Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets[*]

Danilo Ardagna[1], Simona Bernardi[2], Eugenio Gianniti[1], Soroush Karimian Aliabadi[3], Diego Perez-Palacin[1], and José Ignacio Requeno[4]

[1]Dip. di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
{danilo.ardagna,eugenio.gianniti,diego.perez}@polimi.it
[2]Centro Universitario de la Defensa, Academia General Militar, Zaragoza, Spain
simonab@unizar.es
[3]Dept. of Computer Engineering, Sharif University of Technology, Tehran, Iran
skarimian@ce.sharif.ir
[4]Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain
nrequeno@unizar.es

Abstract

Nowadays, many enterprises commit to the extraction of *actionable knowledge* from huge datasets as part of their core business activities. Applications belong to very different domains such as fraud detection or one-to-one marketing, and encompass business analytics and support to decision making in both private and public sectors. In these scenarios, a central place is held by the MapReduce framework and in particular its open source implementation, *Apache Hadoop*. In such environments, new challenges arise in the area of jobs performance prediction, with the needs to provide Service Level Agreement guarantees to the end-user and to avoid waste of computational resources. In this paper we provide performance analysis models to estimate MapReduce job execution times in

[*]Acknowledgments: This work has received funding from the European Union Horizon 2020 research and innovation program under grant agreement No. 644869 (DICE). Experimental data are available as open data at https://zenodo.org/record/58847#.V5i0wmXA45Q.

Hadoop clusters governed by the YARN Capacity Scheduler. We propose models of increasing complexity and accuracy, ranging from queueing networks to stochastic well formed nets, able to estimate job performance under a number of scenarios of interest, including also unreliable resources. The accuracy of our models is evaluated by considering the TPC-DS industry benchmark running experiments on Amazon EC2 and the CINECA Italian supercomputing center. The results have shown that the average accuracy we can achieve is in the range 9–14%.

Keywords: MapReduce, Performance Models.

1 Introduction

The implementation of Big Data applications is steadily growing today [17]. According to recent analyses, the Big Data market reached $16.9 billion in 2015 with a compound annual growth rate of 39.4%, about seven times the one of the overall ICT market [2].

From the technological perspective, MapReduce is capable of analyzing very efficiently large amounts of unstructured data, i.e., it is a viable solution to support both the variety and volume requirements of Big Data analyses [22]. MapReduce has been adopted in multiple application domains, e.g., machine learning, graph processing, and data mining [34], and its open source implementation, Hadoop 2.x, recently introduced a wide set of performance enhancements (e.g., SSD support, caching, and I/O barriers mitigation). IDC estimates that Hadoop touched half of the world data last year [20] supporting both traditional batch and interactive data analysis applications [32].

In this context, one of the main challenges [25, 33] is that the execution time of a MapReduce job is generally unknown in advance. Because of this, predicting the execution time of Hadoop jobs is usually done empirically through experimentation, requiring a costly setup [15]. An alternative is to develop models for predicting performance. Models may be used to support design-time decisions during the initial development and deployment of Big Data applications. For example, design-time models can help to determine the appropriate size of a cluster or to predict the budget required to run Hadoop in public Clouds (a trending scenario, since by 2020 nearly 40% of Big Data analyses will be supported by public Clouds [2]). Models can also be kept alive at run-time and lead the dynamic adjustment of the system configuration [6, 30], for instance to cope with workload fluctuations or to reduce energy costs.

Unfortunately, modeling the performance of such systems is very challenging. Indeed, production Hadoop environments are nowadays very large massively parallel systems where map and reduce tasks coordinate exhibiting precedence constraints and strict synchronization barriers. Morever, with Hadoop 2.x, resources are dynamically allocated between the map and reduce stages. Additionally, in our context, the stakeholders interested in the performance evaluation of Hadoop processes are its users rather than its developers. Therefore, the complexity and novelty of these systems together with the lack of full knowledge of their development details make unclear the concepts that should be included in a performance model in order to be both accurate and manageable by performance evaluation tools.

Our focus in this paper is to provide design-time performance analysis models to estimate MapReduce jobs execution time in Hadoop clusters governed by the YARN Capacity Scheduler. This work combines real experimentation and model-based evaluation exploring different properties of the MapReduce process. This exploration is used to unveil the characteristics of the YARN Capacity Scheduler that have the highest influence in its performance and therefore should be represented in the models used for a model-based performance evaluation. We propose queueing network (QN) models and stochastic well formed nets (SWNs) of incremental complexity and accuracy able to estimate MapReduce job execution times for multiple users and under unreliable resources. In particular, we analyze a Cloud-based scenario where the cluster, to save execution costs, includes also spot virtual machines (VMs) [14]. The utilization of spot VMs offers large discounts in VM prices, with the drawback of a non-guaranteed availability level. We combine the performance and availability dynamics of Cloud resources in a single *performability* model that allows us to evaluate how failures caused by a sudden increase in the price of spot resources by the Cloud provider, which entails a deallocation of VMs, degrade the system performance.

We evaluate the accuracy of our models on real systems by performing experiments based on the TPC-DS industry benchmark for business intelligence data warehouse applications. Amazon EC2 and the CINECA Italian supercomputing center have been considered as target deployments.

QN and SWN model simulation results and experiments performed on real systems have shown that the accuracy we can achieve is within 30% of the actual measurements in the worst case. With respect to previous literature works, to the best of our knowledge ours is one of the first contribution able to study the performance of Hadoop-2.x-based clusters, where the dynamic allocation of resources between map and reduce stages makes the performance analysis much more challenging.

This paper is organized as follows. Section 2 presents our novel proposals for Hadoop modeling, via both QNs and SWNs. Next, Section 3 reports some experimental results to validate and study the properties of our models. In Section 4 we compare our work with other proposals available in the literature and finally draw the conclusions in Section 5.

2 Modeling Hadoop 2.x Applications Performance

In this paper we provide performance models of incremental complexity and accuracy, ranging from QNs to SWNs, able to estimate Hadoop 2.x jobs performance under a number of scenarios of interest.

Modeling the performance of Hadoop 2.x clusters is challenging since, differently from the previous release where resources, i.e., CPU *slots*, were statically split for mappers and reducers, in the latest Hadoop containers are assigned dynamically among ready tasks, leading to a better cluster utilization. Modeling the dynamic assignment of the available resources is the main contribution of this paper. In particular, we focus on clusters governed by the Capacity Scheduler, which allows for partitioning the cluster among multiple customers through queues, each queue being regulated by a FIFO policy. In the following we assume that queues are partitioned and hence we can focus on single class systems (actually, the Capacity scheduler provides for queues to borrow resources when some others are empty, however this scenario is left as part of our future work).

The parallel execution of multiple tasks within higher level jobs is usually modeled in the QN literature with the concept of fork/join: jobs are spawned at a fork node in multiple tasks, which are then submitted to queueing stations modeling the available servers. After all the tasks have been served, they synchronize at a join node. Unfortunately, there is no known closed-form solution for fork-join networks with more than two queues, unless a special structure exists [24]. Hence, the performance metrics of such networks must be computed by considering the Markov Chain underlying the QN, which represents the possible states of the system [24]. However, such approaches are not fit for Hadoop systems, since the state space grows exponentially with the number of tasks [13, 27], in the order of thousands in realistic MapReduce jobs. For this reason, a number of approximation methods have been proposed. In particular, [29] proposed a good approximation technique that, however, is based on service time exponential distribution, which is not the case for Hadoop deployments. Our initial experiments showed that mapper and reducer times follow general distributions, which can be approximated by phase type or in some cases Erlang. Under exponential time hypothesis, the relative error observed in our simulations was around 50–60%. Some other approaches, e.g., [24], are based on an approximate mean value analysis technique and use an iterative hierarchical approach. Along the same lines, [34] combines a precedence graph and a QN to capture the intra-job synchronization constraints, thus being able to estimate the synchronization delays introduced by the communication among mappers and reducers. Unfortunately, even if the approach is rather accurate (around 15% accuracy on real systems), the authors assume that CPU slots are statically assigned to mappers and reducers, hence the proposed method cannot be adopted to estimate performance under the Hadoop 2.x dynamic resource assignment policy.

For this reason, we developed simulation models based on the concept of finite capacity region (FCR) available in modern QN simulators [10]. Unfortunately, QN models capture the behavior of Hadoop 2.x with some approximations. In Section 2.2 we rely on SWNs and provide a model capturing the behavior of real Hadoop 2.x systems. Moreover, we investigate an advanced Cloud-based scenario where some resources are provided by unreliable spot instances and we evaluate the performance of jobs in case of failure.

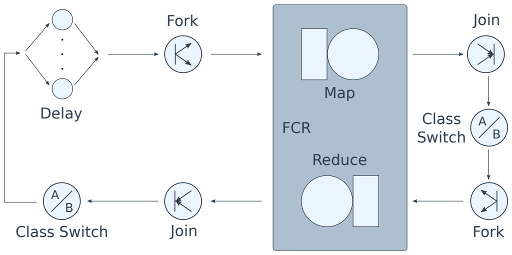


Fig. 1: Queueing network model

2.1 Queueing Network Model

This section discusses a proposal of QN model for MapReduce applications running upon YARN Capacity Scheduler.

The performance model is depicted in Fig. 1. It is a closed QN model where the number of concurrent users is given by H and they start off in the delay center, characterized by the average think time Z. When a user submits her job, this is forked into as many map task requests as stated in the job profile, which then enter the FCR. FCRs model situations where several service centers access resources belonging to a single limited pool, competing to use them. Hence, the FCR enforces an upper bound on the total number of requests served at the same time within itself, allowing tasks to enter according to a FIFO policy, but also supporting prioritization of different classes. The FCR includes two multi-service queues that model the map and reduce execution stages. The FCR and multi-service queues capacities are equal to the total number of cores available in the cluster. In this way, we can model the dynamic assignment of YARN containers to map and reduce tasks whenever they are ready. Map tasks are executed by the first multi-service queue and synchronize after completion by joining back to a single job request; the reduce phase is modeled analogously. Note that the map join is external to the FCR in order to model that when map tasks complete they release container cores, which can be assigned to tasks ready in the FCR FIFO queue. Moreover, the reduce fork is also external to the FCR to model correctly applications characterized by a number of reducers larger than the total cluster capacity.

YARN Capacity Scheduler implements a FIFO scheduling policy within the same queue and containers are allocated to the next job only after all reduce tasks have obtained their resources. The class switches present in the QN are meant to enforce that reduce tasks waiting for resources obtain them with priority. Despite this, the model in Fig. 1 is still an approximation: notwithstanding the higher priority associated to reducers, subsequent users' mappers can still occupy part of the servers available in the FCR when the preceding job has an overall number of map tasks that is not multiple of the cluster capacity. In such a case, the last map wave leaves room for serving further requests, hence the following user can overtake part of the capacity and the reduce stage of the first user will not start processing at full capacity until those mappers complete.

As a final consideration, note that the discussed model is rather general and can be easily extended to consider also Tez or Spark applications, where a Tez directed acyclic graph node or Spark stage is associated to a corresponding multi-server queue.

2.2 Stochastic Well Formed Net Models

The stochastic well formed net (SWN) in Fig. 2 is able to capture completely the behavior of the Capacity Scheduler policy. Jobs execution is modeled by a closed workload, where the $nU1$ users compete to access the cluster and cycle between demanding to execute the MapReduce scenario (subnet in the dotted rectangle), and spending an external delay period between the end of one response and the next request (mean firing time of the *think* transition). The basic color class *User* consists of a single subclass *User1* and the job identities are captured by assigning a token of different color to each job $u_1, \dots, u_{nU1}$.

To enforce the FIFO scheduling each job is assigned an ID i. The initial marking *M3* of the place *IDs1* is set to the first index of the color class *ID*, and once transition *think* sends a job to the ready state it increases this index by one. The transition *generateMaps* will start the job that has the index equal to the one it is getting from place *IDs2*. In other words, the job that has its turn will start the Map phase. Whenever a job gets resources for all of its reduce tasks (place *wait4ResRed* drains), the job with the next index will be started thanks to the transition *startNext*, which updates the *IDs2* place with the next index.

When a job x is ready to be processed and it has its turn—i.e., the place *jobReady* is marked with a token $\langle x, i \rangle$ and the *IDs2* place is marked with the same index i—nM map tasks are generated (firing of *generateMaps* transition). Such tasks, associated to job x, are represented by nM pairs $\langle x, t \rangle$, where the color t belongs to the subclass *Map* of the basic color class *Task*. Each task $\langle x, t \rangle$ needs to acquire a resource r to be executed (firing of *getResMap* transition) and map tasks can be concurrently executed according to resource availability. The set of resources is defined by the basic color class *Resource*, which consists of a unique partition *Core* including nC resources. The timed transition *map* models the duration of the map task execution and its firing time is an Erlang-distributed random variable. The map stage is finished when all the map tasks $\langle x, t \rangle$ associated to job x have been executed: the firing of the *joinMaps* transition models the beginning of the next processing step, where nR reduce tasks are generated. The reducing step is similar to the mapping step, the only difference is that the reduce tasks, associated to job x, are represented by nR pairs $\langle x, t \rangle$ where the color t belongs to the subclass *Reduce* of the basic color class *Task*. Finally, observe that the map tasks $\langle y, t \rangle$, associated to a job y, are generated when all the reduce tasks $\langle x, t \rangle$, associated to the previous job x, are not waiting for resource availability. This condition is modeled by the inhibitor arc inscription from place *wait4ResRed* to transition *startNext*.

The models considered so far can capture the behavior of MapReduce systems running on an enterprise infrastructure or public Clouds based on standard

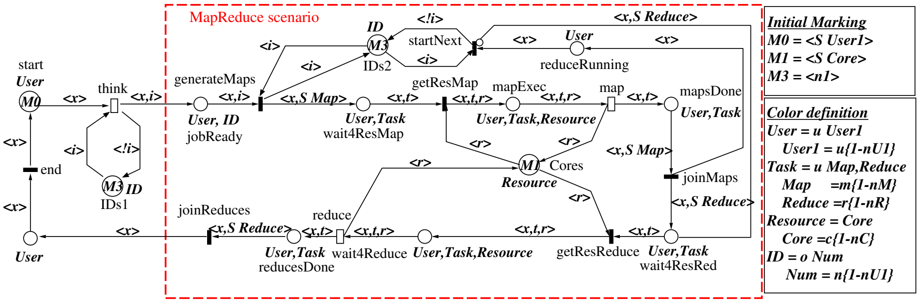


Fig. 2: Basic SWN model

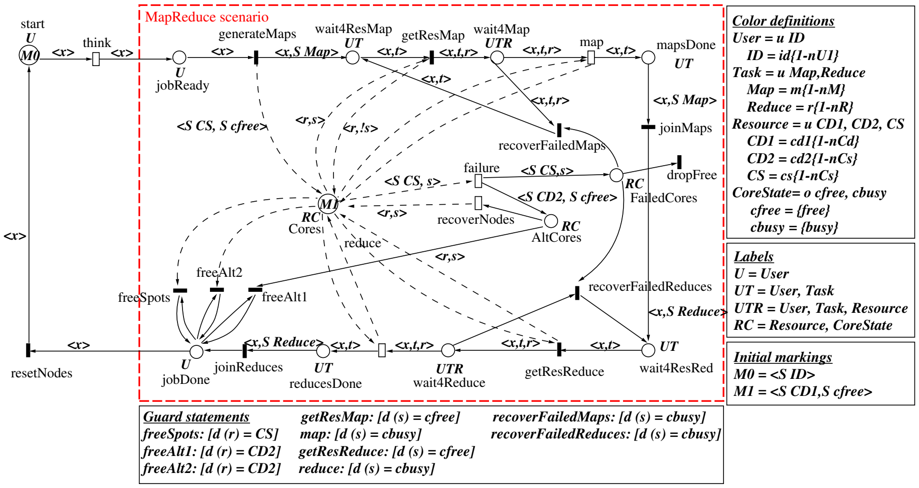


Fig. 3: SWN model with spot resources

resources. Cloud providers (see, e.g., Amazon EC2) offer another type of resource loaning, *spot*, in which customers bid for the instance price. The VM instances are billed the spot price, which is set by the infrastructure provider and fluctuates periodically depending on current energy costs, which vary throughout the day, and also on the overall supply and demand of virtual resources. Spot instances are usually available at a competitive price. However, if the provider raises the spot price above user's bid while she is using spot instances, these can be arbitrarily terminated without notice. Hence, on one side spot resources are an opportunity for lowering the execution cost of MapReduce applications, but on the other side they introduce availability threats.

The model introduced in Fig. 3 allows for exploring different configurations and to evaluate MapReduce jobs performance degradation in case of spot instances termination. In particular, we assume that once spot VMs fail, a monitoring component replaces them with the same number of on-demand resources. Since we are interested in the performance degradation of the job currently in execution, we can drop the FIFO mechanism from the model of Fig. 2, i.e., the places *IDs1*, *IDs2*, and *reduceRunning*; the transition *startNext*; and the color class *ID*. The model in Fig. 3 introduces also a new color class *CoreState*, which consists of two singleton subclasses *cfree* and *cbusy*, to record the state of a node. To enable this, the color domain of the place *Cores* is set to the Cartesian product $Resource \times CoreState$. This change enables the place *Cores* to track the occupied cores as well as the free cores. Transitions *getResMap* and *getResReduce* are modified to change the core status from "free" to "busy" whenever they own a core, and vice versa for transitions *map* and *reduce*. The color class *Resource* is enriched with two new subclasses *CoreSpot (CS)* and *CoreDemand2 (CD2)*, which model spot instances and the on-demand nodes triggered to replace spot resources. The color definition *Core* is renamed to *CoreDemand1 (CD1)* to model the on-demand nodes that are initially started together with spot resources. The places *FailedCores* and *AltCores* are added to identify the failed nodes and the alternative nodes waiting for recovery. The timed transition *failure* takes all the spot instances from the available nodes with a rate proportional to the failure probability. When spot instances fail due to a low bid, it takes at most a YARN heartbeat for the monitoring component to figure out the loss. After this short delay, the replacing process starts with acquiring on-demand nodes. As soon as the new on-demand VMs are ready with running NodeManagers, they have to be registered with the ResourceManager in order to be used by the running job. We summed up all these delays and introduced the timed transition *recoverNodes*, characterized by an appropriate rate. Moreover, we included three instantaneous transitions (*recoverFailedMaps*, *recoverFailedReduces*, and *dropFree*) to move a failed map or reduce task to the waiting list and to drop the failed spot nodes that were not occupied by any task. Since our goal is to evaluate the average performance of the job when a failure happens (note that a failure can occur anytime between the start and end of the job execution and in the latter case the job might complete before new on-demand VMs become available), we have to create the same environment for every successive run of the job. Then, three transitions *freeAlt1*, *freeAlt2*, and *freeSpots* are added to free the places *AltCores* and *Cores* from any residual *CD2* and *CS* at the end of the job execution and one outgoing arc is connected to the transition *generateMaps* to put back the spot cores in the place storing available nodes. As a result, spot failures can happen again while the simulator is running the next iteration. In this way we ensure that every job submission is subject to failure and obtain relevant statistical results.

3 Experimental Analysis and Validation

The models presented in the previous sections have been validated by performing an experimental campaign on Amazon EC2 and CINECA, the Italian supercomputing center. The target version was Hadoop 2.6.0. The Amazon cluster included 30 m4.xlarge instances with a total of 120 vCPUs configured to support 240 containers overall. On PICO[1], the Big Data cluster available at CINECA, we used several configurations ranging from 40 to 120 cores and set up the scheduler to provide one container per core.

The dataset used for testing has been generated using the TPC-DS benchmark[2] data generator, creating at a scale factor ranging from 250 GB to 1 TB several files directly used as external tables by Hive. We chose the TPC-DS benchmark as it is the industry standard for benchmarking data warehouses. We used the GreatSPN [7] and JMT [10] tools with 10% accuracy and 95% confidence interval for the performance analysis of the SWN and QN models, respectively. Next, we performed experiments on five Hive queries, dubbed R1–5 and shown in Fig. 4. The profiling phase has been conducted on a dedicated cluster, extracting average task durations from at least twenty runs of each query. The numbers of map and reduce tasks varied, respectively, in the ranges $(4, 1560)$ and $(1, 1009)$. Parsing Hadoop logs it is also possible to obtain lists of task execution times, which are needed for the replayer in JMT service centers. These logs are also used to choose a proper distribution with right parameters for the *map* transition in the SWN models. As discussed earlier in Section 2, the execution time of map tasks fits better with more general distributions, like Erlang (in particular we used Erlang-2 for R1, Erlang-4 for R2 and R3 and Erlang-5 for R4 and R5). The shape and rate parameters are set according to each query profile. The other timed transitions appearing in the SWN models are considered to be exponentially distributed.

3.1 QN and SWN Models Validation

To start off with, we show results for the validation of the QN and SWN models discussed in Section 2. We feed the models with parameters evaluated via the experimental setup and compare the measured response times with the simulated ones. Specifically, we consider as a quality index the accuracy on

[1] http://www.hpc.cineca.it/hardware/pico

[2] http://www.tpc.org/tpcds/

```
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) R1

```
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;
```

(b) R3

```
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;
```

(c) R2

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

(d) R4

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

(e) R5

Fig. 4: Interactive queries

response time prediction, defined as $\vartheta = (\tau - T)/T$, where τ is the simulated response time, whilst T is the average measured one.

Among these experiments, we considered both single user scenarios, repeatedly running the same query on a dedicated cluster with $Z = 10$ s, and multiple user scenarios.

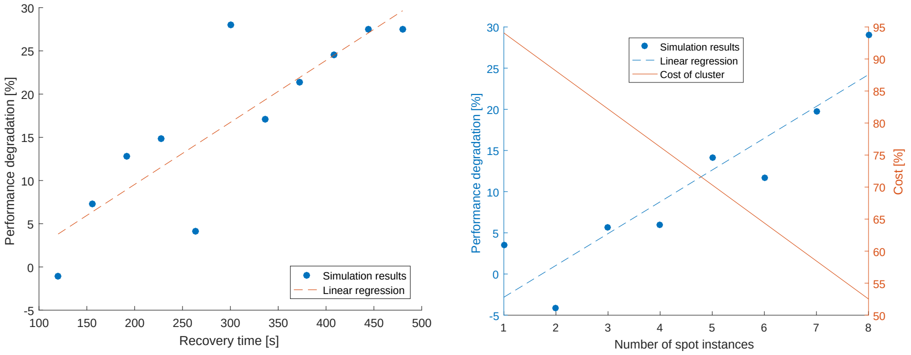
Table 1 shows the results of the QN and SWN models validation. For all the experiments we report the number of concurrent users, the overall cores available in the cluster, the dataset scale factor, and the total number of map and reduce tasks, plus the above mentioned metric. In the worst case, the relative error can reach up to 32.97%, which is in line with the expected accuracy in the performance prediction field [23]. Moreover, the SWN model achieves a higher accuracy, with the average relative error decreasing from the 14.13% of QNs down to 9.08%.

3.2 Spot Failure Analysis

To evaluate the SWN model of Fig. 3 we considered query R1 running on the 1 TB dataset with 15 VMs, 4 cores each. Without failures, the execution time of R1 is 556680 ms, where around 57% of the time is spent in the map stage. The baseline simulation time is $\tau_0 = 533438$ ms, yielding a -4.18% relative error. We define the performance degradation as$\eta\,(t) = (\tau\,(t) - \tau_0)/\tau_0$, where $\tau\,(t)$ is the simulated response time obtained via the SWN model in Fig. 3 when the recovery time is t. To measure the recovery time of our system on Amazon EC2, we switched off one of the VMs, started a new VM after 30 s and evaluated the

Table 1: QN and SWN models accuracy

Query	Users	Cores	Scale [GB]	n^M	n^R	T [ms]	τ_{QN} [ms]	ϑ_{QN} [%]	τ_{SWN} [ms]	ϑ_{SWN} [%]
R1	1	240	250	500	1	55410	50753.34	−8.40	50629.58	−8.63
R2	1	240	250	65	5	36881	27495.31	−25.45	37976.82	2.97
R3	1	240	250	750	1	76806	77260.03	0.60	83317.27	8.48
R4	1	240	250	524	384	92197	78573.96	−14.72	89426.51	−3.01
R1	1	60	500	287	300	378127	411940.93	8.94	330149.74	−12.69
R3	1	100	500	757	793	401827	524759.36	30.59	507758.68	26.36
R3	1	120	750	1148	1009	661214	759230.77	14.82	698276.75	5.61
R4	1	60	750	868	910	808490	844700.85	4.48	806366.51	−0.26
R3	1	80	1000	1560	1009	1019973	1053829.78	−1.00	1020294.84	0.03
R5	1	80	1000	64	68	39206	36598.32	−6.65	38796.47	−1.04
R1	3	20	250	144	151	1002160	1038951.05	3.67	909217.89	−9.27
R1	5	20	250	144	151	1736949	1215490.20	−30.02	1428894.40	−17.74
R2	3	20	250	4	4	95403	112050.45	17.45	99219.94	4.00
R2	5	20	250	4	4	145646	97619.46	−32.97	88683.10	3.09
R1	5	40	250	144	151	636694	660241.29	3.70	613577.53	−3.63
R2	3	40	250	4	4	86023	105785.41	22.97	119712.30	−17.81
R2	5	40	250	4	4	90674	103173.38	13.78	117582.82	29.68



(a) Against recovery time

(b) Varying the number of spot instances

Fig. 5: Performance degradation and cost reduction

time instant when the first task is assigned for execution to the new instance. In ten experiments, the mean recovery time was equal to 331715 ms, where the time to start the new instance was around 180 s.

In the first analysis we consider a conservative scenario where the failure is injected into the system in an early stage of query execution and we fixed the mean time to failure to 50 s. We considered a cluster with 7 spot VMs out of 15 and we varied the recovery time between 120 and 480 s. In this way we estimate system performance degradation in a range where the VMs startup is either faster (e.g., in container-based systems where the startup time is negligible and the recovery time is due only to YARN NodeManagers startup) or slower than on Amazon EC2. The simulation reported in Fig. 5a shows, as expected, that the more the recovery process stalls, the more performance degrades, and it can reach up to 25% if the recovery process takes 8 minutes.

Note that, in some cases we obtained negative values for $\eta(t)$ because simulation data is subject to inaccuracies. This is why we rely on linear regression to estimate the performance degradation trend. In this and the next analysis we obtained a p-value equal to 0.001 for the full model F-test, meaning that the regression line is a good fit except for white noise.

In the second experiment, the mean recovery time is fixed to 331715 ms, the average we measured in our experiments, and the number of spot instances is increased from 1 to 7 out of the total of 15 VMs. The result is shown in Fig. 5b, which reports the curve for the cost reduction due to using spot instances as well as the performance degradation. Cost reduction is computed as $C_{\mathrm{r}} = C\left(s;p\right)/C\left(0;0\right)$, where $C(s;p)$ is the cluster cost when s spot instances are used, with a failure probability p. $C\left(0;0\right)$ is the case where no spot instance is used. $C\left(s;p\right)$ can be computed as:

$$C\left(s;p\right) = \delta\left(R - s\right) + ps\delta + \left(1 - p\right)s\sigma \tag{1}$$

where R is defined to be the total number of VMs, while δ and σ are hourly prices for on-demand and spot instances, respectively. δ and σ are set to \$0.285 and \$0.0313, respectively. The latter is the average spot price from the Amazon website in the last two weeks at the time of writing.

It should be noted that, according to the experiments, the job execution time is less than an hour: according to the Amazon pricing policy [1], in case spot instances are abruptly terminated users are not charged for partial hour usage. The first term on the right hand side of Equation 1 is the cost of the initial on-demand instances, while the second term is the price the user should pay to acquire the same number of on-demand instances as the lost spot ones in case of failure. Finally, with probability $(1 - p)$ spot instances do not fail and the third term indicates the cost in this scenario. We evaluate the probability of spot termination as $p = \mathbb{P}\left(X < T\right)$, where X is the random variable for time to failure and T is the execution time. Assuming X is exponentially distributed, p can be easily obtained in closed form. Fig. 5b shows that, using the proposed SWN model, one can efficiently use spot instances to decrease cluster costs, down to 52.5% for 8 spot instances with just a 25% performance degradation.

4 Related Work

Deploying a task in a computer cluster requires a non-negligible learning curve of the underlying technology and a good knowledge of the process requirements. It is essential to consider that clusters are shared by multiple users or institutions, are vulnerable to hardware failures and have a monetary cost. The minimization of the starvation between user jobs, the impact of the execution errors and the optimization of operational costs are important issues.

Modeling and simulating the configuration of a high-performance distributed computer framework allows for predicting the behavior of the tasks before execution. They enable the detection of potential problems such as bottlenecks, the tolerance to hardware malfunctioning as well as a finer estimation of the resources usage, the running time and throughput (i.e., time and resources are two of the main parameters for guaranteeing a fair scheduling among user jobs and inferring the billing). Initial works for studying generic Cloud systems have been already proposed. For example, Bruneo et al. [11] introduce a Stochastic Reward Petri net model representing an Infrastructure as a Service Cloud where the load conditions can change dynamically, although it has not been validated with real data, yet. More concretely, analytical models such as [4, 16, 21, 26, 35] use mathematical equations for representing a MapReduce system and quantifying the throughput of a particular resource (i.e., network, hardware disk, or CPU). In this paper, we propose two general purpose modeling abstractions for describing the MapReduce environment such as SWNs and QNs. Regression techniques for estimating the response time of future jobs based on past experiences are exploratory approaches not considered in this paper.

Several works describe the adoption of Petri nets (PNs) for MapReduce modeling. For instance, Castiglione et al. [12] describe a Big Data architecture based on Hadoop by means of stochastic PNs and apply Mean Field Analysis to obtain average metrics and estimate its performance. Another approach, presented by Barbierato et al. [8], exploits Generalized Stochastic Petri Nets alongside other formalisms such as process algebras or Markov chains to develop multi-formalism models and capture HQL queries. Adopting the presented tool, the authors investigate how performance depends on some configuration parameters. In literature colored PNs have also been adopted to assess the feasibility of a distributed file system project [3]. The authors design a deployment of HDFS exploiting spare resources in a cluster of workstations available for teaching in their university, so as to provide a sufficiently available distributed file system. PNs are used to assess system availability in a number of configurations of interest. More recently, Ruiz et al. [31] formalize the MapReduce paradigm using Prioritized Timed Colored Petri Nets to obtain complete and unambiguous models of the system behavior. They evaluate the correctness of the system and carry out a trade-off analysis of the number of resources versus processing time and resource cost with CPNTools [18]. Further works with PNs and MapReduce are oriented to measuring performance under failures [19] or studying the fault tolerance mechanism [28].

On the other side, QNs have also been introduced for modeling Cloud systems. Bardhan and Menascé [9] apply QN models for predicting the completion time of the map phase of MapReduce jobs within simple configurations of Hadoop. Alipour et al. [5] develop a Cloud provider independent model with QNs that represents entities involved in the Hadoop MapReduce phases, and customize it for a specific Cloud deployment. Finally, in Yu and Li [36], an analytical queueing mode has been developed to investigate the utilizations and mean waiting times of mappers and reducers, respectively.

Previous works are able to model Hadoop 1.0 clusters with static resource allocation at different levels of detail. Our SWN or QN models are able to capture the dynamic assignment of YARN resource containers and allow for estimating performance in new Hadoop 2.x clusters.

5 Conclusions

In this paper we proposed SWN and QN models for the performance prediction of MapReduce applications running on clusters governed by the Capacity Scheduler. Our preliminary results have shown how our simulation models are effective in capturing the dynamic resource assignment implemented in Hadoop 2.x and can achieve 9% accuracy, thus making them suitable for capacity planning decisions at design-time. In our future work we plan to extend our models to cope with multiple classes in shared clusters governed also by the Fair Scheduler and with job preemptions. Finally, we will embed the models into a design space exploration tool for Cloud resources cost minimization.

References

[1] Amazon EC2 pricing, `http://aws.amazon.com/ec2/pricing/`

[2] The digital universe in 2020, `http://idcdocserv.com/1414`

[3] Aguilera-Mendoza, L., Llorente-Quesada, M.T.: Modeling and simulation of Hadoop Distributed File System in a cluster of workstations. In: Model and Data Engineering, vol. 8216, pp. 1–12 (2013)

[4] Ahmed, S.T., Loguinov, D.: On the performance of MapReduce: A stochastic approach. In: IEEE International Conference on Big Data. pp. 49–54. IEEE (2014)

[5] Alipour, H., Liu, Y., Gorton, I.: Model driven performance simulation of cloud provisioned Hadoop MapReduce applications

[6] Ardagna, D., Ghezzi, C., Mirandola, R.: Rethinking the use of models in software architecture. In: QoSA 2008 Proceedings

[7] Baarir, S., Beccuti, M., Cerotti, D., De Pierro, M., Donatelli, S., Franceschinis, G.: The GreatSPN tool: Recent enhancements. ACM SIGMETRICS PER 36(4), 4–9

[8] Barbierato, E., Gribaudo, M., Iacono, M.: Modeling Apache Hive based applications in Big Data architectures. In: VALUETOOLS 2013 Proceedings

[9] Bardhan, S., Menascé, D.: Queuing network models to predict the completion time of the map phase of Mapreduce jobs. In: Proceedings of the Computer Measurement Group International Conference (2012)

[10] Bertoli, M., Casale, G., Serazzi, G.: JMT: Performance engineering tools for system modeling. SIGMETRICS Perform. Eval. Rev. 36(4), 10–15 (2009)

[11] Bruneo, D., Longo, F., Ghosh, R., Scarpa, M., Puliafito, A., Trivedi, K.S.: Analytical modeling of reactive autonomic management techniques in IaaS clouds. In: IEEE CLOUD 2015 Proceedings

[12] Castiglione, A., Gribaudo, M., Iacono, M., Palmieri, F.: Exploiting mean field analysis to model performances of Big Data architectures. Future Generation Computer Systems 37, 203–211 (2014)

[13] Chu, W.W., Sit, C.M., Leung, K.K.: Task response time for real-time distributed systems with resource contentions. IEEE Trans. Softw. Eng. 17(10), 1076–1092

[14] Dubois, D.J., Casale, G.: OptiSpot: Minimizing application deployment cost using spot Cloud resources. Cluster Computing pp. 1–17 (2016)

[15] Gibilisco, G.P., Li, M., Zhang, L., Ardagna, D.: Stage aware performance modeling of DAG based in memory analytic platforms. In: Cloud (2016)

[16] Herodotou, H.: Hadoop performance models (2011)

[17] Jagadish, H.V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J.M., Ramakrishnan, R., Shahabi, C.: Big Data and its technical challenges. Commun. ACM 57(7), 86–94 (Jul 2014)

[18] Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer 9(3-4), 213–254 (2007)

[19] Jin, H., Qiao, K., Sun, X.H., Li, Y.: Performance under failures of MapReduce applications. In: CCGrid 2011 Proceedings

[20] Kambatla, K., Kollias, G., Kumar, V., Grama, A.: Trends in Big Data analytics. Journal of Parallel and Distributed Computing 74(7), 2561–2573 (2014)

[21] Krevat, E., Shiran, T., Anderson, E., Tucek, J., Wylie, J.J., Ganger, G.R.: Applying performance models to understand data-intensive computing efficiency. Tech. rep., DTIC Document (2010)

[22] Laney, D.: 3D data management: Controlling data volume, velocity, and variety. Tech. rep., META Group (2012)

[23] Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative System Performance. Prentice-Hall (1984)

[24] Liang, D.R., Tripathi, S.K.: On performance prediction of parallel computations with precedent constraints. IEEE Trans. Parallel Distrib. Syst. 11(5), 491–508

[25] Lin, M., Zhang, L., Wierman, A., Tan, J.: Joint optimization of overlapping phases in MapReduce. SIGMETRICS Performance Evaluation Review 41(3), 16–18 (2013)

[26] Lin, X., Meng, Z., Xu, C., Wang, M.: A practical performance model for Hadoop MapReduce. In: Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on. pp. 231–239. IEEE (2012)

[27] Mak, V.W., Lundstrom, S.F.: Predicting performance of parallel computations. IEEE Trans. Parallel Distrib. Syst. 1(3), 257–270 (Jul 1990)

[28] Marynowski, J.E., Santin, A.O., Pimentel, A.R.: Method for testing the fault tolerance of MapReduce frameworks. Computer Networks 86, 1–13 (2015)

[29] Nelson, R.D., Tantawi, A.N.: Approximate analysis of fork/join synchronization in parallel queues. IEEE Trans. Computers 37(6), 739–743 (1988)

[30] Polo, J., Becerra, Y., Carrera, D., Steinder, M., Whalley, I., Torres, J., Ayguadé, E.: Deadline-based MapReduce workload management. IEEE Trans. Network and Service Management 10(2), 231–244 (2013)

[31] Ruiz, M.C., Calleja, J., Cazorla, D.: Petri nets formalization of Map/Reduce paradigm to optimise the performance-cost tradeoff. In: Trustcom/BigDataSE/ISPA, 2015 IEEE. vol. 3, pp. 92–99. IEEE (2015)

[32] Shanklin, C.: Benchmarking Apache Hive 13 for enterprise Hadoop, https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

[33] Verma, A., Cherkasova, L., Campbell, R.H.: ARIA: Automatic resource inference and allocation for MapReduce environments. In: ICAC 2011 Proceedings

[34] Vianna, E., Comarela, G., Pontes, T., Almeida, J.M., Almeida, V.A.F., Wilkinson, K., Kuno, H.A., Dayal, U.: Analytical performance models for MapReduce workloads. International Journal of Parallel Programming 41(4), 495–525 (2013)

[35] Yang, X., Sun, J.: An analytical performance model of MapReduce. In: CCIS 2011

[36] Yu, X., Li, W.: Performance modelling and analysis of MapReduce/Hadoop workloads. In: LANMAN 2015 Proceedings