Modeling Techniques for Pool Depletion Systems

D. Cerotti, M. Gribaudo, R. Pinciroli, and G. Serazzi

Abstract The evolution of digital technologies and software applications have introduced a new computational paradigm that involves the concurrent processing of jobs taken from a large pool in systems with limited computational capacity. *Pool Depletion Systems* is a framework proposed to analyze this paradigm where an optimal admission policy for jobs allocation is adopted to improve the performance of the system. Markov analysis and discrete event simulation, two techniques adopted to study *Pool Depletion Systems* framework, may require a long time before providing results, especially when dealing with complex systems. For this reason, a fluid approximation technique is presented in this chapter; in fact, it can provide results in a very short time, slightly decreasing their accuracy.

1 Introduction

In the last years, the development of new programming paradigms was increasingly affected by the evolution of Big Data technologies. Indeed, the enormous amount of data generated by IoT and any other type of input devices requires specific techniques for storage, access and process. To satisfy the increasing demand of computational capabilities required to process such huge volume of data in a reasonable amount of time, the concepts of parallelism and distributed computation have been extensively

D. Cerotti
DiSIT, Università Piemonte Orientale, Alessandria, Italy, e-mail: davide.cerotti@uniupo.it

M. Gribaudo
DEIB, Politecnico di Milano, Milano, Italy, e-mail: marco.gribaudo@polimi.it

R. Pinciroli
DEIB, Politecnico di Milano, Milano, Italy, e-mail: riccardo.pinciroli@polimi.it

G.. Serazzi
DEIB, Politecnico di Milano, Milano, Italy, e-mail: giuseppe.serazzi@polimi.it

adopted. Big Data applications split the data to be analyzed into blocks and generate multiple tasks dedicated to their processing. The tasks are executed in parallel by the resources of a distributed computing environment. At the end of all the executions, newly created tasks concurrently process the intermediate results to produce the final output of the application. This operational structure, proposed originally by Google with Hadoop MapReduce [5, 8], is adopted by many current Big Data applications. A complete execution of a Big Data application consists of a sequence of parallel computations and their following synchronization. Typically, the computing framework orchestrates the parallel executions of the tasks taking into consideration the configuration of the available architecture. The allocation strategy of the tasks to the distributed systems has a deep influence on the execution time of the applications. Indeed, the load of the systems must be controlled in order to avoid the creation of bottlenecks that severely limit the performance. On the other hand, also the under-load of the resources must be avoided trying to balance their utilization. A policy to allocate the tasks to the distributed systems to minimize the execution time of the entire application was presented in [3].
To this end, we studied *Pool Depletion Systems* (PDS) and in this chapter we describe the techniques to be used for their modeling.

PDS is a framework adopted in [2, 3, 11] to analyze all the systems where a huge amount of tasks, composing a job, must be executed by one or more subsystems with a finite capacity. PDS are studied assuming the tasks may belong to two different classes that are served by subsystem's resources with different service demands. Each PDS is characterized by a *transient behavior* that may be summarized as follows: *i)* initially, all the tasks of a job are in the pool and they are admitted gradually into a finite capacity subsystem to be executed; *ii)* whenever a task is completed, it leaves the subsystem and another task can start being processed; *iii)* when all the tasks have been executed, the job is completed and a new request can be served. The metric we are interested in is the *depletion time*, i.e., the time required to execute all the tasks. The main goal is to find the allocation policy of all the tasks in the pool that minimizes the depletion time.

Two techniques have been used to model PDS: *i)* Markov analysis [2], that provides exact analytic solutions, and *ii)* discrete events simulation [3, 11]. The former may be affected by the state space explosion when the number of tasks to be considered is large. The latter could take a very long time to complete a simulation run with complex models. In this chapter, we propose a further approach, i.e., the fluid approximation technique, to analyze PDS. It allows the analysis of models with large dimensions (e.g., even million of tasks) in a very short amount of time. Comparisons of time required by the three techniques to solve models of various sizes and the accuracy of their results are provided and analyzed.

This chapter is organized as follows. Sections 2 and 3 provide a review of the background results and a description of the system architecture, software and hardware, considered. Section 4 describes and compares different models to study PDS. Section 5 draws conclusions.

2 Related work

In this Section we emphasize some results on product-form closed queuing networks [1] with two-class workload and fixed rate single server stations that will be exploited in the analysis of depletion systems developed in the following Sections. Two-class workloads are simple enough to analytically deal with, while being representative of several realistic systems' workload. In this case, the workload of models with M stations can be characterized by a matrix of service demands $\mathbf{D} = [D_{rc}]$ where D_{rc} is the time required by station $r \in \{1, \ldots, M\}$ to completely process a class $c \in \{A, B\}$ request. Hereinafter, we will consider a system with two resources and the following service demands:

$$\mathbf{D} = \begin{bmatrix} 0.8 & 0.496 \\ 0.2 & 1.25 \end{bmatrix} \tag{1}$$

According to matrix (1), class A service demands on stations 1 and 2 are 0.8 and 0.2 time units, respectively, whereas class B demands on the two stations are 0.496 and 1.25 time units, respectively. Note that, since the system considered in this chapter is a separable queuing network, its solution depends only on the product of visits v_r and service time S_r at each resource r (i.e., on the service demand $D_r = v_r \cdot S_r$) and not on the individual factors [9]. For this reason, the topology of the network is arbitrary, thus we assume the two resources to be in series (as shown in Fig. 1).

The performance of systems with multi-class workloads depends on the fraction of requests in execution for each class, referred to as *population mix*. Let $\boldsymbol{\beta} = (\beta_A, \beta_B)$ represents the population mix. In the example of matrix $\mathbf{D}$, when $\beta_A = 0$ we have only class B requests and the bottleneck (i.e., the saturated resource) is station 2, whereas with $\beta_A = 1$ (only class A requests) the bottleneck is station 1. Thus, varying the value of $\boldsymbol{\beta}$ a *bottleneck switch* occurs. In these conditions, there exists a value $\boldsymbol{\beta}^*$ such that both stations are equally utilized for any number of requests inside the system; such value is called the *equi-utilization point*. In addition, providing that the service demand matrix values allow the occurrence of a bottleneck switch[1] and when the number of requests is sufficiently large, it is possible to identify an interval of values of the population mix that concurrently saturate the stations. Indeed, for values of $\boldsymbol{\beta}$ belonging to such *common saturation sector* (CSS) the system throughput is maximum and the system response time is minimum.

In [12] it is shown that the equi-utilization point is computed as:

$$\boldsymbol{\beta}^* = \left(\beta^*_A = \frac{log \frac{D_{22}}{D_{12}}}{log \frac{D_{11} D_{22}}{D_{12} D_{21}}}, \beta^*_B = 1 - \beta^*_A \right) \tag{2}$$

and it is proved to always belong to the CSS of edges:

[1] It can be verified by checking if $D_{1A} > D_{2A}$ and $D_{1B} < D_{2B}$.

$$\begin{cases} \beta_A^L = D_{2A} \frac{D_{2B} - D_{1B}}{D_{1A} D_{2B} - D_{2A} D_{1B}} \\ \beta_A^U = D_{1A} \frac{D_{2B} - D_{1B}}{D_{1A} D_{2B} - D_{2A} D_{1B}} \end{cases} \tag{3}$$

where β_A^L and β_A^U denote the lower and upper edges, respectively. Moreover, in the same paper the behavior of the per-class throughputs are analytically computed. In particular, we have that for values of β_A inside the CSS the per-class throughputs are:

$$\begin{array}{lll} X_A^{CSS} = & X_A(\mathbf{K}) & = \frac{D_{2B} - D_{1B}}{D_{1A} D_{2B} - D_{2A} D_{1B}} \\ X_B^{CSS} = & X_B(\mathbf{K}) & = \frac{D_{2A} - D_{1A}}{D_{1B} D_{2A} - D_{1A} D_{2B}} \end{array} \tag{4}$$

where $\mathbf{K} = (K_A, K_B)$ denotes the number of requests of each class. For values of β_A at the right of the saturation sector (i.e., with $\beta_A > \beta_A^U$) we have:

$$X_A(\mathbf{K}) = \frac{\beta_A}{D_{1A}} \quad X_B(\mathbf{K}) = \frac{1 - \beta_A}{D_{1B}} \tag{5}$$

Similar equations can be derived for values of β_A at the left of the saturation sector, thus obtaining the overall trend of the per-class throughputs as function of the population mix.

In [12], Rosti et al. also introduced the equi-load point, referred to as $\boldsymbol{\alpha}^*$, that is derived as:

$$\boldsymbol{\alpha}^* = \left(\alpha_A^* = \frac{D_{2B} D_{1B}}{D_{1A} + D_{2B} - D_{1B} - D_{2A}}, \alpha_B^* = 1 - \alpha_A^* \right) \tag{6}$$

Although two equally loaded stations may be expected to be also equally utilized, it is not the case since $\boldsymbol{\alpha}^* \neq \boldsymbol{\beta}^*$.

3 Scenario

We consider a system where each job consists of several independent tasks. Examples of such type of workloads are video transcoding/analysis, applications of business analytics, NoSQL queries and so on [4, 6]. In particular, we focus on jobs composed by two types of tasks, defined as class A and class B. For example, in multimedia stream applications each chunk is processed by a single task and the two classes may represent the computation of audio and video chunks, respectively.

We assume that the tasks of a job are executed sequentially by two resources, denoted as Res_1 and Res_2, e.g., the CPU and the storage of a server. The global time required by resource r for a complete execution of a class c task is referred to as *service demand* D_{rc}. The service demands characterize the workload in terms of total processing requirements to the resources, and their values are considered exponentially distributed. Each resource executes the concurrent tasks according to a *processor sharing* queuing discipline: all tasks are processed by the resources with a service rate proportional to the current number of tasks in service.

We call N_A the number of class A tasks, N_B the number of class B and $N = N_A + N_B$ the total number of tasks of a job. The system can execute at most K tasks concurrently. This limitation may be used to model, for instance, constraints on memory occupancy that prevents all the N tasks to be executed in parallel. In particular, the system is allowed to concurrently execute K_A tasks of class A and K_B of class B, with $K = K_A + K_B$. As soon as one task is completed, another task of the same class is admitted in execution, if available. When all the tasks of one class are completed, the system allows the processing of the remaining tasks of the other class.

Fig. 1 gives a visual representation of the considered scenario. We are interested in studying the total time T required to complete the execution of all the N tasks of a job that are initially into the pool. With the type of applications considered in this study we have $N_A \geq K_A$ and $N_B \geq K_B$.

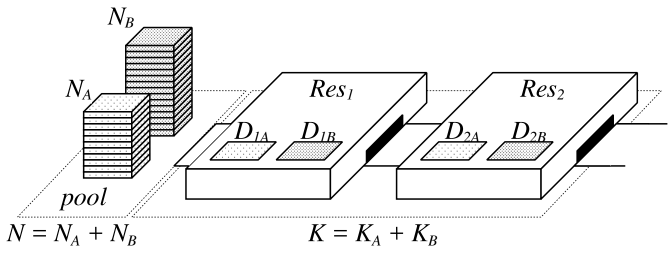

Fig. 1: The considered scenario.

Fig. 2 shows the temporal evolution of the number of tasks in the system. K out of N tasks immediately starts being serviced by the first resource of the system. As long as there are tasks of both classes in the pool waiting to be executed, the number of tasks in the system is constantly kept to $K = K_A + K_B$. We address this phase as Φ_{I}: after a short initial transient period, $T_{\tau 0}$, required to load K tasks into the first resource, the system behaves as a closed queuing model with K_A and K_B customers, since as soon as one of the task of a class leaves the system, it is replaced by another one of the same class. At time $T_{\Phi 1}$ the tasks in the pool of one class are finished and no new tasks of that class may enter the system (in Fig. 2 this happens for class A tasks). At this time the system starts replacing the tasks of the exhausted class with the ones of the other class: phase Φ_{II} begins. After an initial transient (that lasts until time $T_{\tau 12}$) in which all the remaining tasks of the exhausted class are executed, the system behaves as a closed queuing model with K customers (in Fig. 2 these customers are of class B). After some time, the tasks in the pool that still need to be executed are exhausted, and the system begins to execute a decreasing number of tasks. We denote this instant of time by $T_{\Phi 2}$, and the period of time in which the server is working with less than K tasks as phase Φ_{III}. The job completes its execution when all its tasks are terminated at time T, referred to as *depletion time*.

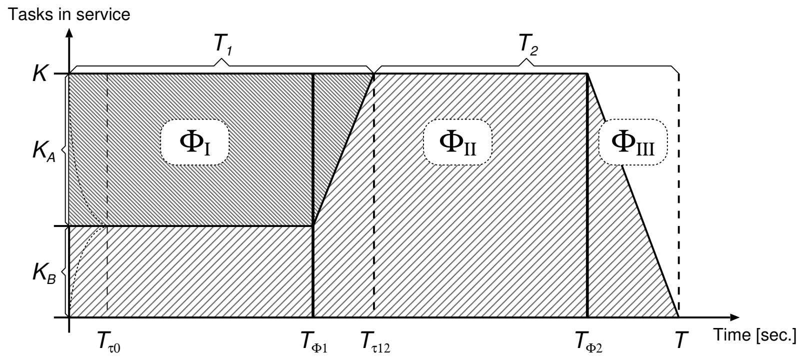

Fig. 2: Behavior of the number of tasks in execution with phases characterization.

4 Models analysis

PDS cannot be analyzed through closed-form formulas when the asymptotic assumptions presented in [3] (i.e., $N \gg K$ and K is large enough to saturate the system) are not satisfied. In this Section, PDS are analyzed using analytic, discrete event simulation and fluid models. In particular, the three models are described and compared to each other in order to determine those that may provide the more accurate results in the shorter time.

4.1 Markov analysis

A continuous-time Markov chain (CTMC) model was proposed in [2] to analytically study a PDS. Such model provides the exact results since it analytically describes each phase of a PDS. In the CTMC the state of the system is described by a tuple counting, for each class, the current number of tasks in the pool, in station Res_1 and in station Res_2. The completions of tasks change the system state and are represented in the CTMC as transitions from a value of the tuple to another. Assuming that all such events follow an exponential distribution, the whole state space of the system is built and the resulting CTMC can be analyzed by standard numerical techniques. In particular, the depletion time and the duration of each phase, including the transient ones, are analytically computed through absorption time analysis. Further details can be found in [2] and [10].

Unfortunately, analytic analysis of CTMCs suffers of the well-known state space explosion problem: large values of K and N make the resulting state space grow exponentially, thus making the solution computationally unfeasible. For this reason, also simulation and fluid models are considered for PDS analysis.

4.2 Discrete event simulation

The single-subsystem PDS may be analyzed recurring to the multi-formalism model in Fig. 3.

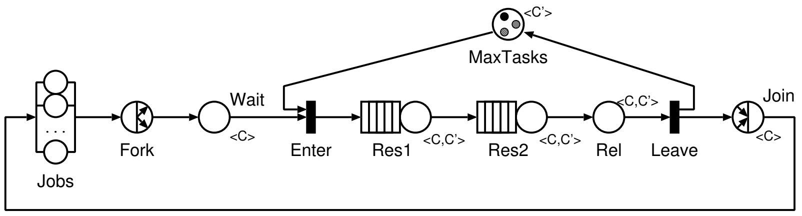

Fig. 3: The multi-formalism model of the considered scenario

Such a model consists of a Colored Petri Net (CPN) and a multi-class fork-and-join queuing network. In fact, the adoption of several formalisms allows to use the most appropriate primitives while modeling a system [7]. The workload of the model is composed by tokens and jobs: the former are used by CPN to model the subsystem finite capacity, the latter by queuing network to represent the tasks of a job and their execution. Note that, since a two-class PDS is considered, there are two classes both for tokens and for jobs.

Initially, a single job is in the system starting in the delay station `Jobs` with zero service time: the purpose of this node is to provide the reference station[2] for computing throughput and response time, and to restart the system after one execution run. The job is immediately split into N_A class A tasks and N_B class B tasks (i.e., $N = N_A + N_B$) by the fork node named `Fork`. Thus, a task that is waiting to be executed is represented as a token in place `Wait`: a color class $< C >$ is assigned to each token to identify the class of that task. The capacity $K = K_A + K_B$ of the subsystem is represented by the number of color $< C' >$ tokens in place `MaxTasks` that are initialized to K_A and K_B for the corresponding task classes. The notation used in Fig. 3 is summarized in Table 1.

Table 1: Color-sets of the discrete event multi-formalism model.

Color-set	**Description**
$< C >$	Task class $C = \{A, B\}$.
$< C' >$	Token color $C' = \{A, B\}$.
$< C, C' >$	Task class C, token color C'

[2] In closed queuing networks throughput and response time can be computed only with respect to a given/specific resource, the so-called reference station [9].

The subsystem starts executing a task when transition `Enter` fires; in fact, this can occur in one of the four different modes shown in Table 2. When a color class A (B) task is waiting in place `Wait` and at least a token of the same color is available in place `MaxTasks`, the system is in phase Φ_{I} and transition `Enter` fires in mode 1 (2). Instead, if system is operating in phase Φ_{II} or Φ_{III}, one of the two classes of tasks is exhausted and its tokens are used to allow the remaining tasks to be executed. In this case, transition `Enter` fires in mode 3 or 4, according to the color class $< C >$ of tasks still waiting to be executed.

Table 2: Firing modes for transitions `Enter` and `Leave`.

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

As shown in Table 1, the color class of a task is referred to as $< C >$, whereas $< C' >$ is used to represent the color of a token. Instead, notation $< C, C' >$ is used to represent the tasks admitted into the subsystem; indeed, a token of color $< C' >$ is always associated to each of these tasks.

The subsystem is composed by two queuing network primitives, `Res1` and `Res2`. They represent the resources of the subsystem and process the tasks currently into it. The service requirements of each task are determined by its color class $< C >$, while the color $< C' >$ of the token associated to that task is used only to correctly return the token in place `MaxTasks`. When a task has been executed by both the resources, it enters place `Rel` and enables transition `Leave`.

The absence of tokens in place `MaxTasks` means that the subsystem has reached its maximum capacity. In such condition further tasks are not admitted until the completion of at least a task currently in execution. When a task is completed, transition `Leave` fires: a token return to place `MaxTasks` and the task is sent to the join node `Join`. Also transition `Leave` can fire in four different modes depending on the color class of the task that has been completed and the color of its associated token. The four modes are shown in Table 2. When all the N tasks generated by `Fork` have been collected by `Join`, the initially job return to the delay station and a new one can start its computation.

Note that, the approximations introduced by the multi-formalism model are related to the technique adopted for its simulation.

4.3 Fluid approximation

The fluid approximation is based on the following assumption: except from the transient behavior that occurs whenever the system switches phases, if we focus only on the service components (namely `Res1` and `Res2`), they operate as in a two class closed model with K_A and K_B jobs. This occurs because as soon as a job leaves the server components, a new one of the same class is re-introduced. Let us call $X_A^{K_A,K_B}$ and $X_B^{K_A,K_B}$ the throughput that the two classes would have in a closed model composed by `Res1` and `Res2`, with K_A and K_B jobs. The time $T_{\Phi 1}$ at which the first phase (Φ_I) ends can thus be approximated as:

$$T_{\Phi 1} = \min\left(\frac{N_A - K_A}{X_A^{K_A,K_B}}, \frac{N_B - K_B}{X_B^{K_A,K_B}}\right) \tag{7}$$

Without loss of generality, let us suppose that class A ends first: $\frac{N_A-K_A}{X_A^{K_A,K_B}} < \frac{N_B-K_B}{X_B^{K_A,K_B}}$. The number $N_{B\Phi 1}$ of class B jobs that still can enter the system at the end of phase Φ_I, can then be determined as:

$$N_{B\Phi 1} = N_B - K_B - X_B^{K_A,K_B} \cdot T_{\Phi 1} \tag{8}$$

For sake of simplicity, let us suppose that $N_{B\Phi 1}$ is large enough to allow the system to complete the transient part up to time $T_{\tau 12}$ (Fig. 2). We approximate the $T_{\tau K_A-1,K_B+1}$ time at which the first of the K_A jobs still in the system ends as:

$$T_{\tau K_A-1,K_B+1} = T_{\Phi 1} + \frac{1}{X_A^{K_A,K_B}} \tag{9}$$

and compute the number $N_{B\tau K_A-1,K_B+1}$ of class B jobs that are still in the pool and need to enter the servers as:

$$N_{B\tau K_A-1,K_B+1} = N_{B\Phi 1} - \frac{X_B^{K_A,K_B}}{X_A^{K_A,K_B}} \tag{10}$$

At this point the population in the system changes to $K'_A = K_A - 1$ and $K'_B = K_B + 1$ since the class A job is replaced by a class B one. We thus compute $X_A^{K_A-1,K_B+1}$ and $X_B^{K_A-1,K_B+1}$ by solving the corresponding closed model with the new population mix, and compute the time $T_{\tau K_A-2,K_B+2}$ at which the next class A job finishes and the corresponding class B population $N_{B\tau K_A-2,K_B+2}$ that still have to enter the system:

$$\begin{aligned} T_{\tau K_A-2,K_B+2} &= T_{\tau K_A-2} + \frac{1}{X_A^{K_A-1,K_B+1}} \\ N_{B\tau K_A-2,K_B+2} &= N_{B\tau K_A-1,K_B+1} - \frac{X_B^{K_A-1,K_B+1}}{X_A^{K_A-1,K_B+1}} \end{aligned} \tag{11}$$

The process is repeated $K - K_A$ times until time $T_{\tau 12}$ is reached (i.e., all the class A jobs inside the system and in the pool are completed). At that point, the systems has only class B jobs. We thus solve the closed model as a single class one (considering only class B) with K jobs, and determine its throughput $X_B^{0,K}$. Let us call $N_{B\tau 12}$ the number of class B jobs still waiting to be executed in the pool (computed with the previous procedure repeating until $N_{B\tau 12} = N_{B\tau 0,K}$). We then approximate the end of the second phase $T_{\Phi 2}$ as:

$$T_{\Phi 2} = T_{\tau 12} + \frac{N_{B\tau 12}}{X_B^{0,K}} \tag{12}$$

Now depletion starts and the number of class B jobs inside the system decreases from K down to zero. Let us call $X_B^{0,k}$ the throughput of the closed model when its population is composed by k class B jobs. The process completion time T is thus approximated as:

$$T = T_{\Phi 2} + \sum_{k=1}^{K} \frac{1}{X_B^{0,k}} \tag{13}$$

The fluid approximation might become slightly more complex when the number of class B jobs waiting in the pool becomes zero while there are still class A jobs in service. In this case we will have again to consider the minimum time between class A jobs and class B jobs, and compute the throughput accordingly.

In practice, depending on the system configuration, all the throughput of a closed model with $k_A + k_B \leq K$ jobs might be required. However, thanks to the properties of the Mean Value Analysis algorithm, all these values can be easily computed with complexity $o(K^2)$, and a minor overhead with respect to the computation of the solution for K_A and K_B alone. Algorithm 1 summarizes the proposed procedure: variables n_A and n_B contain the fluid count of the total population (inside the servers and in the pool) for the two classes, while k_A and k_B the current population inside the server. Lines 3 to 10 considers phases Φ_I and Φ_{II} (see Fig. 2), while lines 11 to 19 deals with phase Φ_{III}, since after line 10 either $k_A = 0$ or $k_B = 0$.

4.4 Techniques comparison

The three techniques previously presented are now compared while analyzing a PDS with a single subsystem characterized by the service demand matrix in Eq. (1). In particular, the accuracy of each model in evaluating the PDS performance is analyzed taking into account the time required to compute the results. For this purpose, the number of tasks that must be executed and the capacity of the subsystem increase, while their ratio is constant and set to $K/N = 0.1$.

The shortest depletion time is expected to be observed when the number of tasks into the pool (N_A and N_B) and subsystem (K_A and K_B) are initialized to their optimal population mix, independently of the values of N and K. The pool and subsystem

Algorithm 1 Fluid approximation of T

```
 1: Compute $X_A^{k_A,k_B}$ and $X_B^{k_A,k_B}$, $\forall k_A, k_B > 0 : k_A + k_B \leq K$ using MVA
 2: $k_A = K_A, k_B = K_B, n_A = N_A, n_B = N_B, T = 0$
 3: while ($k_A > 0$ and $k_B > 0$) do
 4:     $\Delta T_A = (n_A - k_A + 1)/X_A^{k_A,k_B}$, $\Delta T_B = (n_B - k_B + 1)/X_B^{k_A,k_B}$
 5:     if $\Delta T_A < \Delta T_B$ then
 6:         $n_A$-= $\Delta T_A \cdot X_A^{k_A,k_B}$, $n_B$-= $\Delta T_A \cdot X_B^{k_A,k_B}$, $k_A$--, $k_B$++, $T$+= $\Delta T_A$
 7:     else
 8:         $n_A$-= $\Delta T_B \cdot X_A^{k_A,k_B}$, $n_B$-= $\Delta T_B \cdot X_B^{k_A,k_B}$, $k_A$++, $k_B$--, $T$+= $\Delta T_B$
 9:     end if
10: end while
11: if $k_A > 0$ then
12:     while ($k_A > 0$) do
13:         $\Delta T_A = (n_A - k_A + 1)/X_A^{k_A,0}$, $n_A$-= $\Delta T_A \cdot X_A^{k_A,k_B}$, $k_A$--, $T$+= $\Delta T_A$
14:     end while
15: else
16:     while ($k_B > 0$) do
17:         $\Delta T_B = (n_B - k_B + 1)/X_B^{0,k_B}$, $n_B$-= $\Delta T_B \cdot X_B^{k_A,k_B}$, $k_B$--, $T$+= $\Delta T_B$
18:     end while
19: end if
```

optimal population mixes (i.e., $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$, respectively) have been shown to coincide with equi-load and equi-utilization points [3] and may be derived through Eqs. (6) and (2). They only depend on the service demand matrix of the PDS and, considering the one given in Eq. (1), are $\boldsymbol{\alpha}^* = (0.556868, 0.443131)$ and $\boldsymbol{\beta}^* = (0.4, 0.6)$.

Results depicted in Fig. 4 are obtained studying the PDS with the service demand matrix given in Eq. (1) and for pool population mix $\alpha = \alpha^*$. The PDS is studied for $N = 100$ and $K = 10$, while values of K_A and K_B vary. Fig 4 depicts the depletion time T of the configuration considered as a function of the subsystem population mix $\boldsymbol{\beta}$; as expected, the minimum depletion time is measured by all the models for $\boldsymbol{\beta} = \boldsymbol{\beta}^*$. Moreover, the largest error in estimating the depletion time made by simulation and fluid models with respect to the analytic one are 1‰ and 4%, respectively.

Fig. 5 represents the ratio of depletion time to the number of tasks initially into the pool T/N – or normalized depletion time – obtained with fluid model for configurations $(N, K) = \{(100,10), (1000,100), (10000,1000), (100000,10000)\}$. When the pool size and the subsystem capacity are large enough, we can observe the presence of an interval of value of β_a where the normalized depletion time is minimized and constant.

Fig. 6 shows the normalized depletion time T/N as a function of subsystem population mix, when number of tasks initially into the pool changes, but the subsystem capacity is always the same (i.e., $K = 10$). In this case, the normalized depletion time behaves at the same way for all the configurations with $N > 1000$. As shown, the depletion time may be shortened by 30% if $N > 1000$ and PDS works with its optimal pool and subsystem population mixes (i.e., $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$).

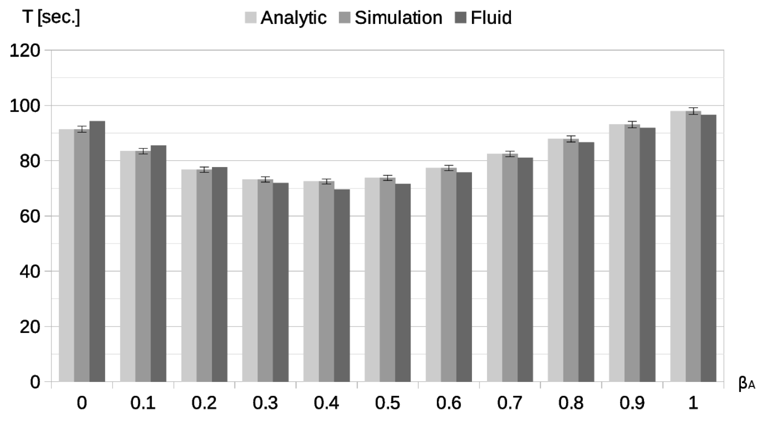

Fig. 4: Depletion time of a PDS as a function of the subsystem population mix obtained using three different models.

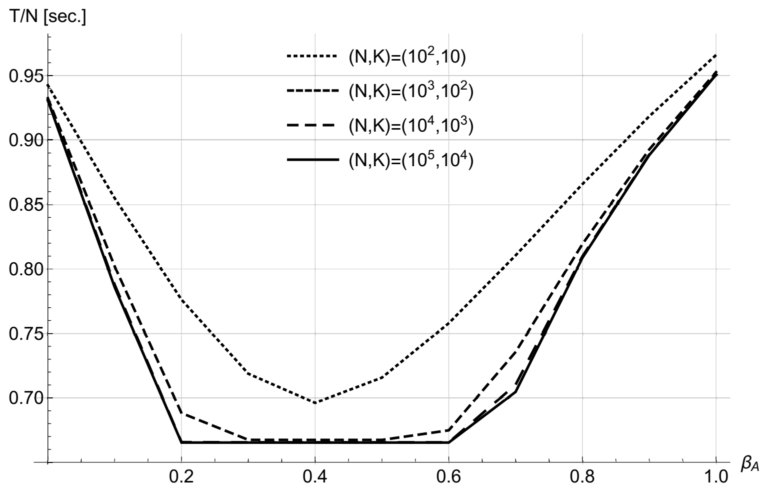

Fig. 5: Ratio of depletion time to the number of tasks initially into the pool, for N and K such that $K/N = 0.1$.

In the PDS analysis, the fluid model provides results in the shortest time and is capable to handle very large values of N and K. Furthermore, although analytic model is faster than the simulation one, it cannot manage large values of pool size and subsystem capacity (i.e., $N > 100$ and $K > 10$) due to state space explosion. For these reasons, we used all the models to study configuration $(N, K) = (100, 10)$, simulation and fluid models for considering complex system and only the fluid one while analyzing very large values of N and K. For further details about performance of the three models, the reader is referred to Table 3 where the total time and the

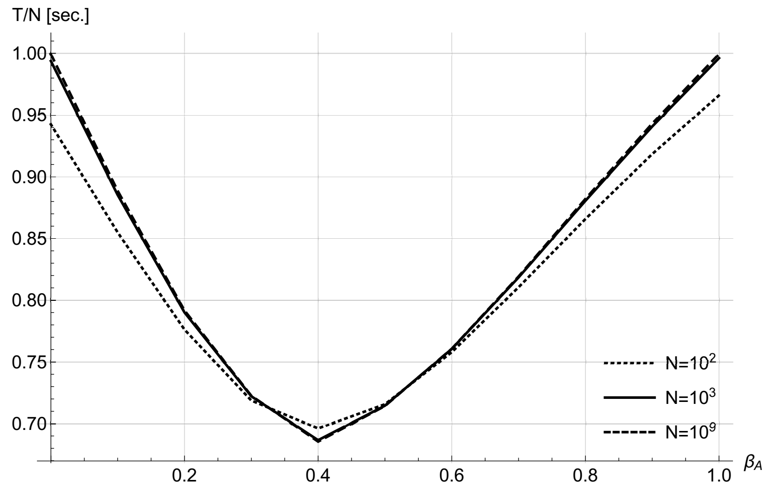

Fig. 6: Ratio of depletion time to the number of tasks initially into the pool, for $K = 10$ and different values of tasks initially into the pool.

maximum Mean Absolute Percentage Error (MAPE) in computing the depletion time of each configuration are reported for all the techniques. In particular, the MAPE made by the simulation model with respect to the analytic one (that provides exact solutions) while computing the depletion time of the PDS is derived as:

$$\text{MAPE}_{sim} = \frac{|T_{sim} - T_{anal}|}{T_{anal}} \tag{14}$$

where T_{sim} and T_{anal} are the depletion times observed with simulation and analytic models respectively. Similarly, the MAPE made by the fluid model with respect to the simulation one is:

$$\text{MAPE}_{fluid} = \frac{|T_{fluid} - T_{sim}|}{T_{sim}} \tag{15}$$

where T_{fluid} is the depletion time estimated by the fluid model.

In Table 3, columns 1 and 2 report, respectively, the values of N and K for which the PDS is studied; the time required by each model to provide the depletion time of the system for different population mixes of the subsystem (i.e., $\beta_A = i/10$, with $i = (0, \ldots, 10)$), is shown in columns 3, 4 and 5; finally, the maximum MAPE made by each simulation and fluid models with respect to the analytic and the simulation ones, respectively, are shown in columns 6 and 7. Since the results obtained using discrete event simulation are very similar to those derived through Markov analysis and, differently from analytic model, the simulation one may be used to study configurations with large values of N and K, the fluid model results are compared to those of the simulation model. Gray cells in Table 3 mean that results are not available due to the long time required for their computation.

While analytic and simulation models performance is affected by values of N and K, the time required by fluid model to provide results mainly depends on values of K. For this reason, adopting fluid model to analyze PDS allows the users to largely increase pool size N. Moreover, even if K affects the performance of fluid model more than N, it is still possible to consider also large values of K and get results in a short time. Finally note that, although maximum MAPE$_{fluid}$ is between 4% and 6% when $K/N = 0.1$, it decreases (i.e., the fluid model's accuracy improves) when K/N is small enough, such as for $N = 10^4$ and $K = 10$.

Table 3: Models execution time and accuracy. A cell is gray if it has not been possible to derive these measures due to long execution time.

N	K	Execution time			Errors	
		Markov chain	Simulation	Fluid approx.	MAPE$_{sim}$	MAPE$_{fluid}$
10^2	10	5 min.	55 min.	38 ms	1‰	4%
10^3	10		9 hr.	34 ms		2.6%
10^3	10^2		9 hr.	38 ms		5.5%
10^4	10		3 days	22 ms		5‰
10^4	10^3		7 days	590 ms		6%
10^5	10			67 ms		
10^5	10^4			53 sec.		
10^6	10			231 ms		
10^7	10			2 sec.		
10^8	10			17 sec.		
10^9	10			3 min.		
10^9	10^5			4 min.		

5 Conclusions

In this chapter the performance of three different techniques used to study PDS were analyzed and compared. The CTMC model provides analytic results, but it cannot deal with large and complex systems due to the well-known state space explosion. Thus, a discrete event multi-formalism model has been adopted to study more complex systems. In fact, it can analyze PDS with large pool size and subsystem capacity, while deriving results with high accuracy. Unfortunately, it may require a long computation time to provide results. Finally, a fluid model has been proposed in order to analyze complex systems in a short time. Although its MAPE with respect to simulation model has been observed between 4% and 6% for larger values of K/N, the results are provided in few minutes also for very large values of N and K, and its accuracy increases when K/N is small.

Although all the models presented in this chapter provides very similar results in different amounts of time, they must still be validated against a real scenario. In fact, the next step of our research will be the validation of the analytic, simulation and fluid models against a PDS deployed on a real cloud environment.

Acknowledgements

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

References

1. F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, Apr. 1975.
2. D. Cerotti, M. Gribaudo, R. Pinciroli, and G. Serazzi. Stochastic analysis of energy consumption in pool depletion systems. In *Measurement, Modelling and Evaluation of Dependable Computer and Communication Systems - 18th International GI/ITG Conference, MMB & DFT 2016, Münster, Germany, April 4-6, 2016, Proceedings*, pages 25–39, 2016.
3. D. Cerotti, M. Gribaudo, R. Pinciroli, and G. Serazzi. Optimal population mix in pool depletion systems with two-class workload. In *proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 11–18. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2017.
4. V. T. Chakaravarthy, A. R. Choudhury, S. Roy, and Y. Sabharwal. Scheduling jobs with multiple non-uniform tasks. In *European Conference on Parallel Processing*, pages 90–101. Springer, 2013.
5. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
6. D. Díaz-Sánchez, A. Marín-López, F. Almenarez, R. Sánchez-Guerrero, and P. Arias. A distributed transcoding system for mobile video delivery. In *Wireless and Mobile Networking Conference (WMNC), 2012 5th Joint IFIP*, pages 10–16. IEEE, 2012.
7. M. Gribaudo and M. Iacono. *Theory and application of multi-formalism modeling*. IGI global, 2013.
8. L. Huang, X.-w. Wang, Y.-d. Zhai, and B. Yang. Extraction of user profile based on the hadoop framework. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on*, pages 1–6. IEEE, 2009.
9. E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984.
10. J. Muppala, M. Malhotra, and K. Trivedi. Markov dependability models of complex systems: Analysis techniques. In S. Ozekici, editor, *Reliability and Maintenance of Complex Systems*, volume 154, pages 442–486. Springer Berlin Heidelberg, 1996.
11. R. Pinciroli, M. Gribaudo, and G. Serazzi. Modeling multiclass task-based applications on heterogeneous distributed environments. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 166–180. Springer, 2017.
12. E. Rosti, F. Schiavoni, and G. Serazzi. Queueing network models with two classes of customers. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1997. MASCOTS'97., Proc. Fifth Int. Symp. on*, pages 229–234. IEEE, 1997.