



# Scalable Performance Modeling and Evaluation of MapReduce Applications

Soroush Karimian-Aliabadi<sup>1</sup>(✉), Danilo Ardagna<sup>2</sup>, Reza Entezari-Maleki<sup>3</sup>,  
and Ali Movaghar<sup>1</sup>

<sup>1</sup> Computer Engineering Department, Sharif University of Technology, Tehran, Iran  
[skarimian@ce.sharif.ir](mailto:skarimian@ce.sharif.ir), [movaghar@sharif.ir](mailto:movaghar@sharif.ir)

<sup>2</sup> Dipartimento di Elettronica Informazione e Bioingegneria,  
Politecnico di Milano, Milan, Italy  
[daniло.ardagna@polimi.it](mailto:daniло.ardagna@polimi.it)

<sup>3</sup> School of Computer Engineering,  
Iran University of Science and Technology (IUST), Tehran, Iran  
[entezari@iust.ac.ir](mailto:entezari@iust.ac.ir)

**Abstract.** Big Data frameworks are becoming complex systems which have to cope with the increasing rate and diversity of data production in nowadays applications. This implies an increase in number of the variables and parameters to set in the framework for it to perform well. Therefor an accurate performance model is necessary to evaluate the execution time before actually executing the application. Two main and prominent Big Data frameworks are Hadoop and Spark, for which multiple performance models have been proposed in literature. Unfortunately, these models lack enough scalability to compete with the increasing size and complexity of the frameworks and of the underlying infrastructures used in production environments. In this paper we propose a scalable Lumped SRN model to predict execution time of multi-stage MapReduce and Spark applications, and validate the model against experiments on TPC-DS benchmark using the CINECA Italian super computing center. Results show that the proposed model enables analysis for multiple simultaneous jobs with multiple users and stages for each job in reasonable time and predicts execution time of an application with an average error about 14.5%.

**Keywords:** Performance modeling · Scalable modeling · Stochastic reward nets · BigData frameworks · Map Reduce

## 1 Introduction

Huge amount of data is available in datacenters and is steadily, being produced in high velocity [1]. Data scientists need fast frameworks, specific algorithms, and even new programming paradigms to efficiently process this Big Data. The

continuous struggle of data scientists with increasing size of data to be analyzed, led to handful of practical tools and methods. In 2008, Dean and Ghemawat proposed MapReduce (MR) paradigm [2] to process large amount of data on multiple node cluster to increase parallelism and therefor improved performance. The MR paradigm was not globally used until useful Hadoop framework [3] developed in 2011 by Apache. The Hadoop Distributed File System (HDFS) is a primary layer of the Hadoop ecosystem but not the only one. In 2013, Vavilapalli et al. [4], introduced YARN layer to the Hadoop cluster in order to specialize the resource management and make it dynamic rather than Hadoop's earlier static allocation scheme. With more complex dataflow in MR applications there was a need to cut down the complexity into multiple stages and thus Directed Acyclic Graphs (DAG) was chosen by Tez [5] developers to demonstrate the dataflow between stages of a complex application. Taking advantage of the memory's high speed and the Resilient Distributed Dataset (RDD) concept, Spark was created and became popular due to high speed and the ease of application development.

Tuning the framework and cluster parameters in order to reduce the execution time of a BigData application was a challenge from the earliest steps and a main part of this optimization process is to predict the execution time for a given set of parameters. But With each step in development of a more advanced framework for processing BigData, new set of parameters and complexity is created and execution time prediction made more and more challenging. A lot of works have been done in literature to simulate [6], model [7], or learn [8] the process, but their accuracy and scalability is only enough for simple runs with a single job running by one or more users and not for more complex applications with multiple multi-stage jobs running by number of users.

Among the approaches put into practice to predict execution time, analytical models play a prominent role. Simulation methods are time-consuming, and comprehensive simulators are rare and heavy to work with. Exploratory approaches like studying the history of past runs are also time-consuming and need a complete setup, while sometimes, predictions have to be made prior to the real setup. Learning traces is limited to a specified criteria and other performance measures need another whole learning process. Analytical methods on the other hand are fast to run, accurate enough, and give more insights on the process, hence number of performance measures can be studied using a single model. While deriving mathematical relations for a complex system like Spark framework is almost impossible, stochastic models are more feasible to work with. Fortunately, Petri Net (PN) and its stochastic derivations proved to be practical in literature and also have great tool support. Not all derivations of PN are analytically solvable, so the choice for the formalism is limited to few options which can be converted to Markov Models like Markov Reward Model (MRM). Stochastic Reward Net (SRN) is a formalism based on Stochastic Petri Nets (SPN) and is chosen as the formalism for building proposed models in this paper.

An inherent drawback of stochastic models like SRNs is the state space explosion problem due to increase in system variables and their cardinality. In the Spark framework which is usually running on top of the Hadoop cluster gov-

erned by YARN resource manager, multiple jobs are submitted simultaneously which have different execution DAGs. Number of Map/Reduce tasks in each stage is also different from other stages and jobs could be submitted by different users and thus dedicated priorities according to the YARN resource management policy. Modeling all of these variabilities in a monolithic model easily meets the state space explosion problem and is not feasible to analyze by regular hardware.

To tackle this challenge, analytical models based on SRNs are proposed in this paper to accurately model the Hadoop and Spark framework running multiple jobs. We assume that MR Job is running on top of the Hadoop cluster and is governed by YARN resource manager with capacity scheduler. A lumping technique is also proposed to break down the complexity of the model and thus eliminate the state space explosion problem. Proposed models are validated against the real experiments of TPC-DS benchmark on CINECA supercomputer. Accuracy of proposed models in predicting execution time of Spark applications compared to results from experiments show an average error of 14.5% and the runtime of the analytic-numeric model solver is 15 s in average, which demonstrate the high scalability of the proposed SRN models.

The remaining parts of this paper are organized as follows. The Sect. 2 is dedicated to related proposals available in the literature, Sect. 3 presents the description of the features of the application frameworks. Our proposed SRN models for Hadoop MR and Spark applications in the single-class and multi-class forms are included in Sects. 4 and 5, respectively. In Sect. 6, we introduce the results obtained by the proposed models and their validation against the real systems. Finally, in Sect. 7, the paper concluded with some directions for future work.

## 2 Related Work

There are several researches on performance analysis of Big Data applications and tuning framework and cluster parameters. In this section, different methods for performing performance evaluation are being classified and reviewed one by one. Performing experiments and studying previous executions is a general way to reach insight on the performance of the framework. For example, in [9] MR job execution logs are used to assess performance measures and predict the future runs. Monitoring the execution of jobs in Hadoop, as proposed in [10], helps fine tuning the cluster and administrating the configuration parameters which are investigated thoroughly in [11].

A good predictor can learn from past executions and machine learning has a handful of tools and methods in this regard. From regression [12] to more sophisticated techniques like SVR [8] have been used in literature. Ernest [13], for example, is a tool set designed to predict Spark job execution time in large scale based on the behavior of the job on small samples of data. The sample set is produced using optimal experiment design and Non-negative Least Squares (NNLS) method to fit the model. Similar to Ernest, in [14] authors have explored sample representative mini dataset to train their model. Each stage of Spark

application is first modeled using multiple polynomial regression and afterwards, Stage predictions are aggregated through the critical path of the execution DAG to estimate the whole job runtime. Combining analytical modeling power with machine learning methods has led to a operational system of MR job execution time estimator in [8]. Ataie et al. have combined queuing network model with SVR technique in [8] to further increase the accuracy and reduce the number of experiments to be performed for training the model.

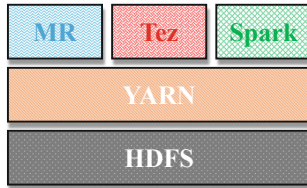
Great efforts have been made to build a comprehensive simulator for Big Data frameworks like MR and Hadoop [6, 15], and here, just few of them are mentioned. Ardagna et al. [6] have proposed DAGSim, a novel ad-hoc and fast discrete event simulator, to model the execution of complex DAGs, which can be used to predict Spark application runtime. Instead of building a Heavyweight simulator, others have designed simple graphical models which can be simulated. The approach, presented by Barbierato et al. [16], exploits Generalized Stochastic Petri Nets (GSPNs) alongside other formalisms such as process algebras and Markov chains to develop multi-formalism models and capture Hive queries performance behavior. More recently, Ruiz et al. [17] formalized the MR paradigm using Prioritized Timed Colored Petri Nets (PTCPNs). They validated the model and carried out a performance cost trade-off analysis. In [18], queuing network and Stochastic Well-formed Nets (SWN) simulation models have been proposed and validated for MR applications, considering YARN as resource manager. Requeno et al. [7] have proposed a UML profile for Apache Tez and transformed the stereotypes of the profile into Stochastic Petri Nets (SPNs).

Models solved analytically instead of simulation are the main focus of this work, specifically, models based on Markovian processes. Analytic model and formulation has a great value in getting to know a system, while the later, is difficult or some time impossible in complex systems like Big Data frameworks. Although finding a mathematic relation for execution time of a MR job is unachievable, Upper and lower bounds were analytically derived for MR job execution time in shared Hadoop clusters by authors in [19]. On the other hand, numerous works have used the more feasible approach of building analytical models. SPNs have been used by [20] for performance prediction of adaptive Big Data architecture. Mean Field Analysis was applied by authors in [20] to obtain average performance metrics. In order to cope with the Inevitable state space explosion problem, authors in [21] used Fluid Petri Nets to simplify the actual model. They proposed fluid models to predict the execution time of the MR and Spark application.

### 3 System Architecture and Application Structure

In order to better understand the behavior of proposed models, it's necessary to provide a background for the target system and application as well as the assumptions. The primary system of concern is Spark framework and Spark application. This choice is due to Spark's extensive use among enterprises and

its popularity between data scientists [22]. However, modeling starts from simple Hadoop MR application, and then the applicability of models to Spark framework is discussed. Spark is able to run on top of the Hadoop cluster governed by YARN resource manager and since it is the usual deployment option [23], it is also the choice of our architecture in this paper. The outline of the system is depicted in Fig. 1.

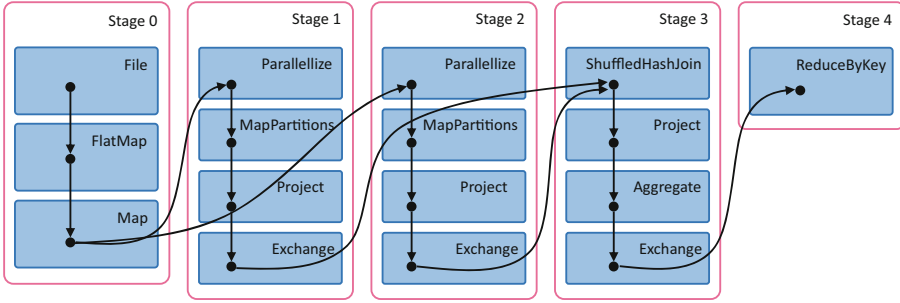


**Fig. 1.** The general schema of BigData frameworks

An MR job is consisted of three main phases: Map, Shuffle, and Reduce. Each of these phases include number of tasks that run in parallel on different cluster nodes. Map tasks perform computation on input data chunks and Shuffle tasks are responsible for gathering Map phase output to Reduce tasks. Reduce tasks perform aggregation on intermediate data and output the final result. Since Reduce and Shuffle tasks run on same thread, from hereafter, we consider a Reduce task an aggregation of corresponding Shuffle task and the succeeding Reduce task.

Spark programming model is similar to MR but extends it with a data-sharing in memory abstraction called Resilient Distributed Datasets (RDD). Every Spark application consists of a number of stages. A stage corresponds to an operation on RDDs and can be seen as Map, Shuffle, or Reduce phases in MR jobs. Each stage consists of multiple tasks running in parallel and distributed in the cluster. Stages are linked to each other in the form of a Directed Acyclic Graph (DAG) which demonstrates the flow of data between stages and also the execution order among them. The execution DAG specifies the parallelization degree and the critical path as well. A sample Spark application can be seen in Fig. 2.

Although, jobs were scheduled in earlier versions of the Hadoop framework by FIFO policy, better schemes are available today. Hadoop 2.x and Hadoop 3 let more complex schedulers (i.e. Capacity and Fair schedulers) to be plugged into the framework. A cluster is a resource pool in YARN, enabling dynamic allocation of resources (containers) to the ready tasks. We assume Capacity scheme for the YARN layer and this means that in the multi class environment jobs in each class run in FIFO manner and next job can only start if the last stage of the previous job has acquired all resources necessary to accomplish. YARN capacity scheme indicates a specified share of the resources for each class, so classes can race to acquire resources until they are under provisioned according



**Fig. 2.** A sample Spark application execution DAG

to their share. Once a class acquired all its share from the resources, it should leave available resources in the favor of other under provisioned classes.

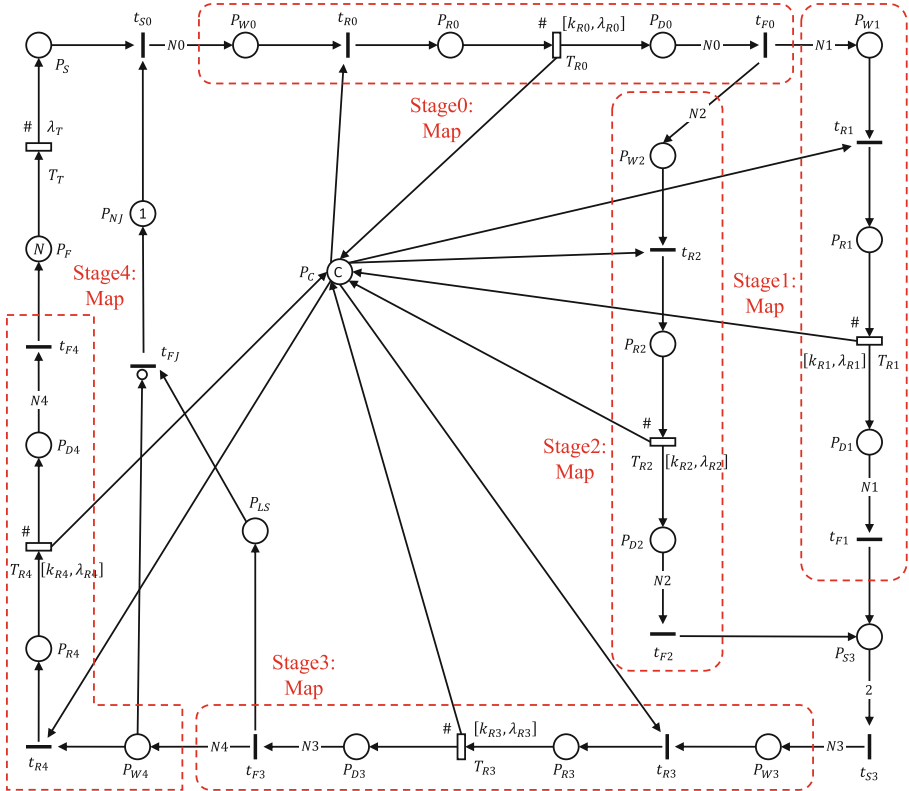
In this paper, we consider target Hadoop clusters running on a set of homogeneous resources [24], including MR and Spark execution engines on top of the YARN Capacity scheduler [18]. This implies that the cluster capacity is partitioned into multiple queues and within a queue, multiple applications are scheduled in a FIFO manner. Multiple users can run the same query, which is submitted to a specific queue. Moreover, after obtaining results, end users can submit the same query again (possibly changing interactively some parameters) after a think time. In other words, a multi-class closed performance model is considered [25].

## 4 Single-Class Model

In this section, the proposed SRN model for a Spark application is presented. Formal definition of SRN formalism, its structure, and behavior are given in [26] which have been omitted herein for the sake of space limitation. SRN formalism is widely used in other areas of computer science such as Cloud Computing [27] and Computational Grids [28], for performance evaluation, and is proved to be practical. First, the proposed model is described in detail and afterwards, the validation and computation time of the model is discussed and lumping method is proposed to overcome the model complexity.

### 4.1 Proposed Model

The SRN model proposed for Spark application is shown in Fig. 3 and is modeling the execution of sample Spark application of Fig. 2. The model includes different stages, shared resources, think time and the scheduling mechanism. It should be noted that since stages in Spark are analogous to phases in Tez and MR frameworks, this model can be easily simplified for modeling Tez and MR applications as well. The detailed description of model behavior is as follows.



**Fig. 3.** The SRN model proposed for single class Spark application

In the initial state, there are  $N$  tokens in place  $P_F$  showing the jobs waiting to start execution. The marking dependent timed transition  $T_T$  models the think time of a waiting job. Upon completion of this transition, a token from place  $P_F$  is moved to place  $P_S$  with rate  $\lambda_T$ , which is the rate of the exponential distribution considered for transition  $T_T$ . Existence of a token in place  $P_S$  triggers instantaneous transition  $t_{S0}$  to start the job if the place  $P_{NJ}$  has a token to consume. The place  $P_{NJ}$  initially contains a token modeling the possibility of starting a waiting job according to the Capacity scheduler policy. At the start of a job, the instantaneous transition  $t_{S0}$  will produce  $N_0$  tokens in place  $P_{W0}$ , each one representing a task in the first stage. Entities  $P_{Wi}$ ,  $t_{Ri}$ ,  $P_{Ri}$ ,  $T_{Ri}$ ,  $P_{Di}$ ,  $N_i$ , and  $t_{Fi}$  where  $0 \leq i \leq 4$  together simulate  $i$ th stage. Place  $P_{Wi}$  is starting point of stage  $i$  and shows tasks waiting for resource. Instantaneous transition  $t_{Ri}$  Allocates an available resource to a task, which removes one token from place  $P_C$  and one from  $P_{Wi}$ , and adds a token to place  $P_{Ri}$ . Place  $P_C$  is modeling the pool of containers, which is initially set to contain  $C$  tokens representing the total number of containers. The execution of a single task is modeled by the timed transition  $T_{Ri}$ , which returns the resource to the pool of available

resources whenever a task is done. This transition is characterized by the Erlang distribution with shape  $k_{Ri}$  and a marking dependent rate  $\lambda_{Ri}$ . According to our experiments, the exponential distribution is not the case for the task execution time, and task execution time fits better with more general distributions like Erlang. On the other hand, for the SRN model to be analytically solvable, all timed activities have to be exponentially distributed [29]. Fortunately, an Erlang distribution can be simulated with a set of continuous exponential activities [30] helping us to use the analytically solvable SRN models, when some actions of the system follow Erlang distribution. Parameters of the distributions are being calculated from the experiment logs.

Once the number of tokens in place  $P_{Di}$  reaches the total number of the tasks, the  $i$ th stage is finished and the instantaneous transition  $t_{Fi}$  consumes  $N_i$  tokens from  $P_{Di}$ , where  $N_i$  denotes the number of tasks in  $i$ th stage, and starts  $(i + 1)$ th stage with producing  $N_{i+1}$  tokens in place  $P_{W(i+1)}$ . The completion of transition  $t_{Fi}$  for the second to last stage, also results in adding a token to place  $P_{LS}$ , which indicates that a job is performing its last stage. Recalling from Sect. 3, the Capacity scheduler implies that the next job can start executing only when the previous job has received all of the necessary resources for completing its last stage. Similarly, in our model, instantaneous transition  $t_{FJ}$  enables, whenever there is a token in place  $P_{LS}$  and there is no token left in place  $P_{W4}$ . Afterwards, transition  $t_{FJ}$  removes a token from place  $P_{LS}$ , and puts a token into place  $P_{NJ}$  enabling instantaneous transition  $t_{S0}$  to start the next job. Notice that place  $P_{S3}$  and transition  $t_{S3}$  are assuring that both *stage*<sub>1</sub> and *stage*<sub>2</sub> are completed before *stage*<sub>3</sub> is started.

The performance measure to be assessed by the proposed model of Fig. 3 is the steady-state mean execution time of jobs, which is the average time a token needs to move from place  $P_S$  to place  $P_F$ . In order to compute the mean execution time, the reward shown in Eq. 1 is defined.

$$r = \frac{N}{throughput_{t_{F4}}} - \frac{1}{\lambda_T} \quad (1)$$

where  $throughput_{t_{F4}}$  is the throughput of the instantaneous transition  $t_{F4}$  and can be calculated by Eq. 2.

$$throughput_{t_{F4}} = \mathbb{P}(\#P_{D4} = N_4 - 1) \cdot \lambda_{R4} \quad (2)$$

where  $\mathbb{P}(\#P_{D4} = N_4 - 1)$  is the probability of being in a state where all but one tasks are finished in the last stage, so there are  $N_4 - 1$  tokens in place  $P_{D4}$  and one token left to finish the entire job. This probability is multiplied by  $\lambda_{R4}$ , which is the rate of executing a task in the last stage.

## 4.2 Lumping Technique

Although the proposed model seems to conform with the Spark execution model, in real world, scalability remains a low point for this model. Technically, the model of Fig. 3 could easily grow in state space with increasing number of stages



and face the state space explosion problem. The issue gets even worse in multi-class environments where multiple multi-stage Spark applications are running in parallel. Our experiments, also support this claim, so that, the SRN model of Fig. 3 takes more than 4 days to solve analytically. Details of experiment setup and tool set is described in Sect. 6. Therefore, a heuristic approach is introduced to decrease the complexity of the model by reducing the cardinality of the parameters. The basics of the heuristic is the fact that according to the experiments, tasks in a single stage run in waves. That is when the number of tasks is greater than number of cores, then at the beginning of the  $i$ th stage, all of the  $C$  cores are assigned to tasks and  $N_i - C$  tasks are left. After a while, running tasks, eventually finish and release their acquired resources. With the assumption that the runtime of an individual task is almost similar to other tasks of the same stage, then according to Fig. 4, next  $C$  tasks will acquire  $C$  available resources. A group of  $C$  tasks, is called a wave.

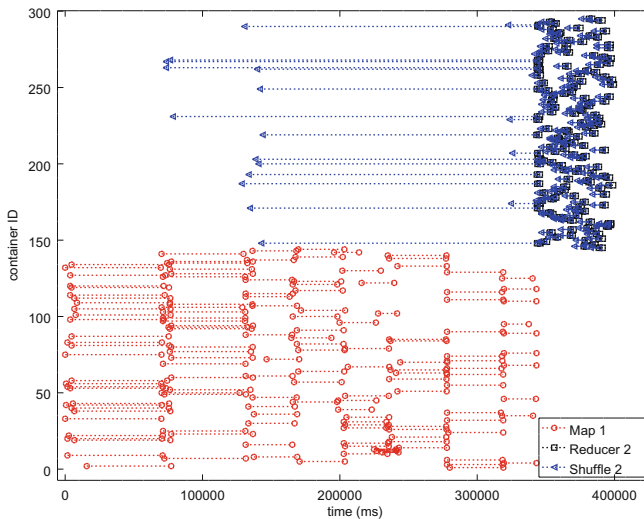


Fig. 4. The execution plot for a sample MR application

Modeling the execution of waves instead of individual tasks, reduces the cardinality of tasks and obviously increases the scalability of the model. This will be done by lumping tasks to waves and assume a single resource with  $\lceil N_i/C \rceil$  waves. Although the assumption is not real and implies some error to the results, since tasks execution times are not exactly same, but experiments in following section shows that the error is still acceptable.

## 5 Multi-class Model

So far, the execution of a single class of application by multiple users is discussed and an analytic SRN model is devised to evaluate the execution time of such

environment with a heuristic to make the model feasible to solve in reasonable time. In this section, the execution of multiple application classes is addressed, each running by multiple users and the SRN model of Fig. 3 is extended to evaluate the performance of a multi-class environment using the lumping technique discussed in Subsect. 4.2.

### 5.1 Proposed Model

According to the YARN scheduler, different classes of applications can be managed in multiple queues each has a share of the resources which is indicated by the framework operator. This means when a class of application is under provisioned according to its share, and at the same time is requesting for a resource then YARN will provide this class with more priority than other classes. In order to increase the utilization YARN also lets fully provisioned classes to acquire more resources than their share, only if remaining resources are free and not being demanded by other under provisioned classes. Recall from Sect. 3 that inside each queue different instances of the application class are running in FIFO manner. As claimed so far, the SRN model for multi-class environment can be built by replicating SRN model of Fig. 3 as many as the number of classes and let them race to acquire resources by the rules described earlier. Here, in order to be more concrete, assume 4 classes of MR applications running in parallel and the model for this sample environment is depicted in Fig. 5. The detail of model structure and behavior is as follows.

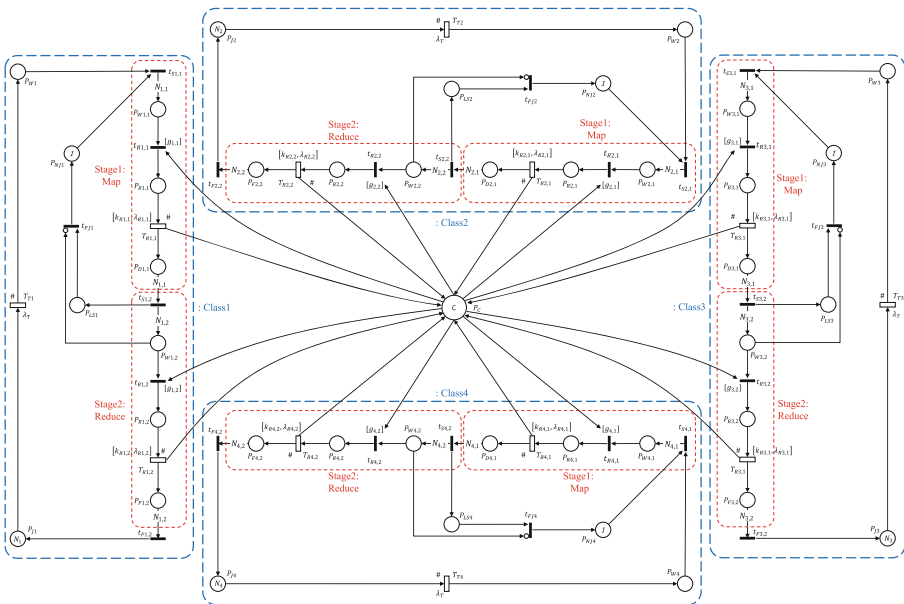


Fig. 5. The SRN model proposed for multi class MR application

For the sake of simplicity, MR applications are considered here, where as you can see in Fig. 5 there are just two Map and Reduce stages in each application. While the behavior of sub-models for each of the classes is similar to Fig. 3, the main difference is how instantaneous transitions  $t_{Ri,j}$  enable. Instantaneous transition  $t_{Ri,j}$  is responsible for acquiring resource in  $i$ th class, where  $1 \leq i \leq 4$ , and  $j$ th stage, where  $1 \leq j \leq 2$  and enables according to the guard function  $g_{i,j}$ . The guard function  $g_{i,j}$  is formulated as below.

$$g_{i,j} = \begin{cases} 1, & \text{if } (\#P_{Wi,j} > 0) \text{ and } (\#P_C > 0) \text{ and } ( \\ & (\sum_{j=1}^2 \#P_{Ri,j} < S_i \cdot C) \text{ or} \\ & \text{for each class } k \text{ and } k \neq i : \\ & (\sum_{j=1}^2 \#P_{Wk,j} = 0 \text{ or } \sum_{j=1}^2 \#P_{Rk,j} \geq S_k \cdot C) \\ & ) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where  $S_i$  denotes the share factor of  $i$ th class. The desired performance measure of the proposed model of Fig. 5 is similar to the performance measure devised for model of Fig. 3, that is steady-state mean execution time of each job in each class and is calculated by Eq. 4.

$$r_i = \frac{N_i}{throughput_{t_{Fi,2}}} - \frac{1}{\lambda_T} \quad (4)$$

where  $throughput_{t_{Fi,2}}$  can be calculated by Eq. 5.

$$throughput_{t_{Fi,2}} = \mathbb{P}(\#P_{Fi,2} = N_{i,2} - 1) \cdot \lambda_{Ri,2} \quad (5)$$

## 5.2 Lumping Technique

The modal of Fig. 5 with multiple classes of applications, each running by multiple users has a huge state space which will be more unattainable if number of tasks in each stage increases. In Sect. 4 lumping technique proposed to scale down the number of tasks to number of waves. Here a similar technique is leveraged to make the model feasible to analyze. Assuming that  $c_i$  containers are assigned to each class  $i$ ,  $c_{gcd}$  can be calculated as the gcd of  $c_i$  values. Afterwards, both the number of tasks and containers are divided by  $c_{gcd}$ , simulating a lumping technique. For example, if there are 240 total containers and each of 4 application classes have a share of 60 containers then a stage with 300 tasks will have approximately, 5 waves. This approximation is not realistic since sometimes the application will receive less or more resources than its share, however this assumption enables the model to be analytically tractable with an acceptable error.

## 6 Numeric-Analytic Results

The numeric-analytic solution to proposed SRN models is preferred to simulation results, since simulation methods can grow in runtime and usually lack in

scalability as discussed in Sect. 1, therefore, we chose SPNP tool [35] and its steady-state iterative solver, in order to analyze our proposed models, considering that SPNP is the state of the art tool for analytic solutions of SRN models. In order to assess the accuracy of the numeric results obtained from the model we have conducted several experiments on real world platforms including public Clouds to private cluster and we defined the measure below as the error value.

$$\theta_{SRN} = \left| \frac{\tau_{SRN} - T}{T} \right| \quad (6)$$

where  $\theta_{SRN}$  denotes the relative error between the SRN model and experiments,  $T$  is the execution time of a job on the real system under test which is measured from the experiment, and  $\tau_{SRN}$  is the execution time received from the SRN model as the result of numeric solution. The experiment setup includes the PICO Big Data cluster available at CINECA [32] configured with number of cores ranging from 40 to 120, each of them assigned to a single container and the private cluster. Our IBM Power8 (P8) private cluster includes 4 VMs with 11 cores and 60GB of RAM for each. Spark executors are configured to leverage 2 cores and 4GB of RAM, while Spark drivers use 8GB of RAM. The configurations vary in number of cores from 6 to 44 and number of executors from 3 to 22.

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

(a)  $Q_1$ 

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

(b)  $Q_2$ 

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

(c)  $Q_3$ 

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

(d)  $Q_4$ 

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

(e)  $Q_5$ 

**Fig. 6.** MR queries ( $Q_3$  to  $Q_5$ )

The dataset used for running the experiments was generated with the TPC-DS benchmark data generator [33], which is the industry standard for benchmarking data warehouses. Datasets are in the form of external tables for the Hive [34] queries and their size varies from 250 GB to 1 TB. Different queries are considered to be executed on datasets as MR or Spark applications. For the case of single-class scenario, both MR and Spark applications were chosen. Queries  $Q_1$  and  $Q_2$  are designed in Fig. 6 for which number of Map and Reduce tasks vary from 1 to 151 in different configurations. Queries  $Q_{26}$  and  $Q_{52}$  are selected from TPC-DS catalog for Spark applications and vary in number of stages from 4 to 8 each one having 1 to 1000 tasks. Finally, for the case of multi-class scenario MR queries were completed with queries  $Q_3$  to  $Q_5$  of Fig. 6 so different configurations of queries  $Q_1$  to  $Q_5$  can be executed in 4 queues. Number of users in each queue differs from 2 to 10 and number of tasks range from 1 to 600 in each Map or Reduce phase.

**Table 1.** Results obtained from the proposed SRN model for single class applications

Query	Users	Cores	Scale [GB]	$T$ [ms]	$\tau_{SRN}$ [ms]	$\vartheta_{SRN}$ [%]
$Q_1$	1	60	250	80316	81285	1.21
$Q_2$	1	60	250	84551	86624	2.45
$Q_1$	3	20	250	1002160	1059403	5.71
$Q_1$	3	40	250	340319	380881	11.92
$Q_2$	3	20	250	95403	88982	6.73
$Q_2$	3	40	250	86023	76936	10.56
$Q_1$	5	20	250	1736949	1827978	5.24
$Q_1$	5	40	250	636694	688759	8.18
$Q_2$	5	20	250	145646	148453	1.93
$Q_2$	5	40	250	90674	106200	17.12
$Q_{26}$	1	24	250	178714	142446	20.29
$Q_{26}$	1	32	250	168041	116364	30.75
$Q_{52}$	1	24	250	181496	144862	20.18
$Q_{52}$	1	32	250	162232	121392	25.17
$Q_{52}$	1	48	750	279243	234573	16
$Q_{52}$	1	48	1000	359987	312014	13.33
$Q_{26}$	1	6	500	2532250	2720902	7.45
$Q_{26}$	1	8	500	2071159	2179066	5.21
$Q_{26}$	1	10	500	1778802	1878948	5.63

in order to estimate the mean execution time of tasks in stages which is necessary to solve our proposed SRN models a profiling step conducted which is a common idea as stated in different literatures [13, 14]. A pilot execution was designed for

**Table 2.** Results obtained from the proposed lumped SRN model for multi class MR applications

Configuration	Query	Users	$T$ [ms]	$\tau_{SRN}$ [ms]	$\vartheta_{SRN}$ [%]
1	$Q_2$	5	118667	86094	27.45
	$Q_5$	5	120947	117429	2.91
2	$Q_1$	4	206938	235267	13.69
	$Q_3$	2	258220	275443	6.67
	$Q_4$	2	246750	250426	1.49
3	$Q_2$	10	252800	182187	27.93
	$Q_5$	10	246702	244860	0.75
4	$Q_1$	10	436212	555341	27.31
	$Q_3$	5	599848	710399	18.43
	$Q_4$	5	584583	633454	8.36
5	$Q_1$	5	264515	335193	26.72
	$Q_2$	5	998941	1163866	16.51
	$Q_3$	3	337716	356155	5.46
	$Q_4$	3	331327	338616	2.2
6	$Q_1$	5	363047	469673	29.37
	$Q_2$	5	1479480	1814878	22.67
	$Q_3$	10	1003998	1064639	6.04
	$Q_4$	10	1015219	1053188	3.74
7	$Q_1$	10	468085	601863	28.58
	$Q_3$	5	613296	658311	7.34
	$Q_4$	5	621132	675170	8.7
	$Q_5$	10	1060763	1317149	24.17
8	$Q_1$	10	452974	522958	15.45
	$Q_2$	10	1870229	2316278	23.85
	$Q_3$	5	613190	718536	17.18
	$Q_4$	5	587453	598438	1.87

each query with a minimum size cluster and mean execution times was measured through execution logs. Obtained values were then used for other cluster and dataset sizes. Task durations are measured as average values between 20 runs and fit better with Erlang distribution for Map tasks while for Reduce tasks exponential distribution fits good enough.

The results obtained from the experiments and the proposed SRN model are shown in Table 2 for single-class MR and Spark applications. The average error is 11.31% which offers the acceptable performance of lumping technique. Finally, the accuracy of the proposed model for multi-class MR applications is evaluated in Table 1 which shows an average error of 14.5%. Despite the increase in error,

model runtime reduced from couple days to couple seconds and the improved scalability is the main contribution of lumping technique.

## 7 Conclusion and Future Works

In this paper we have discussed one of the challenges in BigData area. The fact that a BigData cluster which is equipped with the complete stack of frameworks and tools like Hadoop, YARN, Tez, or Spark, has a huge parameter set and tuning these values is not possible without a useful and accurate performance model which can predict the execution time of applications running in the cluster. Previous works in this field are mainly focused on simulation, learning, experiment, and log analysis which are time-consuming, costly, not accurate enough, and not general according to different performance measures. Therefore, analytical models were chosen, due to their low runtime, more general insight, and high accuracy.

New version of frameworks like Spark, support shared environment for simultaneous applications and users to run and make use of available resources. This feature will cause performance models to grow in state space and face state space explosion problem. While previous work on analytical model [18] considers multiple users and multiple stages, it is limited to single-class executions and suffers from state space explosion problem, our proposed model is scalable in the way that can predict execution time of applications in the presence of other simultaneous applications with different classes of jobs.

In this paper, analytical SRN models were proposed to evaluate most popular BigData frameworks Hadoop and Spark. Despite previous works [18] which have only considered single class executions, a lumping method is proposed to cope with the state space explosion problem and therefore, enable our model to evaluate the performance of multi-class executions. SRN models are then solved using numeric-analytic solver which outperforms other methods in low runtime and sufficient accuracy. Results from numeric-analytic solver are then compared to experiments on CINECA supercomputer considering TPC-DS benchmark workloads and the reported error is 11.3% and 14.5% in average for single-class and multi-class scenarios respectively, which is adequate to support capacity planning decisions and what-if analysis [25].

Future work will extend the models to support additional scenarios of interest like execution with faulty nodes, data placement, and speculative execution. Sensitivity analysis can also be derived in order to find the most effective parameters in execution time of applications. Effective parameters are those which will be optimized first to reach improved performance with the least effort.

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

## References

1. Reinsel, D., Gantz, J., Rydning, J.: Data age 2025: the evolution of data to life-critical (2017). <https://www.seagate.com/de/de/our-story/data-age-2025/>. Accessed July 2018
2. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. *Commun. ACM* **51**(1), 107–113 (2008). <https://doi.org/10.1145/1327452.1327492>
3. Apache, Apache Hadoop. <http://hadoop.apache.org/>. Accessed July 2018
4. Vavilapalli, V.K., et al.: Apache hadoop yarn: yet another resource negotiator. In: *Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC 2013*, pp. 1–16. ACM Press, Santa Clara (2013). <https://doi.org/10.1145/2523616.2523633>
5. Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A., Curino, C.: Apache Tez: a unifying framework for modeling and building data processing applications. In: *Proceedings of the 2015 ACM International Conference on Management of Data, SIGMOD 2015*, pp. 1357–1369. ACM Press, Melbourne (2015). <https://doi.org/10.1145/2723372.2742790>
6. Ardagna, D., et al.: Performance prediction of cloud-based big data applications. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018*, pp. 192–199. ACM Press, Berlin (2018). <https://doi.org/10.1145/3184407.3184420>
7. Requeno, J.I., Gascón, I., Merseguer, J.: Towards the performance analysis of Apache Tez applications. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018*, pp. 147–152. ACM Press, Berlin (2018). <https://doi.org/10.1145/3185768.3186284>
8. Ataie, E., Gianniti, E., Ardagna, D., Movaghar, A.: A combined analytical modeling machine learning approach for performance prediction of MapReduce jobs in cloud environment. In: *Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2016*, pp. 431–439. IEEE, Timisoara (2016). <https://doi.org/10.1109/SYNASC.2016.072>
9. Zhang, Z., Cherkasova, L., Loo, B.T.: Benchmarking approach for designing a MapReduce performance model. In: *Proceedings of the ACM/SPEC International Conference on International Conference on Performance Engineering, ICPE 2013*, pp. 253–258. ACM Press, Prague (2013). <https://doi.org/10.1145/2479871.2479906>
10. Dai, J., Huang, J., Huang, S., Huang, B., Liu, Y.: HiTune: dataflow-based performance analysis for big data cloud. In: *Proceedings of the USENIX Annual Technical Conference*, pp. 87–100. USENIX Association, Portland (2011)
11. Jiang, D., Ooi, B.C., Shi, L., Wu, S.: The performance of MapReduce: an in-depth study. *Proc. VLDB Endowment* **3**(1–2), 472–483 (2010). <https://doi.org/10.14778/1920841.1920903>
12. Yigitbasi, N., Willke, T.L., Liao, G., Epema, D.: Towards machine learning-based auto-tuning of MapReduce. In: *Proceedings of the IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 11–20. IEEE, San Francisco (2013). <https://doi.org/10.1109/MASCOTS.2013.9>
13. Venkataraman, S., Yang, Z., Franklin, M., Recht, B., Stoica, I.: Ernest: efficient performance prediction for large-scale advanced analytics. In: *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, pp. 363–378. USENIX Association, Santa Clara (2016)
14. Gibilisco, G.P., Li, M., Zhang, L., Ardagna, D.: Stage aware performance modeling of DAG based in memory analytic platforms. In: *Proceedings of the 9th Interna-*



- tional Conference on Cloud Computing, CLOUD 2016, pp. 188–195. IEEE, San Francisco (2016). <https://doi.org/10.1109/CLOUD.2016.0034>
15. Liu, Y., Li, M., Alham, N.K., Hammoud, S.: HSim: a MapReduce simulator in enabling cloud computing. *Future Gener. Comput. Syst.* **29**(1), 300–308 (2013). <https://doi.org/10.1016/j.future.2011.05.007>
  16. Gribaudo, M., Barbierato, E., Iacono, M.: Modeling apache hive based applications in big data architectures. In: *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools, ValueTools 2013*, pp. 30–38 ICST, Torino (2013). <https://doi.org/10.4108/icst.valuetools.2013.254398>
  17. Ruiz, M.C., Calleja, J., Cazorla, D.: Petri nets formalization of Map/Reduce paradigm to optimise the performance-cost tradeoff. In: *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, Vol. 3*, pp. 92–99. IEEE, Helsinki (2015). <https://doi.org/10.1109/Trustcom.2015.617>
  18. Ardagna, D., Bernardi, S., Gianniti, E., Karimian Aliabadi, S., Perez-Palacin, D., Requeno, J.I.: Modeling performance of hadoop applications: a journey from queueing networks to stochastic well formed nets. In: Carretero, J., Garcia-Blas, J., Ko, R.K.L., Mueller, P., Nakano, K. (eds.) *ICA3PP 2016. LNCS*, vol. 10048, pp. 599–613. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-49583-5\\_47](https://doi.org/10.1007/978-3-319-49583-5_47)
  19. Malekijajid, M., Ardagna, D., Ciavotta, M., Rizzi, A.M., Passacantando, M.: Optimal map reduce job capacity allocation in cloud systems. *ACM SIGMETRICS Perform. Eval. Rev.* **42**(4), 51–61 (2015). <https://doi.org/10.1145/2788402.2788410>
  20. Castiglione, A., Gribaudo, M., Iacono, M., Palmieri, F.: Exploiting mean field analysis to model performances of big data architectures. *Future Gener. Comput. Syst.* **37**, 203–211 (2014). <https://doi.org/10.1016/j.future.2013.07.016>
  21. Gianniti, E., Rizzi, A.M., Barbierato, E., Gribaudo, M., Ardagna, D.: Fluid petri nets for the performance evaluation of MapReduce and spark applications. *ACM SIGMETRICS Perform. Eval. Rev.* **44**(4), 23–36 (2017). <https://doi.org/10.1145/3092819.3092824>
  22. Spark, Apache Spark. <http://spark.apache.org/>. Accessed July 2018
  23. Alipourfard, O., Harry Liu, H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: CherryPick: adaptively unearthing the best cloud configurations for big data analytics. In: *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2017)*, Boston, MA, USA, pp. 469–482 (2017)
  24. Teng, F., Yu, L., Magoulès, F.: SimMapReduce: a simulator for modeling MapReduce framework. In: *Proceedings of the Fifth FTRA International Conference on Multimedia and Ubiquitous Engineering*, pp. 277–282. IEEE, Loutraki (2011). <https://doi.org/10.1109/MUE.2011.56>
  25. Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, 1st edn. Prentice-Hall, Upper Saddle River (1984)
  26. Ciardo, G., Trivedi, K.S.: A decomposition approach for stochastic reward net models. *Perform. Eval.* **18**(1), 37–59 (1993). [https://doi.org/10.1016/0166-5316\(93\)90026-Q](https://doi.org/10.1016/0166-5316(93)90026-Q). <https://www.sciencedirect.com/science/article/pii/016653169390026Q>
  27. Ataie, E., Entezari-Maleki, R., Rashidi, L., Trivedi, K.S., Ardagna, D., Movaghar, A.: Hierarchical stochastic models for performance, availability, and power consumption analysis of IaaS clouds. *IEEE Trans. Cloud Comput.* (to appear). <https://doi.org/10.1109/TCC.2017.2760836>
  28. Entezari-Maleki, R., Trivedi, K.S., Movaghar, A.: Performability evaluation of grid environments using stochastic reward nets. *IEEE Trans. Dependable Secure Comput.* **12**(2), 204–216 (2015). <https://doi.org/10.1109/TDSC.2014.2320741>

29. Meyer, J.F., Movaghar, A., Sanders, W.H.: Stochastic activity networks: structure, behavior, and application. In: Proceedings of the International Workshop on Timed Petri Nets, Torino, Italy, pp. 106–115 (1985)
30. Reinecke, P., Bodrog, L., Danilkina, A.: Phase-type distributions. In: Wolter, K., Avritzer, A., Vieira, M., van Moorsel, A. (eds.) Resilience Assessment and Evaluation of Computing Systems, pp. 85–113. Springer, Berlin (2012). [https://doi.org/10.1007/978-3-642-29032-9\\_5](https://doi.org/10.1007/978-3-642-29032-9_5)
31. Flexiant: Flexiant cloud management software & cloud orchestration. <https://www.flexiant.com/>. Accessed July 2018
32. Cineca: Cineca computing center. <http://www.cineca.it/>. Accessed July 2018
33. Poess, M., Smith, B., Kollar, L., Larson, P.: TPC-DS, taking decision support benchmarking to the next level. In: Proceedings of the 2002 ACM International Conference on Management of data, SIGMOD 2002, pp. 582–587. ACM Press, Madison (2002). <https://doi.org/10.1145/564691.564759>
34. Hive: Apache Hive. <https://hive.apache.org/>. Accessed July 2018
35. Hirel, C., Tuffin, B., Trivedi, K.S.: SPNP: stochastic petri nets. Version 6.0. In: Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds.) TOOLS 2000. LNCS, vol. 1786, pp. 354–357. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-46429-8\\_30](https://doi.org/10.1007/3-540-46429-8_30)