# Modeling Multiclass Task-Based Applications on Heterogeneous Distributed Environments

Riccardo Pinciroli(✉), Marco Gribaudo, and Giuseppe Serazzi

Dipartimento di Elettronica, Informazione e Bioingengeria, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy
{riccardo.pinciroli,marco.gribaudo,giuseppe.serazzi}@polimi.it

**Abstract.** The volume of data, one of the five "V" characteristics of Big Data, grows at a rate that is much higher than the increase of ability of the existing systems to manage it within an acceptable time. Several technologies have been developed to approach this scalability issue. For instance, MapReduce has been introduced to cope with the problem of processing a huge amount of data, by splitting the computation into a set of tasks that are concurrently executed. The savings of even a marginal time in the processing of all the tasks of a set can bring valuable benefits to the execution of the whole application and to the management costs of the entire data center. To this end, we propose a technique to minimize the global processing time of a set of tasks, having different service requirements, concurrently executed on two or more heterogeneous systems. The validity of the proposed technique is demonstrated using a multiformalism model that consists of a combination of Queueing Networks and Petri Nets. Application of this technique to an Apache Hive case-study shows that the described allocation policy can lead to performance gains on both total execution time and energy consumption.

**Keywords:** Pool depletion systems · MapReduce · Schedulers · Energy efficiency · Performance evaluation · Queueing networks · Petri nets · Multiformalism models

## 1 Introduction

The pervasiveness of Big Data applications in organizations is occurring at a surprising high speed. Successful companies must adopt these new technologies in order to keep their advantage over competitors.

One of the most important characteristics of this new paradigm is the large size of data that must be processed in a reasonable amount of time. To address the resulting performance problem that is created, the Hadoop MapReduce technology has been proposed [9,14] originally by Google. Its operational concept is based on distributed computing and parallelism. Initially, the input data is split into blocks that are processed in parallel by a high number of tasks generated by each job in the Map. In the following Reduce phase, newly created tasks process

N. Thomas and M. Forshaw (Eds.): ASMTA 2017, LNCS 10378, pp. 166–180, 2017.
DOI: 10.1007/978-3-319-61428-1_12

in parallel the intermediate results of the Map phase producing the final output of the job. The execution of a job may require one or more cycles of Map and Reduce phases. Typically, each phase can take hours, or even days, to complete due to the significantly large data sizes and the consequent high number of tasks to be executed in parallel.

In [20] it is described an example of an Apache Hive application (see Fig. 5) regarding a query that retrieves the Facebook status according to users' gender and school. Its structure is based on three MapReduce jobs. The two Maps in *MapReduce1* generate a high number of tasks that, as a function of the query, have different resource consumption, i.e., they belong to two classes of tasks. The tasks are then assigned to Virtual Machines and executed in parallel on the same, or more likely, on different physical systems.

The analysis of resource consumption behavior during the execution of a MapReduce job shows a pattern that is repeated in each phase: generation of a large number of tasks, followed by their parallel execution. Typically, the number of parallel tasks is much higher in the Map phase with respect to the Reduce one.

The execution time of the parallel tasks is deeply influenced by two factors that are related to the characteristics of both the job and the architecture of the computing infrastructure. The first factor concerns the resource requirements of the tasks, since each phase of a job may saturate different resources, i.e., it can have a different bottleneck. The second factor regards the characteristics of the physical systems that are used. Indeed, in cloud infrastructures, the computers that are dynamically allocated to the tasks of a job may be heterogeneous and have different computational power and storage capacity (see e.g., [6]). Furthermore, these physical machines typically have a limitation on the number of tasks to be executed. This constraint, continuously reached in MapReduce jobs due to the high number of tasks, is required for performance control on response times. As a function of the mix of tasks in a single computer (referred to as subsystem), the time required by their parallel execution may be in some cases extremely inflated due to the bottlenecks that may migrate dynamically among the resources. The variability introduced in the execution time of the Map tasks may have a large impact on the execution time of a job. To cope with this problem, the tasks admission policy in each subsystem plays a fundamental role.

In order to analyze the performance of Big Data applications we study the *Pool depletion systems* [7,8]. A Pool depletion system is a framework composed by a pool of tasks and by several subsystems. We assume that the tasks are created in the pool at the same instant of time and then are sent to several parallel subsystems for their execution. The important parameter is the time required by the execution of all the tasks, referred to as depletion time. We may consider a pool as a container for the Map tasks, all of which must be completely executed before starting the following Reduce phase. In this paper, we describe a scheduling policy that minimizes the depletion time of a *pool* of tasks by appropriately allocating them on heterogeneous systems with limited capacity as a function of the different resource requirements. In particular, our technique increases the efficiency of the global system allowing the resources to operate simultaneously at their optimal conditions. Several scheduling strategies have been described in literature to deal with systems that serve a large number of tasks. The objectives of the proposed policies are: (*i*) to minimize the execution time of each task; (*ii*) to optimize the system utilization exploiting the jobs allocation on each resource, (*iii*) to minimize the execution time of a single job that is served by all the resources of the system, as may happen in scientific applications. *Case i* has been deeply analyzed in literature. FIFO, JSQ, MaxWeight and Completely Fair Scheduler strategies are some examples of scheduling policies used in that case. Unfortunately, in the problem that we consider, minimizing the task execution time does not necessarily minimize the job execution. Also *case ii* has been studied in depth. For example, Fair and Capacity scheduler algorithms [18] may be adopted by Hadoop to allocate resources in order to improve system utilization when executing jobs possibly from multiple tenants. The objective of our problem is described by *case iii*. The *Optimal population mix* [19] policy introduced in [8].

In our case, each subsystem is composed by two resources, e.g., CPU and storage. We consider a multi-class workload, with two classes of jobs. From [19] we know that, for a system with two resources and two classes of customers, a mix of classes in execution that maximizes the utilization of all the resources exists, and it is referred to as *optimal population mix*. Furthermore, we can analytically derive the *optimal population mix* for one-subsystem Pool depletion systems [8]. When the system operates with that mix, the pool can be depleted with the shortest time. Two main configurations for multi-subsystem Pool depletion systems are analyzed: the homogeneous and the heterogeneous subsystems configurations. In the former case, all the available subsystems are identical, and their corresponding resources have the same characteristics (i.e., service demands). In the latter case, each subsystem may be different from all the others, thus the resources have different service demands. In particular, the heterogeneous configuration may be used for modeling a cluster with servers having different capacity. To compute the results, we model the applications with a multiformalism model composed by queuing network and Petri nets. We simulated a variable number of subsystems and different internal population mixes, while the pool population mix is set to a constant value corresponding to the considered application.

The remainder of this paper is structured as follows. In Sect. 2 prior work on Pool depletion systems is discussed. Section 3 extends Pool depletion systems with multi-subsystem and presents the multiformalism model used for the extension. The results for homogeneous and heterogeneous multi-subsystem are shown in Sect. 4. An exploitation of this kind of framework is shown in Sect. 5 for the analysis of an Apache Hive case-study. Section 6 concludes the paper.

## 2 Background

The main applications we want to study through Pool depletion systems are the ones used in Big Data environments. Although the key feature of these applications is the parallel execution of a high number of processes, for sake of simplicity the Pool depletion systems have been analyzed so far [7,8] with only one-subsystem configuration. In [7] Pool depletion systems are described, and a Continuous time Markov chain (CTMC) is proposed to investigate multi-class cases. In particular, trade-off between energy consumption and system performance has been evaluated using some of the most common metrics, such as: the product of energy consumption and response time, or Energy-Response time product (ERP) [11,16,17]; their sum, or Energy-Response time weighted sum (ERWS) [1,2,15]; the average energy consumed per job, or Energy per job (EJ) [5,13]. Two-class Pool depletion systems are analytically investigated in [8], and closed-form equations for the derivation of the optimal population mixes of pool and subsystem are provided. For this purpose, the definitions [19] of *equi-utilization point* (i.e., the population mix of a system for which all its resources are equi-utilized) and *equi-load point* (i.e., the population mix for which all the resources of a system are equally loaded) are used. In particular, the optimal population mix of a subsystem, $\boldsymbol{\beta}^*$, corresponds to the *equi-utilization point* of the subsystem itself, and for a subsystem with two resources and two classes it is derived as follows:

$$\boldsymbol{\beta}^* = \left( \beta_A^* = \frac{\log \frac{D_{2B}}{D_{1B}}}{\log \frac{D_{1A} D_{2B}}{D_{1B} D_{2A}}},\ \beta_B^* = 1 - \beta_A^* \right) \tag{1}$$

where $D_{rc}$ is the service demand of a class $c$ job at resource $r$. The pool optimal population mix, $\boldsymbol{\alpha}^*$, corresponds to the *equi-load point* of a closed system, and may be computed as:

$$\boldsymbol{\alpha}^* = \left( \alpha_A^* = \frac{X_A(\boldsymbol{\beta}^*)}{X_A(\boldsymbol{\beta}^*) + X_B(\boldsymbol{\beta}^*)},\ \alpha_B^* = 1 - \alpha_A^* \right) \tag{2}$$

where, $X_c(\boldsymbol{\beta}^*)$ is the throughput of class $c$ jobs when the subsystem population mix is the optimal one. It has also been proved that depletion time (i.e., the time needed to completely execute all the requests initially in the pool) is minimized when the Pool depletion system works with its optimal population mixes. Note that, while subsystem population mix is a feature of the system and $\boldsymbol{\beta}^*$ may be exploited by a smart scheduler to make the whole system work with better performance, pool population mix depends on the type of application we are considering and can change only if the application changes.

In this paper we focus on multi-subsystem Pool depletion systems, where the requests in the pool are executed by several parallel subsystems. The considered subsystems are heterogeneous and have constraints on the number of tasks executed simultaneously. A similar approach was adopted in [5]. In that case, differently from this paper, a multi-class *open network* was taken into consideration. A smart scheduler was implemented to decrease the energy consumption of the global system. In particular, it forwarded each incoming job to one of its subsystems, depending on their current population mix.

## 3 The Model

The considered Pool depletion system is described using the model consisting of a Coloured Petri Net (CPN) and a multi-class fork-and-join queuing network shown in Fig. 1. The workload of the model consists of two type of customers: the tokens, representing the colours of the Petri nets, and the jobs, representing the requests to be executed by the queueing networks. Each type of workload comprises two different classes of customers. CPN tokens are used to control the access to resources by the tasks, while queuing networks primitives are used to describe the service demands of the tasks. According to [12], the use of several formalisms allows exploiting the most appropriate modelling primitives to express the corresponding concepts in the most efficient and natural way.

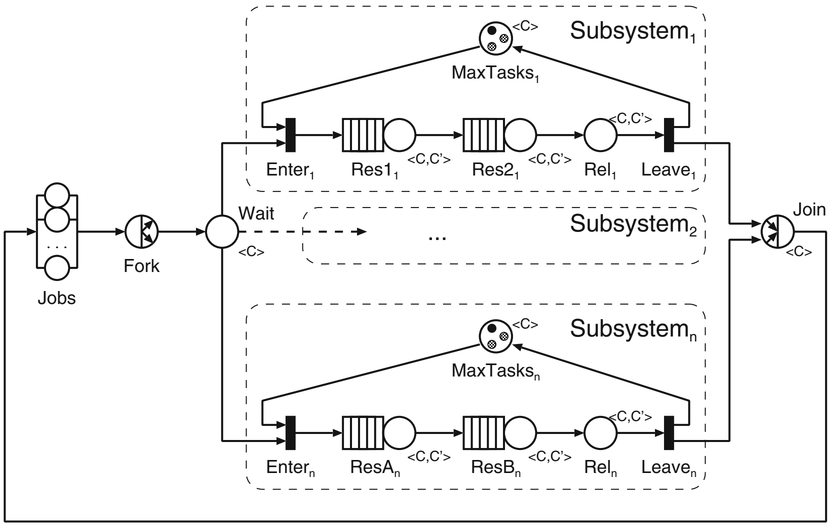

**Fig. 1.** The multiformailsm model of the considered scenario

The jobs are created in the delay station `Jobs`: in this work we will focus, without loss of generality, on a single job that is continuously executed, and we set the "think time" (service time of the `Jobs` station) to $Z = 0$. The job enters the `Fork` node and it is splitted into $N_A$ tasks of class $A$ and $N_B$ tasks of class $B$, with $N = N_A + N_B$. Tasks waiting to be executed are represented as tokens into place `Wait`: in this case, colour class $<C>$ is used to remember the task class as an attribute of the token. Table 1 summarises the colour classes used in the model; in Fig. 1 colour classes are represented, in angled brackets, as labels associated to places and, with a slight abuse of notation, to queuing stations.

The place `MaxTasks` represents the considered scheduling assignment and contains tokens belonging to the $<C>$ colour class, initialised to $K_A$ and $K_B$ tokens for the corresponding task classes. Its marking represents the chosen

**Table 1.** Colour-sets.

| Colour-set | Description |
|---|---|
| $<C>$ | Task class $C = \{A, B\}$. |
| $<C, C'>$ | Task class $C$, original class $C'/$ |

configuration that allows a total of $K = K_A + K_B$ tasks simultaneously in a subsystem. The execution of a task starts with a firing of transition `Enter`, which can occur in one of the four following modes. When the subsystem is in normal operation, transition can fire in mode 1 or 2 removing respectively one token of class $A$ or $B$ from its input place, and creating customers of class $(A, A)$ or $(B, B)$ in the queue `Res1`, that represents the first resource of the subsystem. When there are no more tokens of either class $B$ or $A$ in place `Wait`, transition `Enter` fires in mode 3 or 4 allowing a task of class $A$ or $B$ to enter instead of the one that has already been completed. Queuing stations `Res1` and `Res2` represent the two resources of the subsystem. Even if the task classes generated by the CPN transition are $(A, A)$, $(A, B)$, $(B, B)$ and $(B, A)$, only the first component of each couple is used to determine the service requirements of one task. The second component is instead used in place `Rel` to allow transition `Leave` forwarding the correct task type to the `Join` node, and to return the acquired task token into place `MaxTasks`. This is accomplished by the firing of transition `Leave` according to four modes, as summarised in Table 2.

**Table 2.** Transitions firing modes.

| Transition | Mode | $In_1$ | $In_2$ | $Out_1$ | $Out_2$ | Description |
|---|---|---|---|---|---|---|
| `Enter` | | `Wait` | `MaxTasks` | `Res1` | | |
| | 1 | $A$ | $A$ | $(A, A)$ | | Class A task |
| | 2 | $B$ | $B$ | $(B, B)$ | | Class B task |
| | 3 | $A$ | $B$ with `Wait`.$B = 0$ | $(A, B)$ | | Class A task, depletion |
| | 4 | $B$ | $A$ with `Wait`.$A = 0$ | $(B, A)$ | | Class B task, depletion |
| `Leave` | | `Rel` | | `MaxTasks` | `Join` | |
| | 1 | $(A, A)$ | | $A$ | $A$ | Class A task |
| | 2 | $(B, B)$ | | $B$ | $B$ | Class B task |
| | 3 | $(A, B)$ | | $B$ | $A$ | Class A task, depletion |
| | 4 | $(B, A)$ | | $A$ | $B$ | Class B task, depletion |

Each subsystem has a similar structure, and the same sub-model is repeated $n$ times as shown in Fig. 1. Each subsystem can be characterised by different service demand for its resources, and different tasks allowances $K_A$ and $K_B$. When all tasks have been completed, the `Join` primitive can fire, returning the customer to the reference station, and allowing the next job to start. To characterise the configuration of the system, we will denote with:

$$\boldsymbol{\alpha} = \left(\alpha_A = \frac{N_A}{N_A + N_B},\ \alpha_B = 1 - \alpha_A\right)$$
$$\boldsymbol{\beta} = \left(\beta_A = \frac{K_A}{K_A + K_B},\ \beta_B = 1 - \beta_A\right)$$

the *pool population mix* and the *subsystem population mix*, respectively, as described in Eqs. (1) and (2).

## 4 Results

In this section, the results obtained analyzing multi-class applications on distributed environments are shown. In [8], we showed that analytical equations – to identify the optimal point for a single-machine system – exist. Eqs. (1) and (2) show the results for subsystem and pool of a Pool depletion system, respectively. While considering distributed environments does not affect Eq. (1) since it refers to each single subsystem, it influences the optimal population mix of the pool in Eq. (2). In fact, as said in Sect. 2, optimal population mix of the pool is related to the equi-load point of the system and it varies when we take into account several different subsystems. In particular, we expand $\alpha_A^*$ as follows:

$$\alpha_A^* = \frac{X_A(\boldsymbol{\beta}^*)}{X_A(\boldsymbol{\beta}^*) + X_B(\boldsymbol{\beta}^*)} = \frac{\sum_{s=1}^{n} X_A^s(\boldsymbol{\beta}^{s*})}{\sum_{s=1}^{n} X_A^s(\boldsymbol{\beta}^{s*}) + \sum_{s=1}^{n} X_B^s(\boldsymbol{\beta}^{s*})} \tag{3}$$

where $n$ is the total number of subsystems, and $X_c^s(\boldsymbol{\beta}^{s*})$ is the throughput for class $c$ jobs at subsystem $s$, when the population mix of that subsystem is its optimal population mix, $\boldsymbol{\beta}^{s*}$.

Two main system configurations are considered. First of all, the effects of homogeneous distributed systems – where all the subsystems are identical – are taken into account. Then, we consider heterogeneous distributed environments, assuming that each subsystem may have different characteristics with respect to the other ones. The analyses are performed using JSIMgraph, the JMT [4] simulation tool, and simulating the model described in Sect. 3. All the metrics are computed with 99% confidence interval and a 3% maximum relative error.

### 4.1 Homogeneous

The main advantage introduced by homogeneous distributed systems is the parallelization of jobs execution. Since all the subsystems are the same, their optimal population mix, $\boldsymbol{\beta}^{s*}$, are identical. Due to that, the throughput of each subsystem does not change and Eq. (3), after some algebraic manipulation, becomes equal to Eq. (2).

In Fig. 2 three performance metrics for a homogeneous distributed system are depicted against the number $n$ of subsystems, and their population mix, $\boldsymbol{\beta}$. The pool size of the considered system is $N = 1008$ and the capacity of each subsystem is $K = 108/n$. In the simulations we consider $n = \{1, 2, 3, 4, 6\}$

and the following service demand matrix for each two-resource and two-class subsystem:

$$D_{rc} = \begin{bmatrix} 0.75 & 0.64 \\ 0.48 & 1.25 \end{bmatrix} \tag{4}$$

The optimal population mix of these subsystems is computed through Eq. (1) and it is $\boldsymbol{\beta}^* = (0.6,\ 0.4)$. Their *equi-load point* is $\boldsymbol{\alpha}^* = (0.693,\ 0.307)$. Since all the subsystems are identical, we assume they are all set to work with the same population mix. For sake of simplicity, we only consider an application with $N_A = 724, N_B = 284$ and $\boldsymbol{\alpha} = \left(\frac{724}{1008},\ \frac{284}{1008}\right)$.

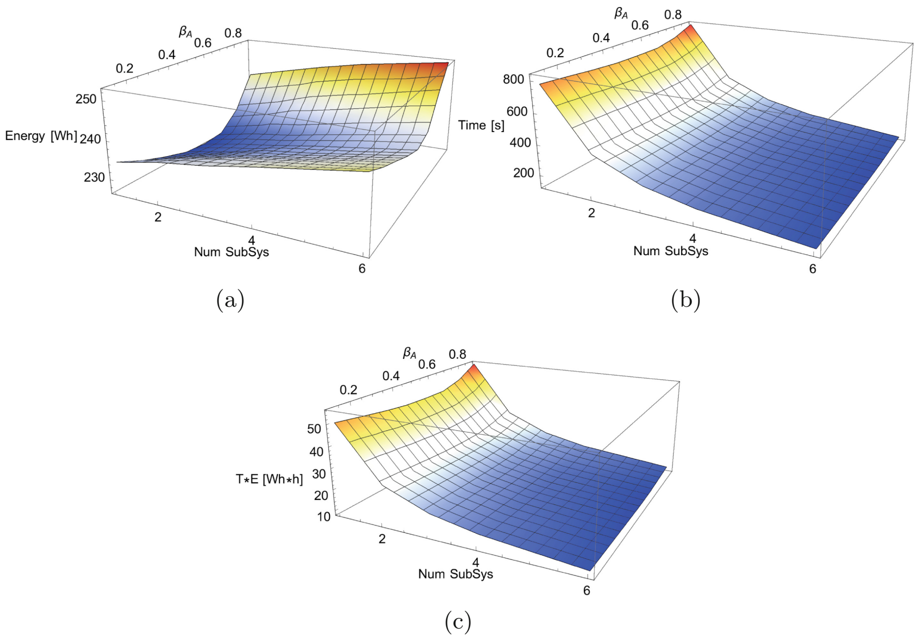

**Fig. 2.** Energy consumption (a), depletion time (b) and Time · Energy (c) of a homogeneous distributed system for $\boldsymbol{\alpha} = \left(\frac{724}{1008},\ \frac{284}{1008}\right)$, as a function of total number of subsystems and their population mix.

Figure 2a represents the energy consumed to serve all the tasks initially in the pool when $P^1_{idle} = 315\,\mathrm{W}, P^1_{busy} = 630\,\mathrm{W}, P^2_{idle} = 250\,\mathrm{W}$ and $P^2_{busy} = 500\,\mathrm{W}$. $P^r_{idle}$ and $P^r_{busy}$ are the power consumption of a resource $r$ when it is idle and fully utilized, respectively. The energy consumed by a Pool depletion system is defined as the product between depletion time and power, $E = T \cdot P$, and the power consumption is computed with equation

$$P(U) = P_{idle} + (P_{busy} - P_{idle}) \cdot U \tag{5}$$

proposed by Fan et al. in [10]. In our case, the minimum value of energy consumption is observed when the system is working with only one subsystem. However, it is interesting to note that working with six subsystems and with the optimal population mix, $\boldsymbol{\beta}^*$, lets the service provider save more energy than working with only one subsystem and with a suboptimal population mix. Figure 2b shows the depletion time of the system (i.e., the time needed by the system to complete all the tasks initially in the pool). In this case, providing a large number of subsystems allows the service provider to parallelize the work, and with six subsystems the depletion time is four times lower than with only one subsystem. In order to take into consideration both the measures (i.e., energy consumption and depletion time), we compute their product and the results are shown in Fig. 2c. Since the saved amount of time working with six subsystems is definitely greater than the energy the system consumes working with only one machine, the configuration with the larger amount of subsystems is the one with the best global performance, assuming that energy is as important as the depletion time for service provider. Furthermore, for the considered application, i.e., $\boldsymbol{\alpha} = \left(\frac{724}{1008},\ \frac{284}{1008}\right)$, all the three analyzed metrics have their minimum point when all the available subsystems work with their optimal population mix, $\boldsymbol{\beta}^*$.

## 4.2 Heterogeneous

The analysis of heterogeneous distributed environements is interesting since it lets us consider more general systems. For example, they can be used to model a datacenter with different types of servers (e.g., new vs. old, fast vs. slow, etc.). In order to study this kind of environments, we focus on the two-subsystem case. Since they must have different features, the first subsystem is defined through service demand matrix in Eq. (4), whereas the following matrix is considered for the second subsystem:

$$D_{rc} = \begin{bmatrix} 0.86 & 0.65 \\ 0.3 & 1.02 \end{bmatrix} \tag{6}$$

The optimal population mix of the subsystem defined in Eq. (6) is $\boldsymbol{\beta}^{2*} = (0.3,\ 0.7)$ and its *equi-load point* is $\boldsymbol{\alpha}^{2*} = (0.409,\ 0.693)$. They have been derived through Eqs. (1) and (2), respectively. As previously said, the optimal population mix of the pool must be computed with Eq. (3) if the system has heterogeneous subsystems, and in this case it is $\boldsymbol{\alpha}^* \simeq (0.55,\ 0.45)$. This result highlights another important gain of a heterogeneous distributed environment, besides the parallelization of the workload: indeed, differently from the homogeneous case, the implementation of inhomogeneous subsystems lets the service provider execute different applications with better performance. This is shown in Fig. 3 that depicts the depletion times of two different applications (i.e., pool population mixes) served by a homogeneous system and a heterogeneous one. The population mixes of the two subsystems is still assumed to be the same for both of them. As said, it is interesting to note that the heterogeneous configuration provides a shorter depletion time than the homogeneous one when it is serving application $\boldsymbol{\alpha} = (0.55,\ 0.45) \simeq \boldsymbol{\alpha}^*_{heter}$. On the contrary, if application $\boldsymbol{\alpha} = (0.70,\ 0.30)$ is considered, the homogeneous environment performs better than the heterogeneous one, since the new application is closer to the homogeneous optimal pool population mix, $\boldsymbol{\alpha}^*_{homo}$.

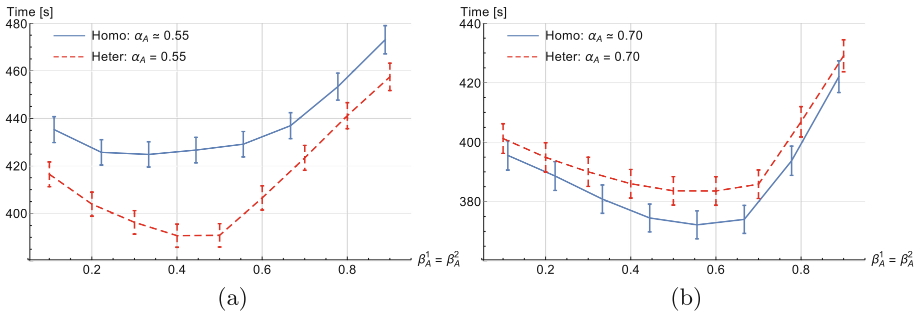

**Fig. 3.** Comparison of depletion time of homogenenous and heterogeneous systems for two different applications: $\boldsymbol{\alpha} = (0.55,\ 0.45)$ (a) and $\boldsymbol{\alpha} = (0.70,\ 0.30)$ (b).

Energy consumption, depletion time and their product are depicted in Fig. 4. Each metric is analyzed against the population mix of each subsystem (i.e., $\boldsymbol{\beta}^1$ and $\boldsymbol{\beta}^2$). For all the three metrics, the minimum values are registered when each subsystem $s$ works with its optimal population mix, $\boldsymbol{\beta}^{s*}$. Also in this case, the amount of time saved to complete all the tasks initially in the pool is definitely larger than the energy saved, thus depletion time affects the system performance more than energy consumption when both the metrics have the same weight on the service provider's decisions.

## 5 Exploitation: Apache Hive

In this section we apply the results obtained in Sect. 4 to a real Big Data application. Simulations have been performed for both FIFO and *Optimal population mix* admission policies. Although Pool depletion systems can model many different Big Data applications, the case-study we consider in this paper is an Apache Hive [20] based application. Apache Hive is an open source data warehouse system and is used on top of Apache Hadoop. It has been extensively adopted by many organizations (e.g., Facebook, Netflix, Spotify) since it makes easier the management of large data and their queries. Hive was introduced by Facebook in [20]. It can be run on different Hadoop's framework (e.g., MapReduce, YARN, Spark, Tez) and provides a SQL-like query language that is called HiveQL. A query to retrieve most popular Facebook status based on users' gender and school is used as an example in [20]. Its simplified query plan is shown in Fig. 5, where three MapReduce jobs are represented.

In order to study the performance of the *Optimal population mix* strategy shown in Sect. 4, we compare the time this strategy needs to complete a MapReduce job with the time required by the FIFO strategy for the same kind of job. We focus on the join function of a Hive query, that is represented by *MapReduce1* in Fig. 5. In that case, both Map and Reduce have multi-class workloads. Indeed, Map phase gets its data from two different tables, whereas the Reduce one returns two temporary tables. Assuming that two tables must be joint, we

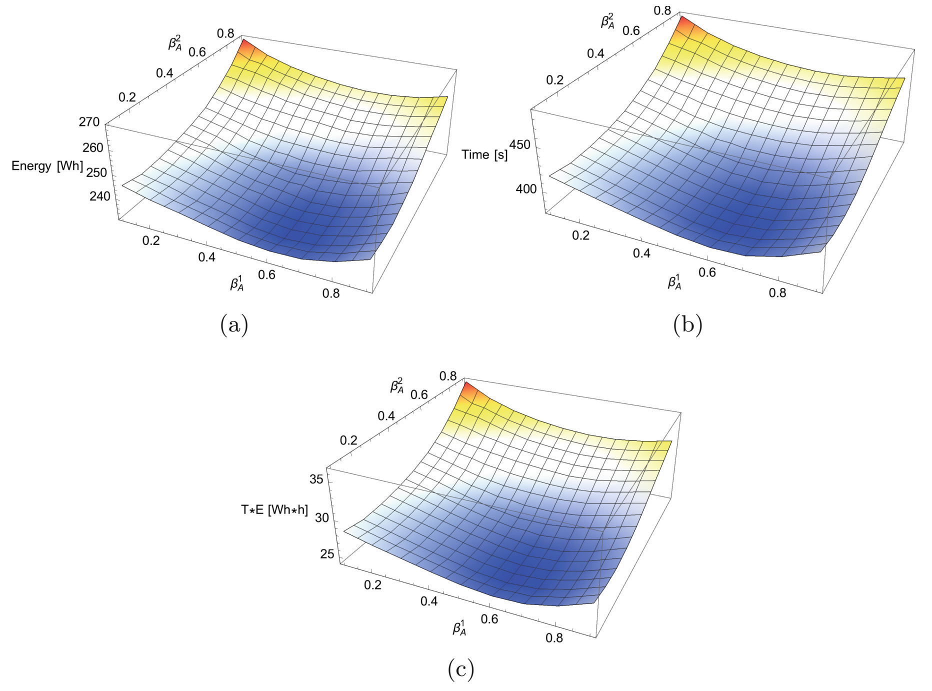

**Fig. 4.** Energy consumption (a), depletion time (b) and Time · Energy (c) of a heterogeneous distributed system for $\boldsymbol{\alpha} = (0.55,\ 0.45)$, as a function of population mix of each subsystem.

want to identify the best way to execute all the tasks of a MapReduce job in order to decrease the total time needed to complete the join clause and get the expected results. The Map and Reduce phases, are modeled by two different heterogeneous two-subsystem environments, that have been characterized starting from [21] and [3]. The former provides some data about MapReduce jobs executed on Facebook's clusters, whereas the latter refers to a public Hadoop repository[1]. Thus, the Map system is defined by the following service demands matrices (in seconds):

$$D_{rc}^{1,Map} = \begin{bmatrix} 25 & 21 \\ 16 & 39 \end{bmatrix} \qquad D_{rc}^{2,Map} = \begin{bmatrix} 55 & 39 \\ 28 & 82 \end{bmatrix} \tag{7}$$

whereas the Reduce one has the following service demands (still in seconds):

$$D_{rc}^{1,Reduce} = \begin{bmatrix} 60 & 46 \\ 17 & 70 \end{bmatrix} \qquad D_{rc}^{1,Reduce} = \begin{bmatrix} 29 & 22 \\ 10 & 34 \end{bmatrix} \tag{8}$$

We assume the MapReduce job is splitted into $N_{Map} = 1000$ and $N_{Red} = 200$ tasks before Map and Reduce phases, respectively. The number of tasks that is

[1] Available at http://ftp.pdl.cmu.edu/pub/datasets/hla/. Please, include *http* at the beginning of the URL to make it work.

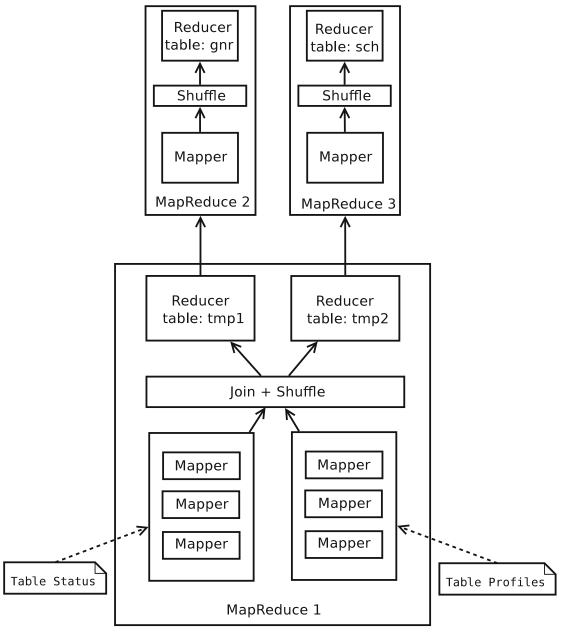

**Fig. 5.** Query plan of three MapReduce jobs analyzed in [20].

concurrently executed during the two phases is $K_{Map} = 100$ for Map and $K_{Red} = 50$ for Reduce. Using Eq. (1) on all the service demand matrices, we derive the optimal population mixes of each subsystem, i.e., $\boldsymbol{\beta}^{1*}_{Map} = (0.58, 0.42)\,, \boldsymbol{\beta}^{2*}_{Map} = (0.29, 0.71)\,, \boldsymbol{\beta}^{1*}_{Red} = (0.52, 0.48)$ and $\boldsymbol{\beta}^{2*}_{Red} = (0.26, 0.74)$. Finally, we take into account an application whose $\boldsymbol{\alpha}_{Map} = (0.53, 0.47)$ for Map phase and $\boldsymbol{\alpha}_{Red} = (0.49, 0.51)$ for the Reduce one. For both Map and Reduce, $\boldsymbol{\alpha} \simeq \boldsymbol{\alpha}^*$ has been computed through Eq. (3).

The results of the simulations have been computed with 99% confidence interval, and they are shown in Fig. 6. Besides the *Optimal population mix* strategy, we analyze two FIFO policies: we consider completion of all *class A* tasks before starting executing *class B* ones, and viceversa.

In Fig. 6 the time to complete all the Map and Reduce tasks into the system are compared based on the scheduling policy adopted by the scheduler. The total time to complete a MapReduce job is also depicted. Due to the large number of tasks we assume to be into the system during the Map, this phase is affecting the global performance of the system more than the Reduce one. When the subsystems work with their optimal population mixes, the service provider complete all the tasks (i.e., the MapReduce job) in a shorter time than using a FIFO strategy. The system requires the longest time to complete all the tasks when it serves *class B* tasks after completing all the *class A* ones. For sake of simplicity, we assume Reduce tasks are served after the Map ones, and no parallelism between tasks belonging to different phases is admitted (i.e., *Total length* = *Map phase length* + *Reduce phase length*). Based on the observed results,

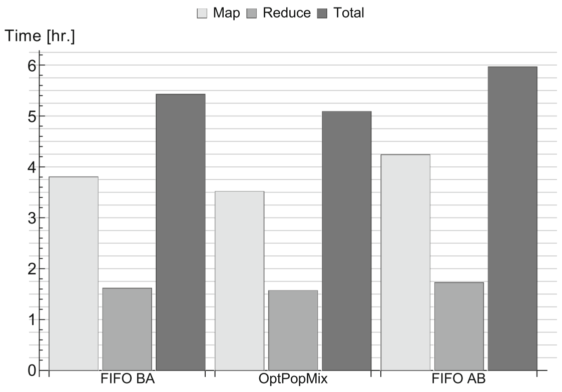

**Fig. 6.** Time to complete the MapReduce jobs adopting different scheduling strategies.

*Optimal population mix* strategy lets the system save up to 15% on the total time, with respect to the FIFO strategies. This result is even more interesting when several MapReduce jobs are concurrently executed and a larger amount of time can be saved just adopting a different scheduling policy.

## 6 Conclusion

In this paper we investigated the performance gain introduced by the adoption of an *Optimal population mix* scheduling strategy in the execution of Big Data applications. In order to do that, Pool depletion systems have been extended with the implementation of two or more subsystems. This extension of Pool depletion systems was necessary since parallel tasks execution is the key feature of the Big Data applications. We identified two main system configurations: the homogeneous and the heterogeneous ones. In the former, all the subsystems are identical and the only advantage with respect to the single-subsystem Pool depletion systems is the possibility to parallelize the execution of the tasks. In the latter case, instead, the subsystems may have different characteristics (i.e., service demands). The service provider may exploit this other feature in order to process some applications (defined by the number of the *Class A* and *Class B* tasks from which they are composed) with better performance.

The analysis of multi-subsystems Pool depletion systems provided in this paper highlights that depletion time (i.e., the time needed to complete all the tasks initially in the pool) affects the global performance of these systems more than the energy consumed to run a larger number of subsystems.

Finally, the *Optimal population mix* scheduling strategy is compared with the default FIFO policy. The comparison is performed considering a specific MapReduce job: Facebook's Apache Hive query. From the simulation of the case-study and the anlysis of its results, the implementation of the *Optimal population mix* policy lets the service provider save up to the 15% on the total amount of time needed to complete the MapReduce job.

**Acknowledgement.** This research was supported in part by the European Commission under the grant ANTAREX H2020 FET-HPC-671623.

## References

1. Andrew, L.L., Lin, M., Wierman, A.: Optimality, fairness, and robustness in speed scaling designs. In: ACM SIGMETRICS Performance Evaluation Review, vol. 38, pp. 37–48. ACM (2010)
2. Bansal, N., Chan, H.L., Pruhs, K.: Speed scaling with an arbitrary power function. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 693–701. Society for Industrial and Applied Mathematics (2009)
3. Barbierato, E., Gribaudo, M., Manini, D.: Fluid approximation of pool depletion systems. In: Wittevrongel, S., Phung-Duc, T. (eds.) ASMTA 2016. LNCS, vol. 9845, pp. 60–75. Springer, Cham (2016). doi:10.1007/978-3-319-43904-4_5
4. Bertoli, M., Casale, G., Serazzi, G.: JMT: performance engineering tools for system modeling. SIGMETRICS Perform. Eval. Rev. **36**(4), 10–15 (2009)
5. Cerotti, D., Gribaudo, M., Piazzolla, P., Pinciroli, R., Serazzi, G.: Multi-class queuing networks models for energy optimization. In: Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools, pp. 98–105. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2014)
6. Cerotti, D., Gribaudo, M., Piazzolla, P., Serazzi, G.: Flexible CPU provisioning in clouds: a new source of performance unpredictability. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, 17–20 September 2012, pp. 230–237 (2012)
7. Cerotti, D., Gribaudo, M., Pinciroli, R., Serazzi, G.: Stochastic analysis of energy consumption in pool depletion systems. In: Remke, A., Haverkort, B.R. (eds.) MMB&DFT 2016. LNCS, vol. 9629, pp. 25–39. Springer, Cham (2016). doi:10.1007/978-3-319-31559-1_4
8. Cerotti, D., Gribaudo, M., Pinciroli, R., Serazzi, G.: Optimal population mix in pool depletion systems with two-class workload. In: 10th EAI International Conference on Performance Evaluation Methodologies and Tools. ACM (2017)
9. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
10. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: ACM SIGARCH Computer Architecture News, vol. 35, pp. 13–23. ACM (2007)
11. Gandhi, A., Gupta, V., Harchol-Balter, M., Kozuch, M.A.: Optimality analysis of energy-performance trade-off for server farm management. Perform. Eval. **67**(11), 1155–1171 (2010)
12. Gribaudo, M., Iacono, M.: Theory and Application of Multi-formalism Modeling. IGI Global, Hershey (2013)
13. Ho, T.T.N., Gribaudo, M., Pernici, B.: Characterizing energy per job in cloud applications. Electronics **5**(4), 90 (2016)
14. Huang, L., Wang, X.W., Zhai, Y.D., Yang, B.: Extraction of user profile based on the hadoop framework. In: 5th International Conference on Wireless Communications, Networking and Mobile Computing, WiCom 2009, pp. 1–6. IEEE (2009)
15. Hyytiä, E., Righter, R., Aalto, S.: Task assignment in a heterogeneous server farm with switching delays and general energy-aware cost structure. Perform. Eval. **75**, 17–35 (2014)

16. Kang, C.W., Abbaspour, S., Pedram, M.: Buffer sizing for minimum energy-delay product by using an approximating polynomial. In: Proceedings of the 13th ACM Great Lakes Symposium on VLSI, pp. 112–115. ACM (2003)
17. Kaxiras, S., Martonosi, M.: Computer architecture techniques for power-efficiency. Synth. Lect. Comput. Archit. **3**(1), 1–207 (2008)
18. Kulkarni, A.P., Khandewal, M.: Survey on hadoop and introduction to YARN. Int. J. Emerg. Technol. Adv. Eng. **4**(5), 82–87 (2014)
19. Rosti, E., Schiavoni, F., Serazzi, G.: Queueing network models with two classes of customers. In: Proceedings Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 1997, pp. 229–234. IEEE (1997)
20. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow. **2**(2), 1626–1629 (2009)
21. Zaharia, M., Borthakur, D., Sarma, J.S., Elmeleegy, K., Shenker, S., Stoica, I.: Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley (2009)