# Fixed-point Iterations Approach to Spark Scalable Performance Modeling and Evaluation

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

**Abstract**—Businesses are dependent on mining of their Big Data more than ever and configuring clusters and frameworks to reach the best performance is still one of the challenges. An accurate performance prediction of the Big Data application helps reducing costs and SLA-violations with better tuning of the configuration parameters. Among the Big Data frameworks, Apache Spark is the widely used and popular one, with the in-memory processing of graph-based workflows, usually running on top of the YARN cluster. While a great number of attempts have been made to predict the execution time of Spark jobs, to the best of our knowledge, none of them considered multiple simultaneous YARN queues and users in the underlying layer which, by the way, play a great role in the cluster's performance. We first presented a monolithic analytical model based on Stochastic Activity Networks (SANs) for the performance evaluation of Spark applications running inside YARN queues submitted by multiple users. Then we improved the scalability of the proposed model using the fixed-point technique and validated the accuracy of the model through real world experiments on TPC-DS benchmark. We compared results of the proposed model in predicting the Spark job execution time with that of measurements and reported a $5.6\%$ average error for all the queues in experiment setups. While the first monolithic proposal was not feasible to study due to the state-space explosion issue, the fixed-point approach is solvable in reasonable time and thus scalable.

**Index Terms**—Apache Spark, Big Data Frameworks, Analytical Modeling, Performance Evaluation, Fixed-point Method, Stochastic Activity Network.

## 1 Introduction

Almost every corner of the computer science is touched by the challenges and opportunities introduced by Big Data analytics. From Data Mining and Business Intelligence to Internet of Things (IoT) and Databases are adopting new technologies that are capable of storing and processing huge amount of data [1]. Data volumes that are constantly increasing in a pace not seen until today, bring difficulties as well as chances. 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, Hadoop framework [4] was the dominant tool in Big Data era as the perfect implementation of the MapReduce (MR) paradigm [5] and still is continuing as the de-facto standard for the underlying cluster setup. While both Spark and Hadoop have their own scheduling mechanism, providers usually use a more sophisticated layer named YARN [6]. Resource management in the cluster is the specialization of the YARN layer which is also equipped with two main scheduling scheme, namely, Capacity and Fair.


- *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 Department of Computer Engineering, Iran University of Science and Technology, 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*


A considerable portion of data intensive businesses, about $35\%$ of them, are predicted to move to the data marketplace in 2024, up from $25\%$ in 2020 [2]. Thereby, it is of paramount importance to improve efficiency and reduce operating costs in Big Data infrastructure. At the same time, however, each of above-mentioned layers and frameworks are adding complexity to the whole system, widening the state space of different configurations [7], [8]. The surge of interest in breaking down this complexity, understanding its behavior, and predicting it has led to a number of researches. In order to manage a huge process and plan for it we need to be able to predict its performance [9]. Accurate prediction is necessary for both providers and end-users since both are paying for the resources. The former is able to avoid SLA violations and the latter can make budget-aware decisions [10]. In the Cloud environment, in order to avoid over-provisioning, it is sometimes possible to start with the minimum amount of resources and then scale it up on-demand, though this scale-up approach do not apply to MR-based applications [11].

Too many details of the Spark framework along with other parameters of the cluster makes it hard to have a straightforward mathematical model for the job completion time and more professional methods should be employed in order to tackle the complexity. Simulation [12], [13], machine learning [14], [15], [16], [17], and analytical modeling [18], [19], [20] are three main approaches in the literature so far. Efforts in building a Spark simulator have resulted in a number of comprehensive ones which are capable of copying the actual framework's behavior [12] while their utilization is time-consuming and needs an extra phase of mastering the library. Different learning techniques from

regression to neural networks are effective in estimating the execution time of Spark applications, albeit tuned only for a single criterion. Analytical modeling, on the other hand, gives a greater insight into the system and takes less time to solve.

Most of the proposals about performance modeling of Big Data frameworks are either focused on conventional architectures or have ignored important details of the Spark nature. Unlike MR, Spark supports workflows in form of a Directed Acyclic Graph (DAG) making it difficult to adopt conventional models. Moreover, there are some aspects of the YARN planning scheme not addressed in previous works. Resources can be allocated to a hierarchy of queues running in parallel based on the category or precedence. The same level of multi-tenancy is also included inside the queue itself and multiple users are welcome to submit their jobs to the queues at once. Although considering these details might increase the accuracy and practicality of the model, it also introduces serious scalability issues.

To address the challenges above, and in order to include important details, we proposed an analytical model based on Stochastic Activity Network (SAN) formalism [21], [22] for multiple YARN queues running Spark applications submitted by multiple users. Our proposed model is numerically solvable, thus enabling faster decisions. The monolithic form of the model and its scalability problem is first presented and then fixed-point iteration approach is employed to make the model feasible. Capacity planning is our choice of scheduling scheme since its wider adaptation in industry and clusters are assumed homogeneous. The aim of this modeling effort is to accurately predict the average execution time of Spark applications in each YARN queue and this claim is assessed through a set of TPC-DS benchmark [23] executions in a real cluster. Results show that the proposed model is accurate enough with the average error of $5.6\%$ and at the same time scalable with few seconds of solving time.

The rest of the paper is organized as follows. Section 2 illustrates the architecture of the target system. Section 3 gives a brief introduction to the SAN formalism and section 4 presents the monolithic form of the proposed prediction model along with its scalability issues. In order to address these issues, a fixed-point approach is leveraged in section 5 and its applicability is examined in section 6. Section 7 discusses related work and section 8 concludes this paper.

## 2 Target Architecture

Before introducing the performance models, it is necessary to discuss the details in the framework which are going to be reflected in the proposed models. This section clarifies the architecture of the target system along with assumptions about the Spark cluster, scheduler, and the application execution model.

### 2.1 Spark

Spark has emerged as one of the most referred distributed Big Data processing engines [24] which apart from the not so novel parallelism, benefits from its high speed in-memory

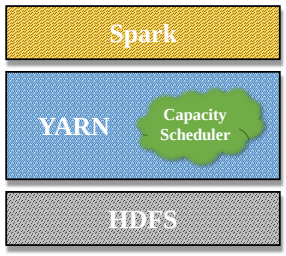


Fig. 1. Target system architecture

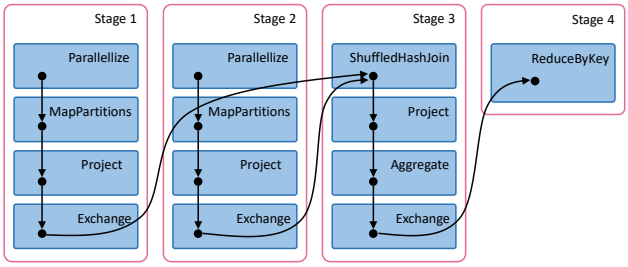


Fig. 2. The DAG including stages for a sample Spark application

calculations. Although Spark is available for standalone usage and is equipped with its own file system, the more popular scenario is the deployment on top of the Hadoop cluster [18]. By doing so, developers can not only take advantage of the mature HDFS, but also the comprehensive stack of frameworks intertwined with Hadoop and most notably, the YARN. YARN provides expert means for managing resources, scheduling, and dividing applications into separate queues of users. It is assumed in this paper that Spark is running on top of the Hadoop-YARN cluster with homogeneous resources as depicted in Fig. 1.

The hierarchical execution style of the Spark framework is a key to handle the complexity together with the parallelism. Applications which are submitted are realized as one or more jobs which some are responsible for the data fetching tasks and some others for doing the actual processing. The data-flow inside each job is represented by a Directed Acyclic Graph (DAG) including vertex and edge concepts. The former stands for a Spark stage and the latter shows the flow of the data from one stage to another. Stages in Spark are meaningful encapsulation of tasks that can run in parallel, knowing that each task is the atomic unit of process. The structure of a sample application is shown in Fig. 2.

The orchestration of concepts described above, however is implemented behind the scene. SparkContext is the main object of the application which holds the code as well as the processes that user has defined. It is wrapped by the Driver which is responsible for collaborating to other nodes and is specific to a single application. The driver needs worker nodes to do the computations and asks for them via the cluster manager. Cluster manager here is assumed to be the YARN but is not limited to. A worker node contains an executer which performs individual tasks. The number of tasks that can run in parallel inside an executer is equivalent to the number of cores configured for the executer. Configuration of each executer, namely, its memory and cores are user-

defined.

Some other details of the application are embedded inside the SparkContext. It will create a job whenever it encounters an action inside the user program and forms the execution DAG afterwards. According to the above-mentioned, this DAG holds stages as well as the tasks residing in them. The SparkContex rationale behind the DAG formation is how the dataset is being transformed from one state to another and relies on an abstraction named Resilient Distributed Dataset (RDD) which is one of the key characteristics of Spark framework. The first RDD is the initial dataset and lazily transforms to other RDDs on-demand, in the sense that all the mappings will be applied at once and right when an action is triggered. Tracking and saving changes on RDDs have enabled Spark to meet the fault tolerance through the lineage. If an RDD is lost eventually, this lineage helps Spark to build that RDD again from scratch.

### 2.2 Capacity Scheduler

When multi-tenancy was added to the Hadoop architecture for the first time FIFO was the scheme for the scheduling but with the development of YARN, it became the expert in scheduling with its comprehensive methods, namely, Capacity and Fair planning. The former option is the more common scheme [16] and easier to study due to the preemptive nature of the latter one. Thus, here we have assumed that the scheduling is being done in Capacity.

Instead of one FIFO queue of submitted applications, Capacity planner allows multiple queues and guaranties minimum amount of resources for each one. Users can submit their applications to one of these queues according to their estimation of resource demand. Hierarchical parent-child structure is another feature of Capacity queues in which there is a root queue by default which accesses all of the resources and distributes them among its sub-queues. This structure ends in leaves where actual applications are running according to the configuration defined for their current queue. This includes minimum and maximum of resources it can consume as percentages of its parent queue. Inside each queue scheduling is done in FIFO and user's acquired resources is also controlled by upper and lower bounds as percentage of queue's minimum capacity.

Free resources can be used by requesting applications in order to increase utilization while under-provisioned ones always have higher priorities to capture free resources. Inside a queue, an application can start just when the previous application has registered for enough resources to finish its final stage.

### 2.3 Execution Model

In order to create a closed performance model, we need to determine a further assumption that every user's application will be submitted again to the queue after a think time. This scenario is not far from real world scenarios where often a single application is executed over and over possibly after assessing its results in previous run [25]. Finally, toward providing a more clear and understandable environment to study, we have considered each queue is dedicated to a specific application. This last assumption again is not odd since one can simulate running different applications in a single queue with multiple queues running each of them.

## 3 Introduction to SAN

Petri Nets (PNs) and their stochastic derivatives are useful in modeling and evaluating computing systems specially distributed ones and among them Stochastic Activity Network (SAN) has a good tool support and is flexible enough to model complex systems compared to other extensions [26], [27], [28], e.g. Stochastic Petri Nets (SPNs) and Generalized Stochastic Petri Nets (GSPNs). SAN which is actually a probabilistic generalization of Activity Networks (ANs) also benefits from novel features like activity time distribution functions, reactivation predicates, and enabling rate functions. While the formal definition is presented in [22], an informal description of its basic elements is given below.

- *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.

Applicability and effectiveness of SAN formalism covers a wide range of expertises including Cloud Computing [27], Computational Grids [28], and Computer Networks [26] and is proved to be effective to analyze complex IT systems performance.

## 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.

TABLE 1
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 | $\#P_{W_{ij}} = \#P_{W_{ij}} - 1$<br>$\#P_C = \#P_C - 1$ |
| | Predicate | $(\#P_{W_{ij}} > 0)$ && $(\#P_C > 0)$ && (<br>$(\sum_{j=1}^{K_i} \#P_{R_{ij}} < S_i \cdot C)$ \|\|<br>$for\ each\ queue\ q,\ q \neq i$ :<br>$(\sum_{j=1}^{K_q} \#P_{W_{qj}} = 0$ \|\|<br>$\sum_{j=1}^{K_q} \#P_{R_{qj}} \geq S_q \cdot C)$<br>) |

### 4.1 SAN model description

As stated in section 2 we have assumed each queue is running a specified 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. It 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.

Red dashed boxes indicate stages inside a Spark application and the SAN sub-model proposed for them is the same for all. Every stage starts with populating its tasks which are $N_{ij}$ for the $j$th 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 $IA_{R_{ij}}$ with the help of an input gate named $IG_{R_{ij}}$ provides free available cores to the waiting tasks. It wraps the logic for the Capacity planning scheme in resource allocation 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 collected in Table 1. 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 $P_C$ 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 that is when 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 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 $i$ is done 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 $[(\#P_{D_i}) \cdot \mu_{T_i}]$ which is exponentially distributed and moves a token from $P_{D_i}$ to $P_{S_i}$ in every firing. 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 $i$.

Recalling from section 2, 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. That is when $P_{W_{ij}}$ becomes empty while $IA_{S_{ij}}$ had been fired and prompted the starting of the stage some time earlier. Thus, for the last stage of the $i$th queue, $K_i$, the output gate $OG_{S_{ik}}$ adds a token to a place named $P_{L_i}$ in addition ot its normal function. A token in $P_{L_i}$ means that the last stage is already started. Moreover, $IA_{N_i}$ fires when $IG_{N_i}$ has noticed an empty $P_{W_{ik}}$ and a token in $P_{L_i}$. This firing introduces a new token to $P_{N_i}$ which in turn enables the next waiting application to fire 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 Performance metric

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 Model (MRM) 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 modeled via the reward function Eq. 1

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

where, $r_i$ is the reward function for the $i$th queue and $throughput_{IA_{D_i}}$ is the rate $IA_{D_i}$ is fired and is computed as following.

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

where $\mathbb{P}(\#P_{D_{ik}} = N_{ik} - 1)$ is the probability of being in a state where all but one task of the last stage of the $i$th 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 rate of execution of a task.

### 4.3 State-space explosion

The proposed model's complexity is proportional to the number of states of the underlying MRM and the larger its state space become, the more time-consuming it would be to solve the model. In the case of the Fig. 3 state space size is reported in Table 3 for different parameter assignments.

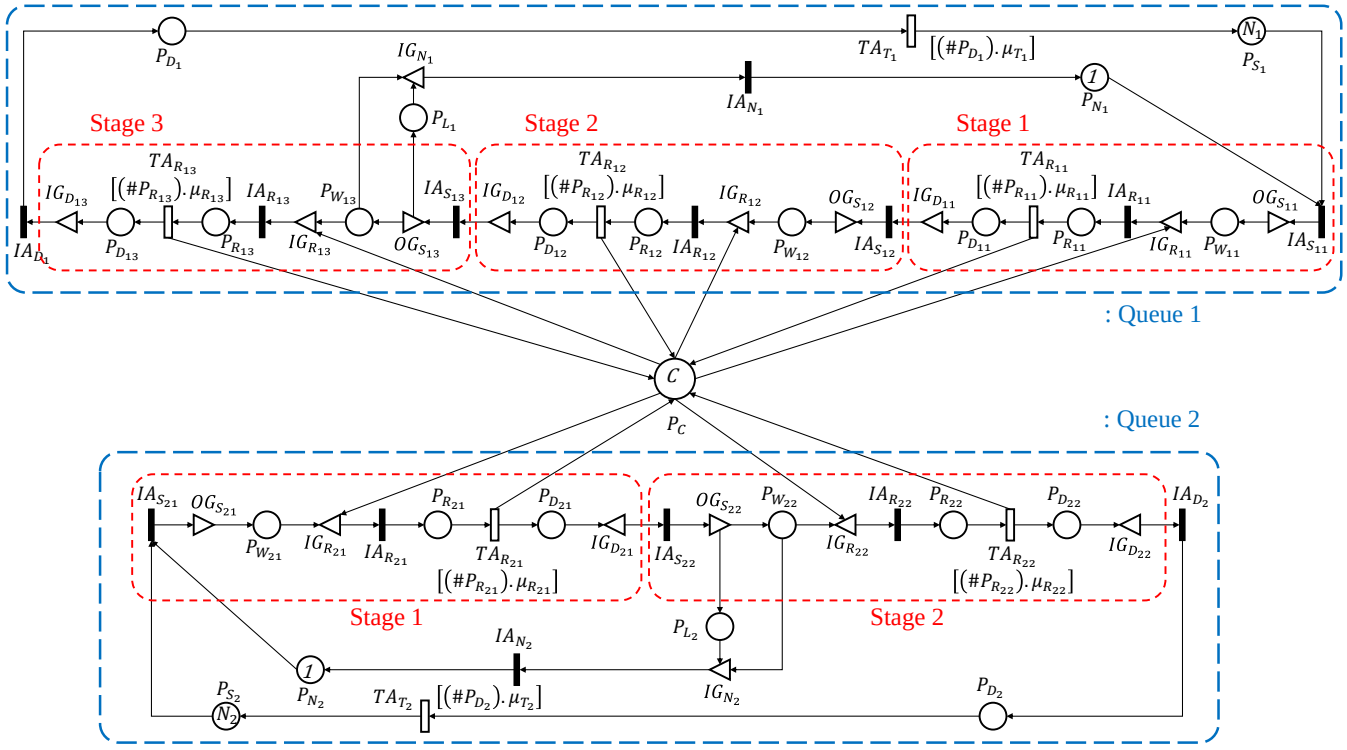


Fig. 3. Monolithic SAN model proposed for a sample scenario of two Capacity queues

TABLE 2
Elements of the SAN model represented in Fig. 3

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

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. Provided that in real usage scenarios of Spark applications, stages usually have hundreds of tasks and plenty of resources, the monolithic model of Fig. 3 is impractical and its scalability should be reconsidered. In the next section the fixed-point iterations approach is proposed in order to break down the complexity of the model and a composite model is presented instead of the monolithic style.

TABLE 3
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 | $>$ 2 154 000 | $>$ 42 |
| 3 | 20 | 10 | 939 085 | 8 |
| 3 | 30 | 10 | 2 153 389 | 39 |
| 3 | 30 | 20 | $>$ 2 154 000 | $>$ 29 |

## 5 Fixed-point Approach

Recalling from the Table 3 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 com-

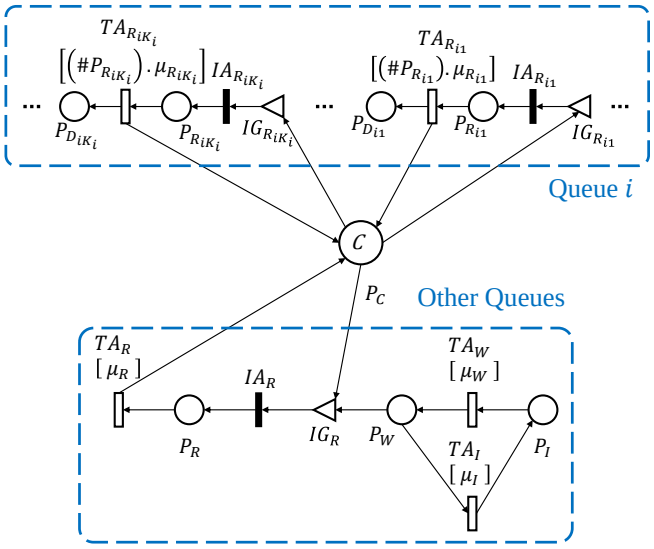


Fig. 4. SAN model proposed for a decomposed queue

TABLE 4
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 | $\#P_{W_{ij}} = \#P_{W_{ij}} - 1$<br>$\#P_C = \#P_C - 1$ |
| | Predicate | $(\#P_{W_{ij}} > 0)$ && $(\#P_C > 0)$ && ( $(\sum_{j=1}^{K_i} \#P_{R_{ij}} < S_i \cdot C)$ \|\| $(\#P_W = 0)$ ) |

pletely tractable, a fragmentation scheme is presented in this section in which each queue sub model is being solved individually and fixed-point theorem [29], [30] is used in order to estimate the final solution. Following describes the decomposed model as well as the fixed-point approach to solving 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 SAN models are represented 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 decomposed equivalent of the queue $i$ in Fig. 3.

Some details of the queue $i$ sub-model are hidden to avoid repetition. In the lower sub-model the behavior of all the other queues that are running in parallel is simplified and estimated by a SAN mechanism which simulates

TABLE 5
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 | $\mu_W$ |
| $TA_I$ | Time from waiting to idle | $\mu_I$ |
| $TA_R$ | Task execution time | $\mu_R$ |
| $IA_R$ | Resource allocation | |

acquiring and returning cores by them as well as their requests. It is important for the queue $i$ whether there is any other queue demanding resources. To model such behavior, two places $P_I$ and $P_W$ are designed which represent the idle and waiting state 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 then 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 $\mu_W$ and $\mu_I$. 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 model of Fig. 4 are collected in Table 4 and a brief description of newly introduced elements is given in Table 5. The exponentially distributed timed activity $TA_R$ with rate $\mu_R$ fires upon completion of tasks and frees resources to $P_C$.

### 5.2 Parameter assignment

SAN model parameter assignment prior to reward computation is another important step in 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 formalism are denoted with $N_i$, $\mu_T$, $S_i$, 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 estimate 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 run. The same approach can be used to realize number of the tasks in each stage, $N_{ij}$. In this regard, learning-based method

developed in [31] is proved practically accurate and is also adopted in this paper.

While assignment of the above-mentioned parameters is feasible, $\mu_W$, $\mu_I$, and $\mu_R$ 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 $\mu_{W_q}$ is the rate of the transition from idle to waiting state and $\mu_{R_q}$ is the rate of task execution, both for the queue $q$. Rates $\mu_I$ and $\mu_W$ 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 $q$ in idle state. So far $\mu_W$, $\mu_R$, and $\mu_I$ are related to parameters in individual queues for which we have designed reward functions.

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

The reward function for estimating $\mu_{W_q}$ is named $r_{W_q}$. In Eq. 7 for each stage $j$ except the last one the probability that one of the tasks is running and all the others are finished is multiplied by the rate of task execution which indicates the rate that stage $j$ enables its next stage. The last stage is not going to enable another stage and the first stage is only enabled with rate $\mu_{T_q}$ 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 $\mu_{R_q}$ reward function $r_{R_q}$ is proposed in Eq. 8 where the marking dependent rate of task execution is aggregated for all the stages and $\pi(P_{R_{qj}})$ denotes the steady state marking of place $P_{R_{qj}}$.

$$r_{R_q} = \sum_{j=1}^{K_q} \pi(P_{R_{qj}}) \cdot \mu_{R_{qj}} \tag{8}$$

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.

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

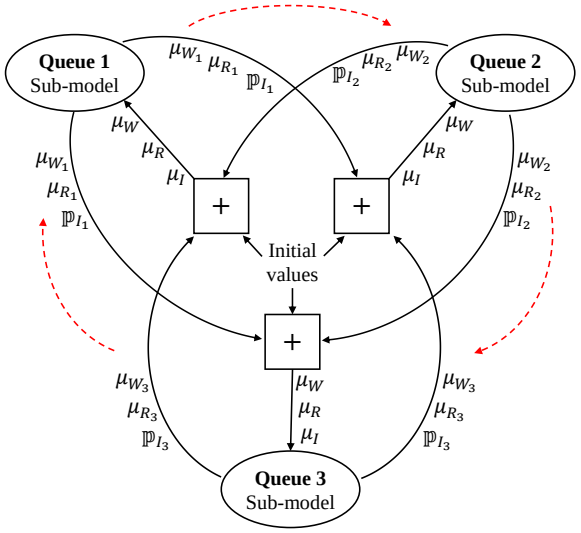


Fig. 5. Fixed-point iterations depicted for a sample scenario of three YARN queues

The performance metric designed in Eq. 1 is still applicable in the decomposed model proposed in this section and a fixed-point scheme is proposed next in order to circulate parameters $\mu_{W_q}$, $\mu_{R_q}$, and $\mathbb{P}_{I_q}$ between sub-models which facilitates realizing an estimation of this metric.

### 5.3 Fixed-point iterations

There is a circular dependency between individual queue sub-models and definite solution of each of them is not possible but a fixed-point 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 input to others while in the first round initial values are used for circulating parameters. The fixed-point scheme is depicted in Fig. 5 for the sake of clarity and for a sample case of three queues running in parallel where red dashed line is showing the order of model solving and central boxes are aggregating updated single queue results to compute $\mu_W$, $\mu_R$, and $\mu_I$ 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.

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

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

In order to demonstrate existence of a solution to Eq. 10 we use Brouwers fixed point theorem [32]:

*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 $\mu_{W_i}$ 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, $\mu_{R_i}$ 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 $i$th component of $y$, and define $z_i = tx_i + (1-t)y_i$. maximum value of $z_i$ happens 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 $\hat{y} \in \mathcal{C}$, $\lim_{y \to \hat{y}} F_i(y) = F_i(\hat{y})$. Component functions of $F$ are those declared from Eq. 7 to Eq. 9 for which the limit converges to its (finite) value at $\hat{y}$ and therefore they are continuous. Hence, $F$ is continuous over $\mathcal{C}$ and a solution exists for the fixed point equation of Eq. 10.

## 6 Analytical and Experimental Results

Extensive experiments performed in order to asses the accuracy and feasibility of the model proposed in Fig. 5 with regards to the performance measure defined in Eq. 1. The results of measurements from real world experiments are compared to those of analytically solving the fixed point model and errors are reported relatively denoted by $\theta$.

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

Where $\tau$ stands for the execution times obtained analytically from the proposed SAN model and $T$ denotes those measured from experiments. Numerical solutions for proposed SAN models are acquired using the state-of-the-art tool, Mobius [33], and its iterative steady state solver. In order to successively solve each sub-model of Fig. 5 and automatically pass parameters among them a script employed which stops when the difference between $\tau$ values of the last iteration and the current iteration falls under a threshold, implying the convergence to the fixed point. The script is represented as a pseudo-code in Fig. 6 where $\delta$ is the convergence threshold and is assigned to $0.01$ throughout this paper. Tests show that the fixed point is reached in no more than 4 iterations, thus the convergence is fast. However, solving time of the model is reported in Table 6 and Table 7. Numerical analysis is performed on a personal computer with an i5 Intel processor and 6 GB of memory.

**Output:** $r_1, r_2, \ldots, r_Q$
$\mu_{W_1}, \mu_{W_2}, \ldots, \mu_{W_Q} \leftarrow initial\ values$
$\mu_{R_1}, \mu_{R_2}, \ldots, \mu_{R_Q} \leftarrow initial\ values$
$\mathbb{P}_{I_1}, \mathbb{P}_{I_2}, \ldots, \mathbb{P}_{I_Q} \leftarrow initial\ values$
$r'_1, r'_2, \ldots, r'_Q \leftarrow initial\ values$
$converge \leftarrow false$
**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**
  $r'_1, r'_2, \ldots, r'_Q \leftarrow r_1, r_2, \ldots, r_Q$
**end while**
**return** $r_1, r_2, \ldots, r_Q$

Fig. 6. Fixed point iterations pseudo-code

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 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 of the TPC-DS query table in different configurations. These queries are of variety of DAGs from two to four stages each containing hundreds of tasks. The profile of each query, namely its DAG, number of tasks in each stage ($N_{ij}$), and average task execution time in each stage ($\mu_{R_{ij}}$) is realized through 10 time pilot execution of the query. The learning-based technique presented in [31] made it possible for us to accurately find out 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 and is omitted here to conserve space. Also for the sake of reproducibility, traces of the experiments, SAN model description, and fixed-point iteration script are distributed online [2].

Spark 2.4.1 release is installed on top of the Hadoop version 3.1.1 cluster comprising 4 VMs each configured with 10 cores and 25 GB of Memory. Each executor has 2 GB of memory and a single dedicated core while 10 GB of RAM

1. github.com/gregrahn/tpcds-kit
2. github.com/mohsenasm/Spark-Fixed-Point-SAN-Model-Paper

TABLE 6
Real and predicted execution times for 2 queues scenario

| # | Queue | $N_q$ | Query | $S_q$ (%) | $T$ (ms) | $\tau$ (ms) | $\theta$ (%) | 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 | |

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 6 and Table 7 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 6 and Table 7 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 scenario of 2 and 3 parallel queues respectively, which is an acceptable error and proves that numerical results are accurate enough. While the reported accuracy is an indicator of the effectiveness of the proposed prediction tool, capturing the behavior of the target system is another advantage that arrives 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. Following is another possible application of the proposed model.

### 6.1 Minimizing the makespan

In many use cases, particularly, minimizing the makespan, it is favorable to balance execution times in different YARN queues. Imposing such equity among queues results in minimized makespan, which is the completion time of the queue with the longest job. Analytical prediction models are useful in finding the most suitable capacity partitioning and here we tested the practicality of our fixed-point model in this regard. We conducted experiments on a representative scenario of two YARN queues running queries 52 and 84 each by a single user and capacity percentages that vary from 90% to 10%.

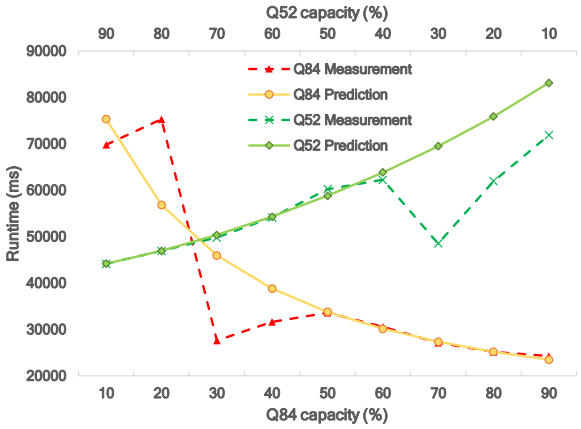


Fig. 7. Makespan analysis for different capacity configurations

Results of experiments are depicted in Fig. 7 with dashed lines which show that an approximate capacity partition of 75% for the query 52 and 25% for the query 84 leads to the almost same execution time for both queues and therefore optimal makespan. The analytical fixed-point model proposed in this paper is also solved for similar configurations and numerical results are illustrated in solid lines. Intersection of numerical lines is just about the one that was obtained from experiments. This implies that the execution time prediction proposed in model of Fig. 5 is useful in finding a proper allocation scheme between YARN queues in order to minimize the overall makespan.

## 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. Their approach to model the performance

TABLE 7
Real and predicted execution times for 3 queues scenario

| # | Queue | $N_q$ | Query | $S_q$ (%) | $T$ (ms) | $\tau$ (ms) | $\theta$ (%) | 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 | |

of the framework 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. Here, we try to introduce different viewpoints and discuss their strengths and weaknesses starting from the most recent ones.

Both simulation and analytical approaches are leveraged in [18] in order to predict the execution time of Spark jobs and 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 they have presented an analytical model based on queuing networks, simulation is used in order to solve the model instead of numerical methods. This is also the case in [34] where authors have proposed a model based on process algebra for Spark and solved it using simulation tool. The proposed formalism, however, is limited to a number of tasks running in parallel and lacks the power to expressive more complex DAGs. Other examples of simulation efforts are [12], [13] 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 [11], [35], [36], [37], linear regression of selected sample executions are considered as the predictor for the actual-size performance of Hadoop application. Based on a similar sampling approach, more sophisticated learning techniques have been adopted such as deep reinforcement learning [14] or combining multiple regression models each for a single stage of the whole application, [15]. Regarding the feature selection challenge, authors in [16] 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 [19] 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 reality and also they have 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 [20] 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 [31], 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 [38] targeted the problem of latency prediction in fork-join environments with analytical models and more particularly with closed-form solutions realized for queuing networks. Employment of stochastic formalisms is still popular and in one of the most recent ones, [39], a Petri-net model is proposed as a Representative of stream processing in Spark.

In our previous work [40], in order to alleviate the scalability issue of predicting multi-queue YARN environment we considered lumping technique which introduced a significant error of about $15\%$. The contribution was built upon the efforts for developing analytical models for single queue scenarios in [41], [42]. 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 not considered multiplicity of nor the users neither the queues. The details of the YARN scheduling mechanism despite the wide adoption in industry, is also rarely explored in literature.

## 8 Conclusions and Future Work

In this paper we presented a numeric analytic model based on SAN formalism for the performance evaluation of Spark jobs running on a YARN cluster considering the multiplicity of YARN queues and users. Due to the low scalability of the monolithic design we then decomposed the model into sub-models each representing a single YARN queue negotiating parameters in a fixed-point scheme. We tested the accuracy and feasibility of numerical results obtained from solving the model against benchmark experiments which showed acceptable errors and low running times for a variety of configurations. Average error of $5.6\%$ proves the applicability of the model as a prediction tool which can be useful in resource management decisions and Service Level Agreements.

Interesting follow-up research can be centered around utilizing the presented model in optimizing cluster parameters with the aim of decreasing operating costs or frequency of SLA-violations. More accurate and faster profiling approaches can also be leveraged in order to further increase the prediction accuracy or lower the solving time.

## 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] Apache, "Hadoop." [Online]. Available: http://hadoop.apache.org/

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] S. Sidhanta, W. Golab, and S. Mukhopadhyay, "Deadline-Aware

Cost Optimization for Spark," *to appear in the IEEE Transactions on Big Data*, 2019.
[32] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, 1970.
[33] 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.
[34] J. Ding, Y. Xie, and M. Zhou, "Performance Modeling of Spark Computing Platform," *Studies in Computational Intelligence*, vol. 810, pp. 121–133, 2020.
[35] M. Alalawi and H. Daly, "Designing a Hadoop MapReduce Performance Model using Micro Benchmarking Approach," in *Proceedings of the International Conference on Innovation in Computer Science and Artificial Intelligence*, London, UK, jul 2019, pp. 1–11.
[36] Z. Fu and Z. Tang, "Optimizing Speculative Execution in Spark Heterogeneous Environments," *to appear in the IEEE Transactions on Cloud Computing*, 2019.
[37] 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.
[38] 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.
[39] 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.
[40] 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.
[41] 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.
[42] 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.

**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 Department of Computer Engineering at the Sharif University of Technology, Tehran, Iran. His main research interests include Performance Evaluation, Big Data frameworks, Cloud Computing, and Failure Prediction.

**Mohammad-Mohsen Aseman-Manzar** is currently MSc. student of Computer Engineering at Sharif University of Technology. He received his BSc. of Computer Engineering from Iran University of Science and Technology (IUST) in 2018. His research interest includes performance modeling, Big Data frameworks, and mixed power and performance modeling of data centers and Cloud environments. In his industrial life, he is a full-stack developer and works with various frameworks and languages.

**Reza Entezari-Maleki** received the B.S. and M.S. degrees from the Iran University of Science and Technology (IUST) in 2007 and 2009, and the Ph.D. degree from the Sharif University of Technology in 2014, all in computer engineering. He was a postdoctoral researcher at Institute for Research in Fundamental Sciences, Iran, before joining IUST as an assistant professor in 2018. His main research interest is performance and dependability modeling.

**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.

**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, and distributed real-time systems. He is a senior member of the IEEE and the ACM.