, r_Q$

Fig. 6. Fixed-point iteration pseudo-code.

across 10 runs with the same system configuration. Numerical results of the proposed SAN models are acquired using the state-of-the-art tool, Möbius [37], and its iterative steady state solver. In order to successively solve each sub-model of Fig. 5 and automatically pass parameters among them, we developed a script which stops when the difference between τ values of the last iteration and the current iteration falls under a threshold, implying the convergence to the fixed-point. The script pseudo-code is reported in Fig. 6 where δ is the convergence threshold and is set to 0.01 throughout this paper. Experiments show that the fixed-point is reached in no more than 4 iterations, thus the convergence is fast. However, the solving time of the model is reported in Table 7 and Table 8. Numerical analysis is performed on a personal computer with an Intel Core i5 processor and 6 GB of memory.

The workload of experiments is provided with the well-known TPC-DS benchmark which is frequently used in industry and literature for characterizing Big Data systems, and it includes data and query generators. TPC-DS Kit [1] is put into use as an edition of TPC-DS v2.10 with some bug fixes and improvements. The dataset generated for this setup is 500 GB in size and is tested against queries 20, 40, 52, and 84 which are characterized by a variety of DAGs from two to four stages each including hundreds of tasks. Think time between successive executions of jobs is set to $10s$ in all experiments. The profile of each query, namely its DAG, the number of tasks in each stage (N_{ij}), and the average task execution time in each stage ($\mu_{R_{ij}}$) is obtained through 10 time pilot execution of each query. The learning-based technique presented in [21] has been used to accurately predict above-mentioned input parameters for the actual size dataset from micro-executions by dividing stages into two sets, namely, those which scale with increasing in the data size and those which remain more or less constant. More information about the selected queries can be found in the TPC-DS documentation [25] and is omitted here for space limitations. Also for the sake of reproducibility, traces of the experiments, SAN model description, and fixed-point iteration script are available online [2].

Spark 2.4.1 release is installed on top of the Hadoop version 3.1.1 cluster comprising 4 nodes each configured with 10 cores and 25 GB of Memory. The cluster is powered with Intel Xeon E5-2690 v4 processors and DDR4 2400 MHz memories. Each executor container has 2 GB of memory and a single dedicated core while 10 GB of RAM and 5 cores are set aside for the YARN's NodeManager. The cluster is able to consume up to 20 executors in all of the trials. Results are reported in Table 7 and Table 8 for scenarios of 2 and 3 parallel queues respectively. Each queue is tested with different values as its share of the resources and the maximum possible allocation is set to 100%, in case other queues are idle.

According to Table 7 and Table 8 the proposed SAN model along with the fixed-point scheme is able to predict the execution time of Spark jobs on YARN queues with the average error of 5.9% and 5.4% for scenarios of 2 and 3 parallel queues, respectively, which is in line with the expected accuracy in the field of analytical performance modeling [38] and proves that numerical results are accurate enough. While the reported accuracy is an indicator of the effectiveness of the proposed model, capturing the behavior of the target system is another advantage that is achieved, thanks to the power of analytical modeling, and is not easily obtained in learning-based methods. An example is the decrease of the average execution time observed when increasing the number of users in a queue. This is due to the overlapping of two consecutive jobs and is also captured in the numerical results of the SAN model. Moreover, the model demonstrates better accuracy with increasing the number of users, since it results in more samples from the same YARN queue. In scenario of 2 parallel YARN queues, the average error of the proposed model for those with 1, 2, and 3 users is 7.2%, 6.1%, and 2.7%, respectively, revealing an inverse relationship between number of users and the accuracy. A similar trend is found in 3 queues scenario where the average error of 8.2%, 4.1%, and 0.7% is captured for cases with 1 to 3 users. The following sections describe two other possible applications of the presented model.

6.2 Minimizing Makespan

In many use cases, particularly in batch processing, minimizing the makespan is the main concern of the provider. The makespan of a batch of parallel processes is the time that all of the processes are done. Therefore, if processes inside the batch are starting altogether, the makespan is the duration of the longest one. Analogously, the makespan of a group of YARN queues is of a great importance in special scenarios and capacity partitioning among queues is the main factor in this regard. Analytical models can be used to find the most suitable capacity partitioning, and herein we tested the practicality of our proposed fixed-point approximate model for this purpose.

We conduct experiments on a representative scenario of two YARN queues running queries 52 and 84 of the TPC-DS

1. github.com/gregrahn/tpcds-kit

2. github.com/mohsenasm/Spark-Fixed-Point-SAN-Model-Paper

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

TABLE 7
Real and predicted execution times for 2 queues scenario.

#	Queue	N_q	Query	S_q (%)	T (ms)	τ (ms)	θ (%)	Solving time (s)
1	1	2	52	50	51 630	51 305	0.63	30
	2	1	52	50	52 162	52 105	0.11	
2	1	2	52	50	83 046	66 470	19.96	39
	2	2	20	50	52 423	53 650	2.34	
3	1	2	20	40	55 104	55 060	0.08	155
	2	3	40	60	285 880	280 305	1.95	
4	1	1	84	30	29 005	32 172	10.92	73
	2	2	20	70	32 734	32 655	0.24	
5	1	1	52	20	76 281	77 410	1.48	126
	2	3	84	80	26 506	26 342	0.62	
6	1	2	40	50	252 590	289 847	14.75	181
	2	1	40	50	274 719	309 636	12.71	
7	1	2	20	40	37 332	39 639	6.18	61
	2	3	20	60	32 186	34 030	5.73	
8	1	2	84	30	28 055	30 049	7.13	156
	2	2	84	70	27 309	26 198	4.07	
9	1	1	52	20	92 132	80 901	12.19	75
	2	1	40	80	313 521	294 396	6.1	

TABLE 8
Real and predicted execution times for 3 queues scenario.

#	Queue	N_q	Query	S_q (%)	T (ms)	τ (ms)	θ (%)	Solving time (s)
1	1	1	52	20	81 960	97 483	18.94	84
	2	1	20	30	41 063	49 115	19.61	
	3	1	40	50	279 585	286 295	2.4	
2	1	1	40	20	431 389	353 955	17.95	182
	2	2	84	30	49 897	48 969	1.86	
	3	3	52	50	72 317	72 852	0.74	
3	1	2	52	30	109 241	119 663	9.54	60
	2	2	20	30	56 702	54 184	4.44	
	3	2	20	40	49 698	49 445	0.51	
4	1	2	52	30	97 671	110 114	12.74	47
	2	2	20	30	54 797	54 813	0.03	
	3	1	20	40	55 339	52 688	4.79	
5	1	1	40	30	306 785	301 109	1.85	150
	2	1	20	30	58 052	56 496	2.68	
	3	2	84	40	33 182	33 584	1.21	
6	1	1	40	25	321 554	317 181	1.36	113
	2	1	20	25	54 730	57 330	4.75	
	3	3	52	50	79 079	78 557	0.66	
7	1	3	52	33	102 579	103 543	0.94	76
	2	2	52	33	133 755	125 328	6.3	
	3	2	52	34	127 184	126 256	0.73	

benchmark each by a single user and capacity percentages that vary from 10% to 90% for the query 84 and 90% to 10% for the query 52. We compare the results of measurements to the results obtained from analytically solving the model in similar configurations. Results are depicted in Fig. 7 with dashed lines for measurements and solid lines representing predictions. According to the above-mentioned definition, for each horizontal point, the makespan is the maximum of query 52 and query 84 execution time. For example, in case of 50% capacity for each queue, makespan is $max(59s, 33s) = 59s$.

The predicted execution time strictly decreases with the increase in capacity of query 84, and at the same time strictly increases due to decrease in the capacity of query 52. Consequently, the optimal makespan happens at the intersection of two curves. Thus, our approach predicts that 28% of capacity for query 84 and 72% of capacity for query 52 lead to the optimal makespan that is $50s$. Experiments are also displaying almost the same trend for both queues and measurement curves intersect at about the same point (26% and 74% of the capacity for queries 84 and 52, respectively). This implies that the estimation of the execution time obtained from the SAN model of Fig. 5 is useful in finding a proper allocation scheme between YARN queues in order to minimize the overall makespan.

Note that measurements show some fluctuations in marginal configurations. For both queues 52 and 84 (queues dedicated to queries 52 and 84), fluctuations occur at points

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

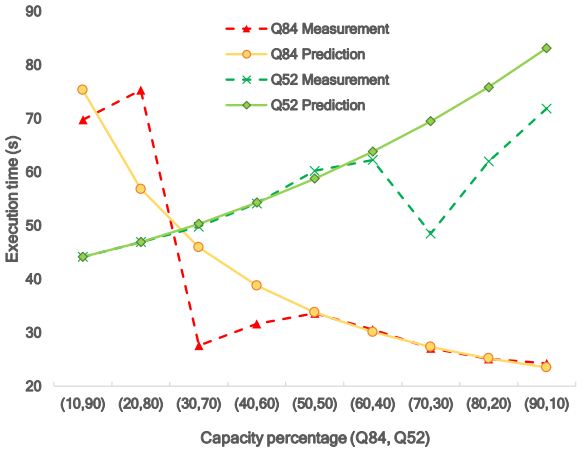

Fig. 7. Makespan analysis for different capacity configurations.

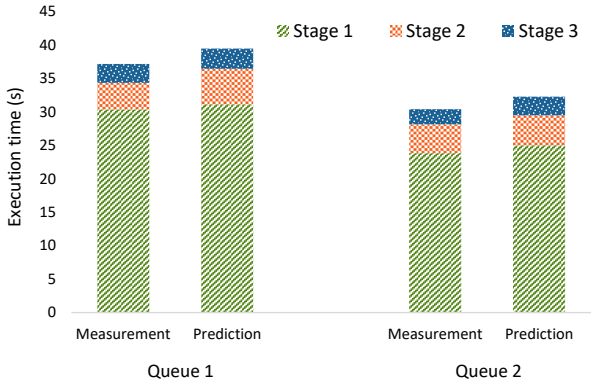

Fig. 8. Real and predicted completion time for each stage of the query 20 in two queues running with 40% and 60% of the capacity (configuration 7 of the Table 7).

where the queue's capacity falls below 40%. With the decrease in the capacity, the number of parallel executors is reduced as well. Therefore, capacity allocations between 10% and 40% are rare, and an insufficient number of measurements are available to devise an accurate approximation. Moreover, system behavior becomes less predictable under conditions when resources are scarce. However, in period during which a reasonable number of executors are available and allocated to the queues, the results of the measurements are very close to those estimated by the proposed model. This supports our statement that the model is practical and can be used to minimize the makespan of jobs submitted to multiple YARN queues.

6.3 Per-stage Analysis

A great power that comes with the SAN formalism is the flexibility in analyzing different performance metrics on a single model. Now that the proposed model has been shown to be accurate, a variety of reward functions can be defined to analyze different performance metrics at once, a capability that can be rarely seen in other modeling approaches.

While the reward function defined in Eq. 1 computes the average steady-state execution time of the whole applications, some users might be interested in the completion time of individual stages, too. Using such an analysis, data scientists can figure out the critical sections of their applications and reduce the whole execution time by only changing specific portions. This can be easily achieved by defining a new reward function on the model of Fig. 3 which computes the average steady-state completion time of an arbitrary stage. The function is defined in Eq. 14.

$$r_{ij} = r_i \cdot \pi(\#P_{D_{ij}} + \#P_{R_{ij}} + \#P_{W_{ij}} > 0) \qquad (14)$$

where, r_{ij} is the reward function to compute steady-state completion time of the jth stage of the ith queue, r_i is the reward function defined in Eq. 1, and $\pi(\#P_{D_{ij}} + \#P_{R_{ij}} + \#P_{W_{ij}} > 0)$ is the steady-state activity probability of the stage, i.e., the probability that the stage is still processing some tasks.

In order to study the behavior of stages using the proposed performance metric, results of the reward function of the Eq. 14 are compared to the measurements of the 7th experiment in Table 7 where two YARN queues are running query 20 simultaneously with 40% and 60% of the resources. Results of this representative scenario are demonstrated in Fig. 8, where it can be seen that the first stage of the application consumes the majority of the total execution time.

7 Related Work

Recently, a notable amount of research is conducted in order to break down the complexity of Big Data processing frameworks, from early ones such as Hadoop to those more recent like Spark. The approaches proposed in the literature to model the performance of Big Data frameworks can be classified mainly as learning-based, analytical, or simulation and their aim is to better tune the configuration parameters, improve resource management, or identify faulty behaviors. Herein, we introduce different viewpoints and discuss their strengths and weaknesses starting from the most recent ones.

Both simulation and analytical approaches are leveraged in [16] in order to predict the execution time of Spark jobs. Moreover, the error of different methods are compared, with regard to sample benchmarks, showing that the simulator is the most accurate with 5.7% error. Although the authors of [16] have presented an analytical model based on queuing networks, simulation is used in order to solve the model instead of analytic-numeric methods. This is also the case in [22] where authors have proposed a model based on process algebra for Spark and solved it using a simulation tool. The proposed formalism, however, is limited to the number of tasks running in parallel and lacks the power to expressive more complex DAGs. Other examples of simulation efforts are [10], [11] which have proposed comprehensive simulators to study a distributed system.

A considerable portion of both analytical and learning-based approaches have reported the employment of sampling and micro-benchmarking. In several researches [20], [39], [40], [41], linear regression of selected sample executions are considered as the predictor for the actual-size performance of a Hadoop application. Based on a similar sampling approach, more sophisticated learning techniques have been adopted such as deep reinforcement learning

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

TABLE 9
Brief comparison of related works.

Works	Approach	DAG	Users	YARN queues	Workload	Average error (%)
Chao et al. [13]	Regression	✓			TeraSort and Kmeans	2.36 - 7.99
Sidhanta et al. [21]	Closed-form analysis	✓			ALS and PageRank	1.2 - 12.6
Chen et al. [18]	Computational geometry				HiBench	4.5
Xu et al. [17]	Closed-form analysis	✓			WordCount and ALS	8.6 - 10.6
Ardagna et al. [16]	Task precedence analytical model	✓			TPC-DS	7.4
Ding et al. [22]	Process algebra					
Fixed-point iterations	**Analytical SAN model**	✓	✓	✓	**TPC-DS**	**5.6**

[12] or combining multiple regression models each for a single stage of the whole application [13]. Regarding the feature selection challenge, authors in [14] have investigated a comprehensive list of features from the application to the underlying infrastructure, and then trained multiple models in order to compare accuracies.

A mathematical formulation has been proposed in [17] for Spark job completion time based on an assumption that stages in independent branches of the DAG are all running in parallel which is not always the case in real systems. The authors of [17] have also ignored the synchronization waiting time between the end of a stage and starting of the next stage. From a geometrical viewpoint, every configuration of YARN cluster parameters can be seen as a point in an n-dimensional coordinate system where interrelation of execution time with parameters can be expressed as a surface. This computational geometry idea was seen through by [18] where authors used sampling in order to construct the mesh. While accuracy of the prediction is acceptable, choosing the proper set of dimensions needs trials and might vary for different applications. A black-box formulation of the execution time in Spark is presented in [21], dividing processing phases into two groups, namely those scaling with the data size and others which remain constant. The former group is then predicted from statistics gathered from sample executions. Authors in [19] targeted the problem of latency prediction in fork-join environments with analytical models and more particularly with closed-form solutions realized for queuing networks. The employment of stochastic formalisms is still popular and in one of the most recent ones, [42], a Petri-net model is proposed to study the performance of stream processing in Spark.
In [43], the lumping technique was used to alleviate the scalability issue of predicting YARN environment, introducing a significant error of about 15%. The contribution was built upon the efforts for developing analytical models for single queue scenarios in [44], [45]. Our findings from related work are summarized in Table 9. We closely examined the state-of-the-art in this area and more precisely those specific to Spark which, to the best of our knowledge, have neither considered multiple queues nor users. Despite its wide adoption in the industry, the details of the YARN scheduling mechanism are also rarely explored in the literature.

8 Conclusions and Future Work

This paper presented analytic-numeric models based on Stochastic Activity Networks to evaluate the performance of Apache Spark jobs running on a YARN cluster. The models consider realistic use cases with multiple YARN queues and users. The limited scalability of the monolithic model is overcome by decomposing the model into approximate sub-models that each represent a single YARN queue. This decomposed model is iteratively evaluated until its parameters reach a fixed-point. We showed that such a fixed-point exists and that the model converges quickly. We evaluated the models by comparing the performance prediction with results obtained on a real cluster for the TPC-DS benchmark suite. The fixed-point model is able to predict the job execution time with an average error of only 5.6% and with a much lower computational overhead compared to the monolithic model. The availability of an accurate, fast, and flexible model enables businesses to employ it as an analysis tool to optimize their cluster configuration, model SLA violations, and thus lower the cost of big data processing.

Analytical models usually take advantage of symmetries, and in our presented model, a homogeneous infrastructure is satisfying this aspect. Although cloud-powered Big Data clusters are mostly supporting this assumption, edge and IoT networks with heterogeneous infrastructures are emerging as new hosts for Big Data applications. Therefore, future research is expected to consider such special environments with proper heuristics and methods. While this work focused on batch data processing, stream processing is another type of workload that is getting more popular nowadays. Stream processing enables real-time analytics for a data stream and is supported by a number of Big Data frameworks. Our work can serve as a basis to implement an analytical SAN model for stream processing where throughput and not execution time is the critical performance metric. Resources are not always available in the real world, and delays due to unavailability, failure, and recovery degrade the overall performance of a cluster. Another possible extension of this work would thus be to add resource failure and recovery to the presented models and to consider multiple clusters at once.

Acknowledgments

This work was supported, in part, by BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU, 21A20151113068) of the Korean National

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

Research Foundation. ICT at Seoul National University provided research facilities for this study. Moreover, the results of this work have been funded by the European Commission under the DICE H2020 research project (grant agreement no. 644869).

REFERENCES

[1] A. Singh, A. Payal, and S. Bharti, "A Walkthrough of the Emerging IoT Paradigm: Visualizing Inside Functionalities, Key Features, and Open Issues," *Journal of Network and Computer Applications*, vol. 143, pp. 111–151, 2019.

[2] Gartner, "Top 10 Trends in Data and Analytics for 2020." [Online]. Available: https://www.gartner.com/smarterwithgartner/gartner-top-10-trends-in-data-and-analytics-for-2020/

[3] Apache, "Spark." [Online]. Available: http://spark.apache.org/

[4] ——, "Hadoop." [Online]. Available: http://hadoop.apache.org/

[5] 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, and H. Shah, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing, SOCC '13*. Santa Clara, California, USA: ACM Press, oct 2013, pp. 1–16.

[6] R. Krishna, C. Tang, K. Sullivan, and B. Ray, "ConEx: Efficient Exploration of Big-Data System Configurations for Better Performance," *to appear in the IEEE Transactions on Software Engineering*, 2019.

[7] L. Cai, Y. Qi, W. Wei, J. Wu, and J. Li, "mrMoulder: A Recommendation-based Adaptive Parameter Tuning Approach for Big Data Processing Platform," *Future Generation Computer Systems*, vol. 93, pp. 570–582, 2019.

[8] L. Bao, C. Wu, X. Bu, N. Ren, and M. Shen, "Performance Modeling and Workflow Scheduling of Microservice-based Applications in Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2101–2116, 2019.

[9] D. Cheng, X. Zhou, Y. Xu, L. Liu, and C. Jiang, "Deadline-Aware MapReduce Job Scheduling with Dynamic Resource Availability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 814–826, 2019.

[10] P. Czarnul, J. Kuchta, M. Matuszek, J. Proficz, P. Rościszewski, M. Wójcik, and J. Szymański, "MERPSYS: An Environment for Simulation of Parallel Application Execution on Large Scale HPC Systems," *Simulation Modelling Practice and Theory*, vol. 77, pp. 124–140, 2017.

[11] Y. Liu, C. Zhang, B. Li, and J. Niu, "DeMS: A Hybrid Scheme of Task Scheduling and Load Balancing in Computing Clusters," *Journal of Network and Computer Applications*, vol. 83, pp. 213–220, 2017.

[12] T.-Y. Mu, A. Al-Fuqaha, and K. Salah, "Automating the Configuration of MapReduce: A Reinforcement Learning Scheme," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–14, 2019.

[13] Z. Chao, S. Shi, H. Gao, J. Luo, and H. Wang, "A Gray-box Performance Model for Apache Spark," *Future Generation Computer Systems*, vol. 89, pp. 58–67, 2018.

[14] Á. B. Hernández, M. S. Perez, S. Gupta, and V. Muntés-Mulero, "Using Machine Learning to Optimize Parallelism in Big Data Applications," *Future Generation Computer Systems*, vol. 86, pp. 1076–1092, 2018.

[15] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, "Towards Accurate Prediction for High-Dimensional and Highly-Variable Cloud Workloads with Deep Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 923–934, 2020.

[16] D. Ardagna, E. Barbierato, E. Gianniti, M. Gribaudo, T. B. Pinto, A. P. da Silva, and J. M. Almeida, "Predicting the Performance of Big Data Applications on the Cloud," *Journal of Supercomputing*, pp. 1–33, 2020.

[17] F. Xu, H. Zheng, H. Jiang, W. Shao, H. Liu, and Z. Zhou, "Cost-effective Cloud Server Provisioning for Predictable Performance of Big Data Analytics," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1036–1051, 2019.

[18] Y. Chen, P. Goetsch, M. A. Hoque, J. Lu, and S. Tarkoma, "d-Simplexed: Adaptive Delaunay Triangulation for Performance Modeling and Prediction on Big Data Analytics," *to appear in the IEEE Transactions on Big Data*, 2019.

[19] M. Nguyen, S. Alesawi, N. Li, H. Che, and H. Jiang, "A Black-box Fork-join Latency Prediction Model for Data-intensive Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 1983–2000, 2020.

[20] Y. Li, F. Liu, Q. Chen, Y. Sheng, M. Zhao, and J. Wang, "MarVeLScaler: A Multi-View Learning based Auto-Scaling System for MapReduce," *to appear in the IEEE Transactions on Cloud Computing*, 2019.

[21] S. Sidhanta, W. Golab, and S. Mukhopadhyay, "Deadline-Aware Cost Optimization for Spark," *to appear in the IEEE Transactions on Big Data*, 2019.

[22] J. Ding, Y. Xie, and M. Zhou, "Performance Modeling of Spark Computing Platform," *Studies in Computational Intelligence*, vol. 810, pp. 121–133, 2020.

[23] J. F. Meyer, A. Movaghar, and 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.

[24] A. Movaghar and 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.

[25] M. Poess, B. Smith, L. Kollar, and P. Larson, "TPC-DS, Taking Decision Support Benchmarking to the Next Level," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data, SIGMOD '02*. Madison, Wisconsin, USA: ACM Press, jun 2002, pp. 582–587.

[26] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A Survey on Spark Ecosystem: Big Data Processing Infrastructure, Machine Learning, and Applications," *to appear in the IEEE Transactions on Knowledge and Data Engineering*, 2020.

[27] H. Zhang, H. Huang, and L. Wang, "Meteor: Optimizing Spark-on-YARN for Short Applications," *Future Generation Computer Systems*, vol. 101, pp. 262–271, 2019.

[28] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, 1st ed. John Wiley & Sons, 1995.

[29] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, 1st ed. Prentice Hall, 1981.

[30] L. Rashidi, R. Entezari-Maleki, D. Chatzopoulos, P. Hui, K. S. Trivedi, and A. Movaghar, "Performance Evaluation of Epidemic Content Retrieval in DTNs with Restricted Mobility," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 701–714, 2019.

[31] E. Ataie, R. Entezari-Maleki, S. E. Etesami, B. Egger, D. Ardagna, and A. Movaghar, "Power-aware Performance Analysis of Self-adaptive Resource Management in IaaS Clouds," *Future Generation Computer Systems*, vol. 86, pp. 134–144, 2018.

[32] R. Entezari-Maleki, M. Bagheri, S. Mehri, and A. Movaghar, "Performance aware Scheduling Considering Resource Availability in Grid Computing," *Engineering with Computers*, vol. 33, no. 2, pp. 191–206, 2017.

[33] R. Entezari-Maleki, Y. Cho, and B. Egger, "Evaluation of memory performance in numa architectures using stochastic reward nets," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 172–188, October 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731520303075

[34] V. Mainkar and K. S. Trivedi, "Sufficient Conditions for Existence of a Fixed point in Stochastic Reward net-based Iterative Models," *IEEE Transactions on Software Engineering*, vol. 22, no. 9, pp. 640–653, 1996.

[35] L. A. Tomek and K. S. Trivedi, "Fixed Point Iteration in Availability Modeling," *M. Dal Cin (Ed.), Informatik-Fachberichte*, vol. 91, pp. 229–240, 1991.

[36] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, 1970.

[37] T. Courtney, S. Gaonkar, K. Keefe, E. W. D. Rozier, and 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)*. Lisbon, Portugal: IEEE, jun 2009, pp. 353–358.

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

[39] M. Alalawi and H. Daly, "Designing a Hadoop MapReduce Performance Model using Micro Benchmarking Approach," in

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

Proceedings of the International Conference on Innovation in Computer Science and Artificial Intelligence, London, UK, jul 2019, pp. 1–11.

[40] Z. Fu and Z. Tang, "Optimizing Speculative Execution in Spark Heterogeneous Environments," *to appear in the IEEE Transactions on Cloud Computing*, 2019.

[41] A. Gandomi, A. Movaghar, M. Reshadi, and A. Khademzadeh, "Designing a MapReduce Performance Model in Distributed Heterogeneous Platforms Based on Benchmarking Approach," *Journal of Supercomputing*, vol. 76, no. 9, pp. 7177–7203, 2020.

[42] R. Tolosana-Calasanz, J. Á. Bañares, and J. M. Colom, "Model-driven Development of Data Intensive Applications Over Cloud Resources," *Future Generation Computer Systems*, vol. 87, pp. 888–909, 2018.

[43] S. Karimian-Aliabadi, D. Ardagna, R. Entezari-Maleki, and A. Movaghar, "Scalable Performance Modeling and Evaluation of MapReduce Applications," in *Proceedings of the Communications in Computer and Information Science*, vol. 891. Tehran, Iran: Springer, apr 2019, pp. 441–458.

[44] S. Karimian-Aliabadi, D. Ardagna, R. Entezari-Maleki, E. Gianniti, and A. Movaghar, "Analytical Composite Performance Models for Big Data Applications," *Journal of Network and Computer Applications*, vol. 142, pp. 63–75, 2019.

[45] D. Ardagna, S. Bernardi, E. Gianniti, S. K. Aliabadi, D. Perez-Palacin, and J. I. Requeno, "Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well formed Nets," in *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*. Granada, Spain: Springer Verlag, dec 2016, pp. 599–613.

Reza Entezari-Maleki received the B.S. and M.S. degrees from the Iran University of Science and Technology, in 2007 and 2009, respectively, and the Ph.D. degree from the Sharif University of Technology, in 2014, all in computer engineering. He worked at the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, as a post-doctoral researcher, from 2015 to 2018. He is currently an assistant professor in the School of Computer Engineering, Iran University of Science and Technology, and an associate researcher at INESC-ID, Instituto Superior Técnico (IST), Universidade de Lisboa, Portugal. He visited the Seoul National University, Seoul, South Korea, in 2012, 2017, and 2019, the Duke University, NC, USA, in 2013, and the Instituto Superior Técnico, Lisbon, Portugal in 2015. His main research interests are performance/dependability modeling and evaluation of distributed computing systems. More information is available at http://webpages.iust.ac.ir/entezari.

Danilo Ardagna received the PhD degree in computer engineering in 2004 from Politecnico di Milano, from which he also graduated in December 2000. He is an associate professor at the Dipartimento di Elettronica Informazione and Bioingegneria, Politecnico di Milano. His work focuses on the design, prototype and evaluation of optimization algorithms for resource management of self-adaptive and cloud systems.

Soroush Karimian-Aliabadi received the BS degree in Computer Engineering from the Tehran University and MS degree in Software Engineering from the Sharif University of Technology. He is currently working toward the PhD degree in Software Engineering at the Sharif University of Technology, under supervision of Prof. Ali Movaghar. In 2016, he joined Prof. Danilo Ardagna's group at Politecnico di Milano as a visiting researcher, where he worked on experimental validation of MapReduce performance model. He is interested in Analytical Performance Modeling, Evaluation, and Prediction of Distributed Systems while his background includes Big Data frameworks, Cloud Computing, and Failure Prediction.

Bernhard Egger received the diploma in computer science from the Swiss Federal Institute of Technology, Zürich (ETHZ) in 2001 and the PhD degree in computer science and engineering from Seoul National University in 2008. He worked on the Samsung Reconfigurable Processor as a senior research engineer at the Samsung Advanced Institute of Technology, Samsung Electronics' research institute, from 2008 to 2011. The same year, he joined Seoul National University as a faculty member where he currently is a professor in the Department of Computer Science and Engineering. Bernhard's research interests include programming language design, compilers, runtimes, and operating systems for parallel and heterogeneous architectures with the goal to maximize resource utilization through the reconciliation of static (compile-time) and dynamic (run-time) information. He is a member of the IEEE and ACM. More information is available at https://csap.snu.ac.kr/.

Mohammad-Mohsen Aseman-Manzar is currently PhD student of Computer Engineering at Sharif University of Technology. He received his MSc. and BSc. in Computer Engineering from Sharif University of Technology and Iran University of Science and Technology (IUST), respectively. His research interests include performance modeling, Big Data frameworks, Cloud Computing, and distributed systems. In his industrial life, he is a full-stack developer and works with various frameworks and languages.

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3119943, IEEE Transactions on Cloud Computing

Ali Movaghar received the BS degree in electrical engineering from the University of Tehran in 1977, and the MS and PhD degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1979 and 1985, respectively. He is a professor in the Department of Computer Engineering at the Sharif University of Technology in Tehran, Iran and has been on the Sharif faculty since 1993. He visited the Institut National de Recherche en Informatique et en Automatique in Paris, France and the Department of Electrical Engineering and Computer Science at the University of California, Irvine, in 1984 and 2011, respectively. He worked at AT&T Information Systems in Naperville, Illinois in 1985-1986, and taught at the University of Michigan, Ann Arbor, in 1987-1989. His research interests include performance/dependability modeling and formal verification of wireless networks, distributed real-time systems, and the Internet of Things. He is a senior member of the IEEE and the ACM.

2168-7161 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Seoul National University. Downloaded on December 31,2021 at 11:16:39 UTC from IEEE Xplore. Restrictions apply.