Fluid Petri Nets for the Performance Evaluation of MapReduce Applications

Eugenio Gianniti[*], Alessandro Maria Rizzi[†], Enrico Barbierato[‡], Marco Gribaudo[§] and Danilo Ardagna[¶]

Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Via Golgi 42 20133 Milan, Italy

Abstract

Big Data applications allow to successfully analyze large amounts of data not necessarily structured, though at the same time they present new challenges. For example, predicting the performance of frameworks such as Hadoop can be a costly task, hence the necessity to provide models that can be a valuable support for designers and developers. This paper provides a new contribution in studying a novel modeling approach based on fluid Petri nets to predict MapReduce jobs execution time.

The experiments we performed at CINECA, the Italian supercomputing center, have shown that the achieved accuracy is within 16% of the actual measurements on average.

Keywords: Map Reduce, Hadoop, fluid Petri nets.

1 Introduction

The implementation of Big Data applications is steadily growing today [12]. According to recent analysis, 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 [1].

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 [14]. MapReduce has been adopted in multiple application domains, e.g., machine learning, graph processing, and data mining [25]. 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 [13], supporting both traditional batch and interactive data analysis applications [23].

In this context, one of the main challenges [17, 24] 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 [10]. In alternative, it is possible 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 or at run-time to trigger system reconfiguration. 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 pub-

[*] eugenio.gianniti@polimi.it
[†] alessandromaria.rizzi@polimi.it
[‡] enrico.barbierato@polimi.it
[§] marco.gribaudo@polimi.it
[¶] danilo.ardagna@polimi.it

lic Clouds (a trending scenario, since by 2020 nearly 40% of Big Data analyses will be supported by public Clouds [1]). Models can also be kept alive at run-time and lead the dynamic adjustment of the system configuration [3, 22], for instance to cope with workload fluctuations or to reduce energy costs.

While in early Hadoop versions CPU *slots* and other resources were separated between mapper and reducers using a static approach, in Hadoop 2.x containers are distributed among ready tasks in a dynamic fashion. On the one hand, this allows a better cluster utilization, on the other hand modeling the performance is not a task of negligible difficulty.

The originality of this paper consists of a new modeling technique concerning the dynamic assignment of the available cluster resources. We assume that the cluster is governed by the Capacity Scheduler, which partitions the available resources among multiple customers through queues, each queue being regulated by a FIFO policy.

This work proposes a fluid Petri Nets model to estimate MapReduce jobs execution time, combining real experimentation and model-based evaluation and exploring different properties of the MapReduce process 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.

The accuracy of the model is evaluated on real systems by performing experiments based on the TPC-DS industry benchmark for business intelligence data warehouse applications. The Italian supercomputing center, CINECA, has been considered as target deployment.

Results and experiments performed on real systems have shown that the achieved accuracy is within 16% of the actual measurements on average. With respect to previous literature, to the best of our knowledge this article is one of the first contributions 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 compares this work with other proposals available in the literature; Section 3 presents our novel proposals for Hadoop modeling, via fluid Petri Nets. Next, Section 4 reports some experimental results to validate and study the properties of our models. Finally, Section 5 draws the conclusions, providing the lines for future work.

2 Related Work

The literature provides a large number of performance studies for Hadoop 1.x, since the framework has been widely adopted in the ICT industry, often supporting core business activities. Two main approaches have been explored: i) simulation-based models implement the single constituents of Hadoop and of the job, replaying in a simulated environment the steps and delays of the real system; ii) analytical models, on the other hand, define a mathematical representation of those constituents, avoiding the costs of running multiple simulations. Both approaches make use of information such as input dataset size, cluster resources, and Hadoop specific parameters. The computational effort and the time spent in running simulation-based models make them hardly fit for the purposes of runtime cluster management: thus, we hereby consider only analytical models, according to the focus of our paper.

The authors of [24] propose the ARIA framework for estimating the makespan of jobs in MapReduce clusters. This approach relies on information extracted from the logs of previous executions of similar jobs. Adopting scheduling techniques, the authors prove lower and upper bounds on makespans. From these results, they derive formulae for performance prediction. They obtain both a conservative estimate, suitable for hard deadlines, and an alternative that does not offer guarantees of meeting deadlines, but boasts a relative error below 10% in the validation against measured timings.

Another performance model estimating the execution time by considering the single costs of the various phases of a MapReduce job is described in [18]. In this work, the authors go down to the very low level elements that determine the cost of single job phases, writing a 37-parameter model that provides execution times within 10% of those measured in a real cluster. Even with such an accurate model, the validation considers just single job executions.

In queueing network (QN) literature, the fork/join paradigm (see [21] and [4]) is used to denote the modeling of the concurrent execution of many tasks within higher level jobs. Specifically, this approach operates through two steps: i) jobs are spawned at a fork node in multiple tasks, then ii) they are submitted to queueing stations that, in turn, model the available servers.

Once all the tasks have been served, they can synchronize at a join node. It has to be noted that when a fork-join network has more than two queues, a closed-form solution is not possible. That said, it is possible to mitigate the issue by using a special kind of structure, as shown in [16] which considers the Markov Chain underlying the QN representing the possible states of the system. Unfortunately, as noted in [9, 19], the state space grows exponentially when the tasks number corresponds to realistic MapReduce jobs—in the order of thousands—thus making the above approaches unsuitable.

An interesting alternative in the shape of approximation methods is introduced in [20], which proposes a method based on exponentially distributed service times, though it is not applicable to Hadoop deployments. Indeed, as per preliminary experiments conducted in this work, it is safe to assume that phase type (or in some cases Erlang) can approximate the general distributions of mapper and reducer times, while assuming an exponential distribution led to around 50–60% relative error on the prediction of the overall job execution time. To this concern, see also [16], where an approximate mean value analysis technique is proposed, by using an iterative hierarchical approach.

Other approaches prefer adopting a MapReduce modeling based on Petri nets (PNs). For example, in [8] the authors use a stochastic PN, applying Mean Field Analysis to calculate average metrics and estimate the performance of a Big Data architecture based on Hadoop.

In order to capture the performance of Hive queries, Generalized Stochastic Petri Nets together with other formalisms (i.e., process algebras or Markov chains) are used to create multi-formalism models in [5]. Specifically, the authors show how performance can depend on some configuration parameters.

In [2], the authors propose colored PNs to determine the feasibility of a distributed file system project. After designing a deployment of HDFS (using a set of spare resources in a cluster of workstations in order to provide a sufficiently available distributed file system), the system availability is assessed by exploiting PNs.

Though earlier methods exploited static resource allocation (at different levels of detail) to model Hadoop 1.0 clusters, the fluid Petri Net models used here can capture the dynamic assignment of YARN resource containers (for example, by using the model presented in [7]) and are capable of estimating performance of Hadoop 2.x jobs very efficiently.

3 Fluid models

Very often modelers are bound to face several problems when trying to provide a description of a physical system by using a formalism. Typically, these problems are more evident in techniques using discrete states, since the analysis produces a state space explosion, with an exponential growth in the number of states following the complexity of the model. Different alternatives to mitigate this problem have been proposed. Specifically, *hybrid* techniques involving a continuous and discrete part have proved to reduce significantly the severity of the issue. Continuous variables can be exploited in different scenarios: for example, they can be used to represent the rising temperature in a closed room or the water leaking out of a full bucket.

In literature, fluid models have been presented in different ways for what concerns the realization of the continuous aspect. Among the different flavors, in [11], the authors refer to i) Fluid Stochastic Petri Nets (FSPNs), ii) Reward and iii) Fluid models. FSPNs add to Stochastic Petri Nets introducing the ability to handle continuous parts. In reward models, the idea consists in using a Markov chain and associating a Reward rate, a positive weight whose value depends on the time spent in a particular state.

Fluid models can be used to successfully study the MapReduce framework and even more sophisticate ones, such as Spark [26], including paradigms like concurrent programming. In [7], the authors consider a scenario to analyze the energy-related performance metrics by using a broad Markov chain, taking into account different configuration parameters and finally focusing on the solution process that is run in parallel on all the available cores. The completion times are taken as a benchmark. It is noted that all the observed cases show a ladder-like shape, denoting the level of parallelism considered.

In the following, Section 3.1 describes the FSPN used, while Section 3.2 describes how to generate an approximation of the average execution time of jobs with deterministic execution time.

3.1 FSPN Model for a MapReduce Job

In order to formalize the fluid model proposed in this paper, we will describe it using the FSPN shown in Figure 1 (the purpose of this FSPN model is just to specify the underlying fluid model without enumerating its states).

In this formalization, single circles represent *discrete places*, which can hold an integer number of tokens; single boxes represent *timed transitions* that can fire after an exponential amount of time; bars represent *immediate transitions*, which fire as soon as they are enabled; thin lines define *standard arcs* that move tokens among places and enable the connected transitions; and thin lines with circular ends define *inhibitor arcs*, which prevent the corresponding transitions from firing whenever the input place is marked. Double circles represent *fluid places*, that can hold a continuous amount of fluid; double boxes identify *fluid transitions*, that continuously pump fluid in and out of continuous places; double arrows represents *fluid arcs* that can remove fluid from their input places; and thick arrows represents *set arcs*, that immediately insert a given amount of fluid in their destination place when their input transition fires.

The discrete amount of tokens is moved across the discrete arcs as usual. With regard to the fluid places, they include a level denoted by a continuous variable, which flows according to an instantaneous rate. The discrete FSPN component regulates the fluid flow through the continuous part, while the conditions enabling a transition depend on the discrete component only.

The model considers N users (corresponding to the marking of place `Users`) that can submit jobs to the system after a think time Z. The submission of jobs from a user is modeled by the firing of transition `Think` characterized by the infinite server semantic. According to the YARN Capacity Scheduler, we consider that only one job at a time can be executed by the system: this is obtained via place `Available`, which holds a token whenever the infrastructure is ready to run a new job, thus enabling the corresponding `Start` transition. Both the map and reduce phases are modeled by a similar sub-network. Place `Ready` holds a token whenever the phase is ready to start, and its beginning is modeled by the firing of immediate transition `Start`. This transition inserts (using its output set arc) the number of tasks (either map or reduce) that need to be executed in the corresponding phase in fluid place `Queue`. The execution of the jobs is modeled by the time-dependent fluid transition `Exec`, which is connected to place `Queue` with an output arc. Transitions `Exec` have a special time-dependent semantic that will be described in the following paragraphs: for this reason it is represented with a small clock drawn at its side. Whenever the queue is empty, the next phase can start or the job can end thanks to the firing of transition `End`.

Following [6], the fluid evolution is defined as a function $\phi : \mathbb{R}^2 \to \mathbb{R}$, which defines how the fluid level of the corresponding place changes with time. In particular, $\phi(x, t)$ represents the fluid level reached by a place at time t, if it starts at level x at time $t = 0$. This semantic is represented graphically by drawing a fluid arc that connects a fluid place (`Queue` in Figure 1) to a time-dependent fluid transition (`Exec` in Figure 1). In the model, two functions $\phi_M(x, t)$ and $\phi_R(x, t)$ are associated respectively to the map and reduce phases. These functions regulate the evolution of the fluid in the corresponding places, so that the

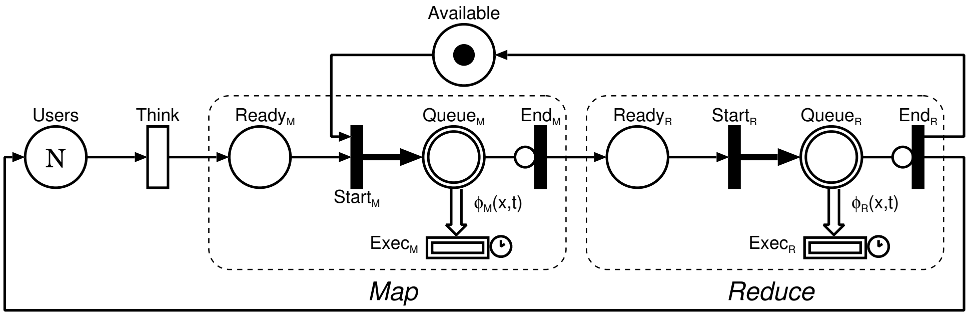

Figure 1: FSPN representation of the fluid model underlying a MapReduce job.

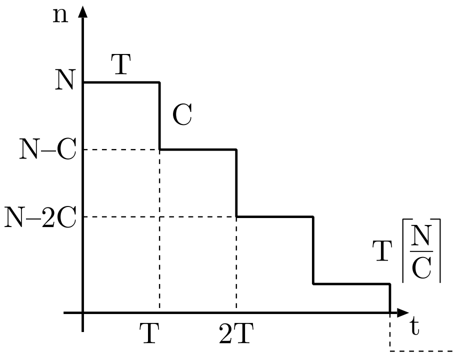

Figure 2: Fluid evolution: Deterministic task execution time

fluid level x represents the average remaining number of tasks that still need to be executed in a phase. In the following, the representation of function $\phi(x,t)$ to model the average execution time of jobs with deterministic task duration is provided. This is an initial approximation, which is acceptable within the scope of this paper.

3.2 Deterministic Task Execution Time

Task duration distributions can be generally regarded as phase-type or Erlang, parameterized according to the observed coefficient of variation (CV). The CV ranges between 0 and 1, whence the two limiting cases of deterministic and exponential distribution, respectively. In this paper we adopt the low CV approximation with deterministic task execution times, in accordance with the observed distributions.

Let us suppose that our MapReduce jobs are executed by C containers. Let us also assume that a phase is composed of N tasks, and that each task requires a deterministic time T to be executed. Figure 2 represents the evolution of the number of remaining tasks as function of time, which in our fluid model corresponds to the fluid evolution function $\phi(N,t)$. At time $t = 0$, C out of N tasks are immediately assigned to the C containers. Since their duration is deterministic, all the C tasks will end at the same time T, leaving just $N - C$ tasks left to be executed in the system. Then the next batch of C tasks will start, and it will end at $2T$, leaving in the system $N - 2C$ jobs. Function $\phi(x,t)$ can then be defined as follows:

$$\phi(x,t) = \max\left(0, x - \left\lfloor\frac{t}{T}\right\rfloor C\right) \tag{1}$$

In this scenario, the time $\tau(N,C,T)$ required to run N tasks with deterministic running time T on C containers can be computed as:

$$\tau(N,C,T) = T\left\lceil\frac{N}{C}\right\rceil \tag{2}$$

4 Experiments

In this section, we describe the results of the experiments we conducted to validate our approach. All these experiments have been performed at CINECA, the Italian supercomputing center. The experiments consist of a set of Hive queries taken from the TPC-DS benchmark suite and run on a dedicated Hadoop cluster. The rest of the section is organized as follows: Section 4.1 thoroughly describes how the experiments have been executed; Section 4.2 presents the obtained results by applying the fluid models.

4.1 Experimental Settings

The models presented in the previous sections have been validated with an experimental campaign on CINECA, the Italian supercomputing center.

PICO[1], the Big Data cluster available at CINECA, is composed of 74 nodes, each of them boasting two Intel Xeon 10-core 2670 v2@2.5GHz, with 128 GB RAM per node. Out of this 74 nodes, up to 66 are available for computation. In our experiments on PICO, we used several configurations ranging from 40 to 120 cores and set up the scheduler to provide one container per core.

The cluster is shared among different users, hence resources are managed by the Portable Batch System (PBS). PBS Professional allows for submitting jobs and checking their progress, configuring at a fine-grained level the computational requirements: for all submissions it is possible to request a number of nodes and to define how many CPUs and what amount of memory are needed on each of them. Since the cluster is shared among different users, the performance of single jobs depends on the overall system load, even though PBS tries to split the resources. Due to this, it is possible to have large variations in performance according to the total usage of the cluster. In particular, storage is not handled directly by PBS, thus leading to an even greater impact on performance.

We tried to mitigate this variability first of all by requesting entire nodes of the cluster for the execution of our experiments. In such a way, we could be sure that nobody else could run other jobs on the same nodes, thus interfering with the performance measurement.

```
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 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) R2

```
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) R3

```
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

Figure 3: Interactive queries

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

An ephemeral Hadoop cluster has been created at the beginning of each experiment on the allocated nodes. The PICO cluster provides the myHadoop tool for setting up a Hadoop 2.5.1 cluster, upon which we used Hive 1.2.1. HDFS is kept locally on the selected nodes, in order to experience lower variability than the one observed using the centralized storage.

In spite of these settings, still the experiments showed high variability, in particular with a few runs characterized by extremely high execution time. To further reduce this variability, in our analyses we discarded runs with an anomalous execution time, taking out all the experiments that lie more than three standard deviations away from the average computed for the same configuration.

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. Further, we performed experiments on five Hive queries, dubbed R1–5 and shown in Figure 3. These are *ad hoc* queries[3] that Hive runs as a single MapReduce job. The profiling phase has been conducted by extracting average task durations from at least twenty runs of each query. The numbers of map and reduce tasks varied, respectively, in the ranges $[2, 1560]$ and $[2, 1009]$. The discussed parameters are shown in Table 1. In the table we report the scale factor d, the number of tasks of each phase, respectively n^{M} and n^{R} for mappers and reducers, and the average task durations, respectively $\bar{t}^{\mathrm{M}}$ and $\bar{t}^{\mathrm{R}}$.

Table 1: Fitted parameters

Query	$\boldsymbol{d}$ **[GB]**	$\boldsymbol{n^{\mathrm{M}}}$	$\boldsymbol{\bar{t}^{\mathrm{M}}}$ **[ms]**	$\boldsymbol{n^{\mathrm{M}}}$	$\boldsymbol{\bar{t}^{\mathrm{R}}}$ **[ms]**
R1	250	144	25970	151	2346
R1	500	287	32159	300	1958
R1	750	434	34244	455	1996
R1	1000	591	40534	619	3063
R2	250	4	57326	4	8785
R2	500	2	42202	2	6582
R2	750	3	48869	3	7500
R2	1000	65	1082274	68	13086
R3	250	381	28659	400	2369
R3	500	757	37018	793	2570
R3	750	1148	42348	1009	2785
R3	1000	1560	41961	1009	3048
R4	250	288	25087	302	2958
R4	500	573	41007	601	2961
R4	750	868	43902	910	3259
R4	1000	1183	42615	1009	8667
R5	250	4	13456	4	1424
R5	500	2	11774	2	1499
R5	750	3	12682	3	1462
R5	1000	64	19557	68	1610

4.2 Experimental Results

In this section we present and discuss the results obtained with the previously described fluid techniques. First of all, we studied the empirical cumulative distribution functions (CDFs) of task durations on different nodes, in order to identify the cases in which performance was strongly affected by exogenous interference. We then considered the accuracy that

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

[3] https://github.com/deib-polimi/Hive-Experiment-Runner

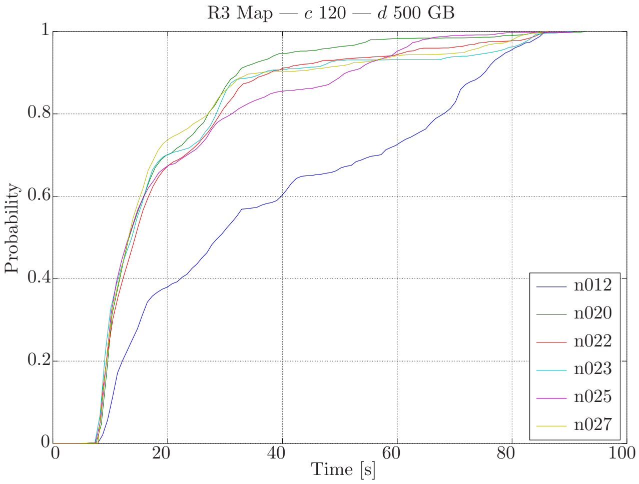

Figure 4: R3 map, 120 containers, 500 GB dataset

can be reached assuming deterministic task execution time.

4.2.1 Performance by Node

Figures 4, 5, 6 and 7 show the empirical CDFs derived from the measurements. Figures 4 and 5 refer to subsequent phases of the same experimental run, namely, query R3 running on the six-node, 120-container cluster deployment over the 500 GB dataset. Accordingly, Figures 6 and 7 show query R2 on the two-node, 40-container cluster over the same dataset.

As a general trend well represented in Figures 5 and 7, reducers tend to behave quite regularly. Most likely, possible variabilities spread across the shuffle stage that overlaps with the map phase, hence end up being hardly noticeable in each reducer task duration.

On the other hand, mappers suffer a stronger impact from external interference. In the reported graphics we have examples of both good and problematic performance. Figure 4 shows how all but one node have a very similar behavior. Further, the only different node is not significantly distant from the others, hence probably the low variability is imputable to the physiological effects of data locality. Instead, Figure 6 reports that the two involved nodes have a strongly diverse performance, with a twofold

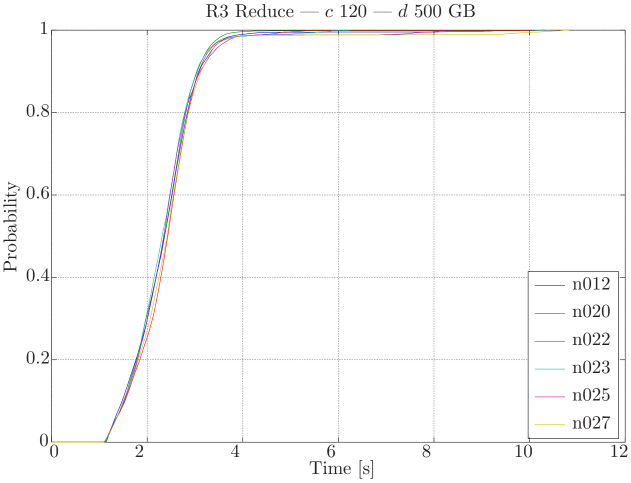

Figure 5: R3 reduce, 120 containers, 500 GB dataset

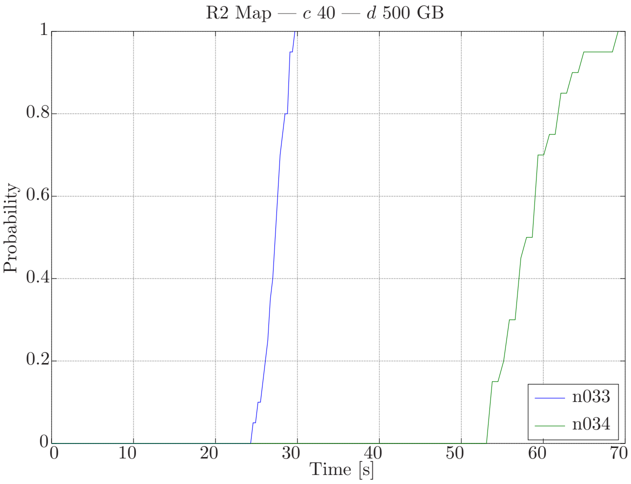

Figure 6: R2 map, 40 containers, 500 GB dataset

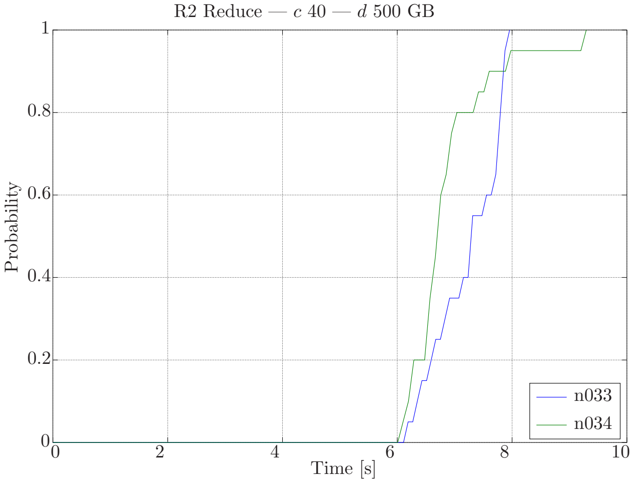

Figure 7: R2 reduce, 40 containers, 500 GB dataset

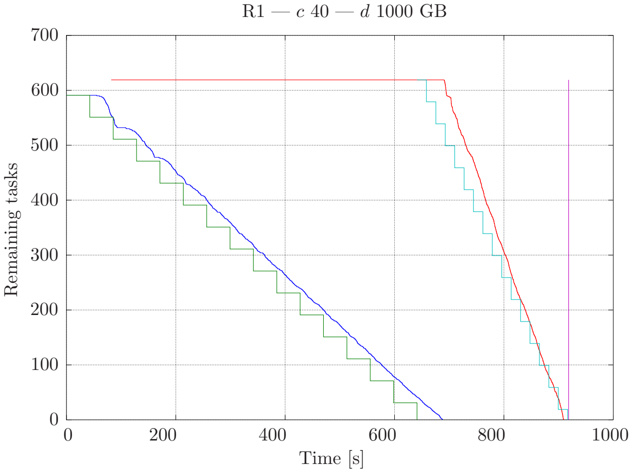

Figure 8: Query R1, 40 containers, 1 TB dataset

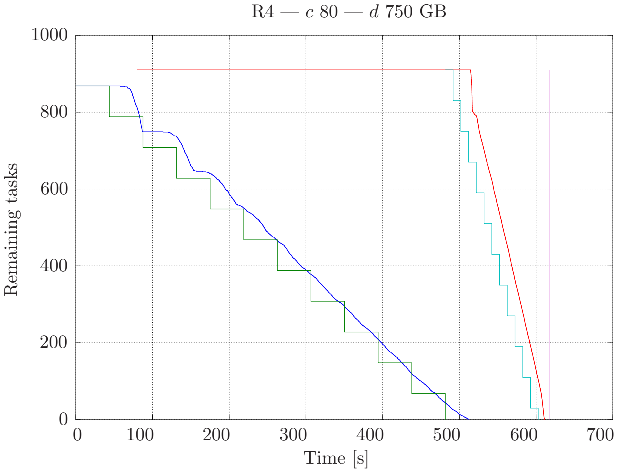

Figure 9: Query R4, 80 containers, 750 GB dataset

difference in task durations. In this case, since we are dealing with a small interactive query, any exogenous interference may cause a strong impact on the overall response time, thus making prediction harder.

4.2.2 Accuracy

Table 2 reports the results of the accuracy assessment for the proposed method. In particular, we consider the fluid evolution function discussed in Section 3.2. For several experiments we report the involved query, the total number of containers available in the cluster (c), the scale factor for the dataset generator (d), the average measured (t) and predicted (τ) execution time, and the relative error (ε). The latter is defined as:

$$\varepsilon \triangleq \frac{\tau - t}{t} \tag{3}$$

The average accuracy achieved with the approximate formula is 15.33%, perfectly in line with the expectations in the performance prediction field [15], where a 30% accuracy on response times is reasonable. The Q-Q plot in Figure 10 shows that the errors follow a Gaussian distribution, as expected. Only a handful of experiments show a relatively high error, peaking at 41.79% in absolute value. Nonetheless, the standard error of the mean is 1.75%. The largest

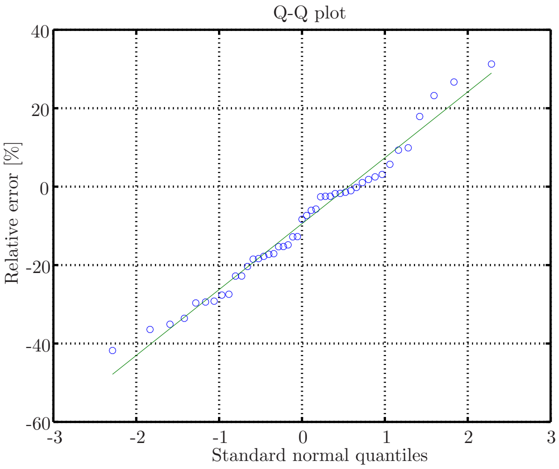

Figure 10: Q-Q plot of the errors

relative errors tend to appear either in the experiments involving R2 and R5, small queries both in terms of number of tasks and of overall duration, or in those run at 120 containers. This behavior suggests that the relative abundance of resources might amplify the effects of variability, whilst the basic assumption of the deterministic approximation for the fluid evolution is that the CV is null.

Moreover, you can see how the approximate formula (2) compares to the measured data for a couple of experiments in Figures 8 and 9. The blue lines represent the average completion times of each task in the map phase, analogously the red lines for the reduce phase. Notice that the latter begins when the first reducer container was allocated: all the overlapping time spent shuffling has not been considered to improve prediction accuracy. The stair-shaped green and cyan lines are the approximate fluid function in the limiting case of deterministic task duration, respectively for the map and reduce phase. In the end, the purple vertical line shows the average measured response time of the whole query.

5 Conclusions

In this paper we proposed fluid Petri Nets able to model the execution of MapReduce jobs running

Table 2: Accuracy with the approximate formula

Query	c	d [GB]	t [ms]	τ [ms]	ε [%]
R1	40	250	122084	106528	−12.74
R2	40	250	83364	69119	−17.09
R3	40	250	525374	530780	1.03
R1	40	500	383265	390264	1.83
R2	40	500	86864	57697	−33.58
R3	40	500	1033004	1006209	−2.59
R1	40	750	603006	592265	−1.78
R2	40	750	81292	58983	−27.44
R3	40	750	1534601	1512636	−1.43
R1	40	1000	918454	916637	−0.20
R2	40	1000	3774111	4953744	31.26
R1	60	250	80316	82314	2.49
R2	60	250	84551	68916	−18.49
R3	60	250	275684	228130	−17.25
R4	60	250	219243	201000	−8.32
R5	60	250	25924	16827	−35.09
R1	60	1000	556680	547176	−1.71
R2	60	1000	2009929	2546188	26.68
R3	60	1000	1374024	1295475	−5.72
R4	60	1000	1374244	1502290	9.32
R5	60	1000	48839	60180	23.22
R1	80	250	82531	67386	−18.35
R3	80	250	197388	138850	−29.66
R4	80	250	164811	131252	−20.36
R1	80	500	143139	103568	−27.65
R3	80	500	526760	494960	−6.04
R4	80	500	410376	423152	3.11
R1	80	750	268821	228888	−14.85
R3	80	750	791314	783088	−1.04
R4	80	750	618045	602829	−2.46
R1	80	1000	439052	406720	−7.36
R3	80	1000	1019973	994976	−2.45
R4	80	1000	960985	1015617	5.69
R1	120	250	46215	50790	9.90
R3	120	250	143650	101708	−29.20
R4	120	250	97829	82851	−15.31
R1	120	500	91809	80058	−12.80
R3	120	500	303843	214438	−29.42
R4	120	500	275407	226472	−17.77
R1	120	750	199234	115968	−41.79
R3	120	750	661214	510431	−22.80
R4	120	750	507861	430320	−15.27
R1	120	1000	349425	222165	−36.42
R3	120	1000	838708	647536	−22.79
R4	120	1000	2837286	3344833	17.89

in Hadoop 2.x clusters governed by the Capacity Scheduler. Preliminary experimental results have shown that the average percentage error that can be achieved is around 16%, making the approach suitable for design-time capacity planning or run-time cluster management. Future work will consider the extension of the approach to cope with jobs described as directed acyclic graphs, characteristic of Apache Tez or Spark. Moreover, the framework will be integrated within an optimization tool for run-time cluster management.

Acknowledgments

The results of this paper have been partially funded by EUBra-BIGSEA (grant agreement no. 690116), a Research and Innovation Action (RIA) funded by the European Commission under the Cooperation Programme, Horizon 2020 and the Ministério de Ciência, Tecnologia e Inovação (MCTI), RNP/Brazil (grant GA0000000650/04).

Eugenio Gianniti is also partially supported by the DICE H2020 research project (grant agreement no. 644869).

References

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

[2] L. Aguilera-Mendoza and M. T. Llorente-Quesada. Modeling and simulation of Hadoop Distributed File System in a cluster of workstations. In A. Cuzzocrea and S. Maabout, editors, *Model and Data Engineering*, volume 8216 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2013.

[3] D. Ardagna, C. Ghezzi, and R. Mirandola. Rethinking the use of models in software architecture. In *QoSA 2008*, pages 1–27. Springer Berlin Heidelberg, Oct. 2008.

[4] S. Balsamo, P. G. Harrison, and A. Marin. Methodological construction of product-form stochastic petri nets for performance evaluation. *Journal of Systems and Software*, 85(7):1520–1539, 2012.

[5] E. Barbierato, M. Gribaudo, and M. Iacono. Modeling Apache Hive based applications in Big Data architectures. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '13, pages 30–38. ICST, 2013.

[6] E. Barbierato, M. Gribaudo, and M. Iacono. Modeling hybrid systems in SIMTHESys. In *Proceedings of the 8th International Workshop on Practical Applications of Stochastic Modelling*. to appear, 2016.

[7] E. Barbierato, M. Gribaudo, and D. Manini. *Fluid Approximation of Pool Depletion Systems*, pages 60–75. Springer International Publishing, Cham, 2016.

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

[9] W. W. Chu, C.-M. Sit, and K. K. Leung. Task response time for real-time distributed systems with resource contentions. *IEEE Trans. Softw. Eng.*, 17(10):1076–1092, Oct. 1991.

[10] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna. Stage aware performance modeling of DAG based in memory analytic platforms. In *Cloud*, 2016.

[11] M. Gribaudo and M. Telek. *Fluid Models in Performance Analysis*, pages 271–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

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

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

[14] D. Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, 2012.

[15] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[16] D.-R. Liang and S. K. Tripathi. On performance prediction of parallel computations with precedent constraints. *IEEE Trans. Parallel Distrib. Syst.*, 11(5):491–508, May 2000.

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

[18] X. Lin, Z. Meng, C. Xu, and M. Wang. A practical performance model for Hadoop MapReduce. In *International Conference on Cluster Computing Workshops*. IEEE, 2012.

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

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

[21] R. Osman and P. G. Harrison. Approximating closed fork-join queueing networks using product-form stochastic petri-nets. *J. Syst. Softw.*, 110(C):264–278, Dec. 2015.

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

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

[24] A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic resource inference and allocation for MapReduce environments. In *ICAC '11*, June 2011.

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

[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.