Evaluating Replication for Parallel Jobs: an Efficient Approach

Zhan Qiu, *Member, IEEE* and Juan F. Pérez, *Member, IEEE*

Abstract—Many modern software applications rely on parallel job processing to exploit large resource pools available in cloud and grid infrastructures. The response time of a parallel job, made of many subtasks, is determined by the last subtask that finishes. Thus, a single laggard subtask or a failure, requiring re-processing, may increase the response time substantially. To overcome these issues, we explore concurrent replication with canceling. This mechanism executes two job replicas concurrently, and retrieves the result of the first replica that completes, immediately canceling the other one. To analyze this mechanism we propose a stochastic model that considers replication at both job-level and task-level. We find that task-level replication achieves a much higher reliability and shorter response times than job-level replication. We also observe that the impact of replication depends on the system utilization, the subtask reliability, and the correlation among replica failures. Based on the model, we propose a resource-provisioning strategy that determines the minimum number of computing nodes needed to achieve a service-level objective (SLO) defined as a response-time percentile. This strategy is evaluated by considering realistic traffic patterns from a parallel cluster, where task-level replication shows the potential to reduce the resource requirements for tight response-time SLOs.

Index Terms—Parallel-job processing, Performance analysis, Quality of Service.

1 Introduction

Parallel job processing has become a common feature of many software systems. For instance, many scientific applications and data processing frameworks rely on splitting a job into many subtasks, taking advantage of the large resource pools offered in grid and cloud infrastructures. When processing parallel jobs, the response time is determined by the last subtask to finish, a factor that can amplify the variability in the observed response times significantly, as observed in Google services [1]. This effect is further intensified when the subtasks that compose a job fail, creating an overall unreliable job out of many highly-reliable subtasks. A common approach to deal with unreliable subtasks is to re-execute them if the response is not obtained after a timeout. This however increases the response times and their variability, harming latency-sensitive applications [2].

A different alternative to deal with this problem is by *concurrent replication with canceling*, where the job and a replica are executed concurrently on different resources, and once a result is obtained from either, the outstanding replica is canceled. Concurrent replication, without canceling, has been proposed recently [1], [3], [4], [5] in different settings, including computing clusters, DNS servers, and TCP connection establishment. Its appeal lies in that a result can be provided as soon as one of the replicas completes, avoiding delays caused by waiting for a timeout and re-executing. However, in computing clusters its main drawback lies in the additional resources required to process the replicas concurrently, increasing the load on the servers and potentially leading to longer response times. This effect can be mitigated by the introduction of replica cancellation, as the resources used by a job and its replica are freed as soon as one of them completes service. In addition, concurrent replication is appealing in the light of the low utilization common in data centers, with reported average utilization of 18% [6]. Also, traces released by Facebook reveal median CPU and memory utilization under 20% [3].

While effective to improve reliability, it is still unclear up to what extent replicating parallel jobs in computing clusters can provide gains in terms of response times, particularly for the tail of the response-time distribution. To better understand this effect, we propose a stochastic model to determine the *response-time distribution* in a system that processes synchronous parallel jobs and uses replication to improve reliability. The main features of the model are

- It considers both task-level and job-level replication with canceling, as well as the no-replication case, in a single framework, as described in Section 4.
- It explicitly considers correlation between replica failures, covering cases from fully-independent to perfectly-correlated failures.
- It allows for general job arrival processes that can display different levels of variability and auto-correlation.
- In Section 5 we develop an efficient method to solve this model, enabling us to consider jobs with a large number of subtasks.

An early version of this model was presented in [7], which focused on the *mean* response time only, and was limited to independent replica failures. Instead, in this paper we focus on the response-time *distribution*, which requires different

- *Z. Qiu is with the Department of Computing, Imperial College London, UK.*
  E-mail: z.qiu11@imperial.ac.uk
- *Juan F. Pérez is with the Department of Mathematics and Statistics, University of Melbourne, Australia.*
  E-mail: juan.perez@unimelb.edu.au

The research of Juan F. Pérez is supported by the ARC Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS).

modeling and solution approaches.

Making use of this model, Section 6 evaluates the impact of replication on the offered response times, and how this is affected by the system parameters. We find that concurrent replication with canceling, compared to its counterpart *without* canceling, significantly reduces the resource utilization and delivers shorter response times, making fault-tolerance via replication more applicable. While both job-level and task-level replication improve the system reliability, task-level replication achieves a much higher reliability while introducing a lower load and delivering shorter response times. Further, we observe that the impact of replication is affected not only by the system utilization, but also by the subtask reliability and the correlation among replica failures. Under low to medium utilizations, task-level replication is able to improve the response times, as long as the subtask reliability is high. Scenarios with low subtask reliability benefit from replication in terms of reliability, but the room for improving the response times is more limited.

In Section 7 we rely on the proposed model to devise a resource-provisioning strategy that determines whether replication should be adopted or not, and obtains the minimum number of computing nodes needed to comply with a service-level objective (SLO) [8]. We consider SLOs on the percentiles of the response-time distribution or on moments such as mean or variance. In Section 8 we evaluate this resource-provisioning strategy by considering realistic traffic patterns from a parallel cluster, making use of logs from the RICC Cluster [9]. The results highlight that ignoring the variability in the arrival process at the time of provisioning may lead to large SLO violations. We also observe that, in addition to the gains in reliability, task-level replication has the potential to reduce the resource requirements under tight response-time SLOs. Before introducing the model, we review related works in Section 2 and introduce the reference model and replication strategies in Section 3.

2 Related Work

When processing parallel jobs a subtask may fail during service due to a number of reasons, such as errors in the input data, the detection of a deadlock, communication failures, among others. Due to the scale and complexity of large-scale parallel systems, it is infeasible to eliminate all possible failures [1], [10]. A traditional fault-tolerance mechanism to handle request failures is to re-execute the request after a timeout [10], [11]. Although effective to handle failures, this approach introduces additional delays that degrade the response times offered by the application. This effect is particularly critical for latency-sensitive applications, for which all subtasks must complete within a strict deadline, in order for the application to be responsive [1]. Concurrent replication has been recently considered [1], [3], [4], [5] to reduce latency in different settings, issuing replicas to diverse resources, and using the result from whichever replica responds first. For instance, [3] proposes to run multiple copies of small interactive jobs to mitigate the effect of latency, showing a significant reduction in the mean response time, at a small cost in additional resources. [4] analytically characterizes the conditions under which redundant requests help in reducing latency, focusing on the *mean* response times of fully reliable jobs. [10] demonstrates replication to be an effective way to achieve robustness under unpredictable failures while limiting latency. Replication has also been considered at the operating-system level [12], [13], at both thread and process levels, where the latter shows a better performance thanks to its lower overhead. However, the execution of concurrent replicas may increase the resource utilization, and the response times, beyond desirable levels. To curb the additional load created by concurrent replication, [1] proposes to cancel any outstanding replicas in process immediately after the first one completes. This approach, referred to as replication with canceling, was analyzed for the case of single-task jobs with two independent queues and servers in [14], as well as for synchronous parallel processors in [7], [15], which focused on the *mean* response time. Instead, in this paper we look into the response-time *distribution*, and especially on its tail. Further, we remove the assumption of independent replica failures, explicitly modeling their correlation.

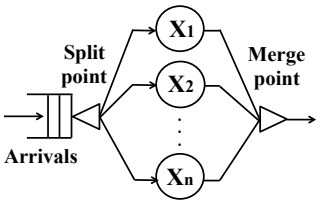


Fig. 1: Split-merge queue

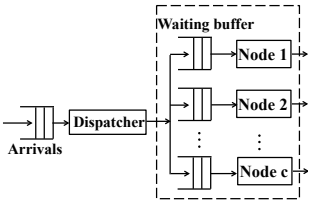


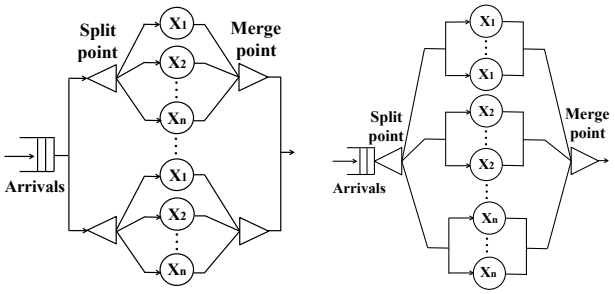
Fig. 2: Reference Model

To model the execution of synchronous parallel jobs, and their replicas, we make use of split-merge queues. In a split-merge queue, shown in Figure 1, an incoming job joins a queue and, upon reaching the head of the queue, splits into multiple subtasks, which are processed simultaneously by a set of parallel processors. These processors are all blocked until all the subtasks finish service, at which point the subtasks rejoin and depart from the system. Split-merge queues have been studied, for example, in [16], which determined the job service time assuming exponential task service times, and introduced an approximation for general heterogeneous service times. Also, [17] derived an approximation for the response-time distribution in a split-merge queue with general service times and Poisson arrivals. These results however do not consider task failures nor replication, which are the focus of this paper.

3 Background

3.1 Reference Model

We consider a system consisting of a central dispatcher and c distributed and homogeneous computing nodes, as shown in Figure 2, where jobs that arrive at the dispatcher are assigned to the computing nodes in either a round-robin or random fashion. The advantage of these scheduling policies is that jobs are distributed evenly among the nodes, and no communication is required between the nodes and the dispatcher about the nodes' state. A job starts service directly if the corresponding node is idle, or joins the end of the queue in front of the node, if it is busy, and waits until all the jobs in front are processed, with first-come first-served (FCFS) scheduling. Each node is composed of



(a) Job-level Replication (b) Task-level Replication

Fig. 3: Reference job-level and task-level models

multiple processors, such that, when a job starts service, each of its n subtasks is assigned to one of the processors in the corresponding node. The number of subtasks in a job, n, is also referred to as the *job size*. The subtasks processing times are assumed to be exponentially distributed with rate μ. Subtasks in service are subject to failures, with the time to failure being exponentially distributed with rate α. When a subtask fails during service, and no replication is adopted, the whole job fails and leaves the computing node. We assume that the node is not affected by the failure, and continues to serve the next job in front of the queue. To increase the system reliability, we propose to *concurrently* process job replicas, and consider two levels of replication granularity, as described next.

3.2 Replication Granularity

For each arriving job, $r \geq 1$ replicas are submitted concurrently to the central dispatcher. In this paper we consider the cases $r = 1$, where no replication is adopted, and $r = 2$, where a total of two job replicas are submitted, including the original copy. We consider the two replication granularity levels proposed in [3] for parallel jobs. The first option is job-level replication, where for every job submitted to the cluster, two replicas are spawned and processed independently, as shown in Figure 3(a). Once the result from the earliest job-replica is obtained, the other one is canceled immediately if it is still in service. Job-level replication is appealing due to its simplicity, but each job-replica is successful only if all its subtasks are successful. The second alternative is to clone at the subtask level, as shown in Figure 3(b), where every subtask is cloned and the result of the first replica to complete is returned, canceling its sibling if still in process. In this case it is enough that a single replica of each subtask succeeds to have a successful job completion.

In the system without replication, we assume the cluster is composed of c nodes and cn processors in total, such that it can process c jobs, each of size n, in parallel. If two replicas are adopted, either at job or task level, then every two nodes, i.e., $2n$ processors, are grouped together, allowing the processing of $c/2$ concurrent replicated jobs.

In Section 4 we show that, while the *subtask* processing times are exponentially distributed, the *job* processing times follow a phase-type (PH) distribution. Further, to consider the variability and auto-correlation of the job inter-arrival times (IATs) observed in compute clusters, we model the job arrival process as a Markovian Arrival Process (MAP). We thus introduce PH distributions and MAPs.

3.3 Phase-type Distributions

Consider a Markov chain with $n+1$ states where the first n states are transient and the state $n+1$ is absorbing. The generator matrix of this chain can be written as $\begin{bmatrix} B & \boldsymbol{b} \\ \mathbf{0}' & 0 \end{bmatrix}$, where the matrix B holds the transition rates among the n transient states, and the exit vector $\boldsymbol{b}{=}{-}B\mathbf{1}$ holds the rates at which the chain jumps into the absorbing state. Here $\mathbf{1}$ is a column vector of ones, $\mathbf{0}$ a column vector of zeros, and $'$ stands for the matrix transpose. A phase-type (PH) distribution [18] is defined as the distribution of the time to absorption X in this chain. We denote it as PH($\boldsymbol{\tau}, B$), where $\boldsymbol{\tau}$ is the $1{\times}n$ vector holding the initial probability distribution with which the chain starts in any of the n transient states. The cumulative distribution function (CDF) of the time to absorption is $F(x){=}1{-}\boldsymbol{\tau}\exp(Bx)\mathbf{1}$, for $x \geq 0$, and the expected absorption time is $E[X]{=}{-}\boldsymbol{\tau}B^{-1}\mathbf{1}$.

3.4 Markovian Arrival Processes

Markovian arrival processes (MAPs) were introduced in [19] as a generalization of PH distributions to represent correlated point processes with PH inter-event distributions. The continuous-time MAP [18] is a marked Markov chain with generator matrix $D{=}D_0{+}D_1$, where the elements of D_0 and D_1 represent transitions without and with arrivals, respectively. D_1 is a non-negative matrix, and D_0 has non-negative off-diagonal entries and strictly negative diagonals, such that $(D_0{+}D_1)\mathbf{1}{=}\mathbf{0}$. As an example, consider a MAP with two phases and matrices

$$D_0 = \begin{bmatrix} -\lambda_1 & 0 \\ 0 & -\lambda_2 \end{bmatrix},\ D_1 = \begin{bmatrix} 0 & \lambda_1 \\ \lambda_2 & 0 \end{bmatrix}.$$

When the arrival process is in the first phase, a job arrives with rate λ_1, which causes the process to jump to the second phase, where a job arrives instead with rate λ_2, triggering the process back to phase one. The mean arrival rate is $\lambda{=}\boldsymbol{\gamma}D_1\mathbf{1}$, where $\boldsymbol{\gamma}$ is the stationary distribution of the underlying Markov chain, i.e., $\boldsymbol{\gamma}D{=}0$ and $\boldsymbol{\gamma}\mathbf{1}{=}1$. A MAP can represent a renewal process with PH($\boldsymbol{\tau}, B$) inter-arrival times by setting $D_0{=}B$ and $D_1{=}\boldsymbol{b}\boldsymbol{\tau}$.

4 The Job Service-time Distribution

The job service time is the interval between the time that all the job subtasks enter a node's processors, and the time the job leaves the node, either with service completion or failure. In this section we show that, under the assumption of exponential service and failure times for the subtasks, the *job* service time follows a PH distribution with representation $(\boldsymbol{\alpha}_{\text{ser}}, S_{\text{ser}})$, either with or without replication. Further, we consider the case where failures across replicas are positively correlated, i.e., the failure of a replica makes more likely the failure of its partner. Finally, we show how the analysis of a tagged node can be extended to the multi-node case under round-robin scheduling.

4.1 No-replication Model

In the no-replication case, the job service state is described by $N_1(t)$, the number of subtasks in service at time t. The set of service states, or phases, is thus

N_J=$\{n_1|n_1\in\{1,2,\ldots,n\}\}$, and the job leaves the system when either all its subtasks complete service successfully, or one of them fails. The job service time thus has a PH($\boldsymbol{\alpha}_{\text{ser}}, S_{\text{ser}}$) representation, where N_J is the set of transient states, and we consider two absorbing states, S and F, representing the cases where the job completes service successfully or encounters a failure, respectively. The absorption vectors $\boldsymbol{t}_S$ and $\boldsymbol{t}_F$ hold the absorption rates into states S and F, respectively, and $\boldsymbol{t}$=$\boldsymbol{t}_S$+$\boldsymbol{t}_F$. If a job is in phase n_1>1, a subtask service completion leads the job to phase n_1-1 with rate $n_1\mu$. The (non-diagonal) entries of S_{ser} are thus $S_{\text{ser}}(n_1, n_1-1)$=$n_1\mu$. If n_1=1, the subtask service completion leads the job to a successful completion, thus to absorption in phase S. Instead, if the job is in phase $n_1 \in N_J$ and a subtask fails, the service process is absorbed in phase F with rate $n_1\alpha$. Finally, a job always starts service in phase n, as all its subtasks start service at the same time, thus the initial probability vector is such that $\boldsymbol{\alpha}_{\text{ser}}(n)$=1, and all its other entries are zero. As an example, consider the case with n=2 subtasks, where the service phases are N_J=$\{2,1\}$. Letting β=α+μ, the matrix $[S_{\text{ser}}|\boldsymbol{t}_S, \boldsymbol{t}_F]$ is

$$\begin{array}{c} \\ 2 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc:cc} 2 & 1 & \text{S} & \text{F} \end{array} \\ \begin{pmatrix} -2\beta & 2\mu & 0 & 2\alpha \\ 0 & -\beta & \mu & \alpha \end{pmatrix} \end{array}. \tag{1}$$

4.2 Task-level Replication

For task-level replication, we define the job service state to be $(N_2(t), N_1(t))$, where $N_2(t)$ is the number of subtasks with both replicas in service, and $N_1(t)$ is the number of subtasks with one failed replica at time t. As each job has n subtasks, the set of service phases is N_J=$\{(n_2, n_1)|n_2, n_1 \in \{0,\ldots,n\}, 1\leq n_2+n_1\leq n\}$, and the number of phases is $n(n+3)/2$. The transition rates of the sub-generator matrix S_{ser} are summarized in Table 1. Consider for instance the state $(n_2, n_1)\in N_J$ with $n_2\geq 1$, where either of the two replicas of the n_2 subtasks may encounter a failure, making the service process jump to state (n_2-1, n_1+1) with rate $2n_2\alpha$. Further, since every subtask in the job starts with both replicas in process, the service process starts in state $(n, 0)$, thus $\boldsymbol{\alpha}_{\text{ser}}((n,0))$=1, and all the other entries of $\boldsymbol{\alpha}_{\text{ser}}$ are zero. As an example, consider the case with n=2 phases, which has 5 service phases, N_J=$\{$(2,0), (1,1), (1,0), (0,2), (0,1)$\}$. The matrix $[S_{\text{ser}}|\boldsymbol{t}_S, \boldsymbol{t}_F]$ is thus given by

$$\begin{array}{c} \\ (2,0) \\ (1,1) \\ (1,0) \\ (0,2) \\ (0,1) \end{array} \begin{array}{c} \begin{array}{ccccc:cc} (2,0) & (1,1) & (1,0) & (0,2) & (0,1) & \text{S} & \text{F} \end{array} \\ \begin{pmatrix} -4\beta & 4\alpha & 4\mu & 0 & 0 & 0 & 0 \\ 0 & -3\beta & \mu & 2\alpha & 2\mu & 0 & \alpha \\ 0 & 0 & -2\beta & 0 & 2\alpha & 2\mu & 0 \\ 0 & 0 & 0 & -2\beta & 2\mu & 0 & 2\alpha \\ 0 & 0 & 0 & 0 & -\beta & \mu & \alpha \end{pmatrix} \end{array}. \tag{2}$$

4.3 Job-level Replication

For job-level replication, we track the number of subtasks in service of each of the two job replicas as $(N_2(t), N_1(t))$. To limit the state space, we let $N_2(t)$ be the number of subtasks in service of the job replica with *more* subtasks in service, and let $N_1(t)$ be the number of subtasks in service of the other job replica. The service-phase space is thus N_J=$\{(n_2, n_1)|n_2, n_1\in\{0,\ldots,n\}, n_2\geq n_1\}$, which has cardinality $n(n+3)/2$. In the example above, with n=2, we have 5 service phases: {(2,2), (2,1), (2,0), (1,1), (1,0)}. Notice that when $N_1(t)$ equals 0, one of the job replicas has failed while the other is still in service. As an example, consider the case where the service process is in state $(n_2, n_1)\in N_J$ with n_2>$n_1\geq 1$, and one of the n_1 subtasks of the job with the least number of subtasks in process fails; the service process jumps to state $(n_2, 0)$ with rate $n_1\alpha$. This and the other transition rates for this case are summarized in Table 2. Also, as both job replicas start with all their subtasks in process, all jobs start service in phase (n, n), thus $\boldsymbol{\alpha}_{\text{ser}}((n,n))$=1.

TABLE 1: Transition rates for **task-level** replication

From	To	Rate	Range
(n_2, n_1)	(n_2-1, n_1+1)	$2n_2\alpha$	$n_2\geq 1, n_1\geq 0$
(n_2, n_1)	(n_2-1, n_1)	$2n_2\mu$	$n_2\geq 2, n_1\geq 0$
(n_2, n_1)	(n_2, n_1-1)	$n_1\mu$	$n_2\geq 0, n_1\geq 2$
(n_2, n_1)	(F)	$n_1\alpha$	$n_2\geq 0, n_2\geq 1$
(1,1)	(1,0)	μ	$n_2=1, n_1=1$
(1,1)	(0,1)	2μ	$n_2=1, n_1=1$
(1,0)	(S)	2μ	$n_2=1, n_1=0$
(0,1)	(S)	μ	$n_2=0, n_1=1$

Remark 1. Notice that in all the cases, with or without replication, the S_{ser} matrix has an upper-triangular structure. In the case without replication we simply order the phase space in decreasing order, and see that the only transition allowed from state n_1 is to state n_1-1, resulting in an upper-triangular matrix S_{ser}, as in Eq. (1). For the cases with replication, we obtain a similar structure by ordering the phase space lexicographically in decreasing order. From tables 1 and 2 we observe that all transitions from a phase (n_2, n_1) are to a phase (n_2, n_1-1) or to $(n_2-1, -)$, i.e., either the first or the second variable decrease. Thus, under a decreasing lexicographic order, the matrix S_{ser} is upper triangular. This is exemplified in Eq. (2). This structure will be exploited in the numerical methods proposed in Section 5.

4.4 Introducing Correlation Between Replicas

So far we have assumed that the failures of the replicas of a subtask or job are statistically independent. For instance, in Eq. (2), in state $(2,0)$ each task fails independently with rate α, thus the total rate to jump to state $(1,1)$ is 4α. This is not necessarily true for real applications, where the source of a failure may be located in the task itself, for instance due to its input data. We thus introduce correlation between replicas, which we model by defining the probability p that the failure of one replica causes its partner to fail as well.

4.4.1 Task-level replication

For task-level replication, the time to the first failure among the two subtask replicas is exponentially distributed with rate 2α. To incorporate correlation, we model the time to failure for the remaining replica as the product of two independent random variables, a Bernoulli random variable with mean $1-p$ and an exponential random variable with parameter α, i.e., the probability that the time to failure of the second replica is less than t is $p+(1-p)(1-\exp^{-\alpha t})$,

TABLE 2: Transition rates for **job-level** replication

From	To	Rate	Range
(n_2,n_1)	(n_2,n_1-1)	$n_1\mu$	$n_2>n_1\geq 2$
(n_2,n_1)	(n_2,n_1-1)	$2n_1\mu$	$n_2=n_1\geq 2$
(n_2,n_1)	(n_2-1,n_1)	$n_2\mu$	$n_2>n_1\geq 1$
(n_2,n_1)	$(n_2,0)$	$n_1\alpha$	$n_2>n_1\geq 1$
(n_2,n_1)	$(n_1,0)$	$n_2\alpha$	$n_2>n_1\geq 1$
(n_2,n_1)	$(n_2,0)$	$2n_1\alpha$	$n_2=n_1\geq 1$
$(n_2,0)$	$(n_2-1,0)$	$n_2\mu$	$n_2\geq 2, n_1=0$
$(n_2,0)$	(F)	$n_2\alpha$	$n_2\geq 1, n_1=0$
$(n_2,1)$	(S)	μ	$n_2\geq 2, n_1=1$
(1,1)	(S)	2μ	$n_2=1, n_1=1$
(1,0)	(S)	μ	$n_2=1, n_1=0$

for $0\leq p\leq 1$ and $t\geq 0$. Clearly, the case when p equals 0 is equivalent to the independent case since the second replica fails with exponential rate α. Instead, when p equals 1, task-level replication works exactly as the no-replication case since the failure of either subtask replica leads to the failure of its sibling, resulting in the failure of the whole job. Intermediate cases are covered by values of p between 0 and 1, where a larger value of p decreases the job reliability as it makes more likely the failure of both subtask replicas.

4.4.2 Job-level replication

The introduction of correlation in job-level replication is more involved due to the lack of synchronization between the replicas, since the completion of a *subtask* replica does not cancel its partner. As the subtasks are homogeneous, if a job is in service phase $(n_2,n_1)\in N_J$, for each of the n_2 subtasks in the first job replica, its partner is still in service in the second job replica with probability n_1/n, or it has already completed service with probability $(n-n_1)/n$. Thus, the failure of any of these n_2 subtasks causes the failure of its sibling subtask in the other job replica with probability pn_1/n. As such an event triggers the failure of the whole job, the job jumps from phase (n_2,n_1) to the absorbing phase F with rate $n_2\alpha(pn_1/n)$. On the other hand, the failure of any of the n_2 subtasks does not trigger the failure of its sibling in service with probability $(1-p)n_1/n$. Also, the sibling subtask may have already completed service with probability $(n-n_1)/n$. As a result, the job transits from phase (n_2,n_1) to phase $(n_1,0)$ with rate $n_2\alpha((1-p)n_1/n+(n-n_1)/n)=n_2\alpha(1-pn_1/n)$.

Using a similar argument, the failure of any of the n_1 subtasks in the second job replica leads the job to transit from phase (n_2,n_1) to phase $(n_2,0)$ with rate $n_1\alpha(1-pn_2/n)$, and to the absorbing state F with rate $n_1\alpha(pn_2/n)$. The transition rates for job-level replication with correlation are summarized in Table 3, from which it is clear that the case with $p=0$ is equivalent to the independent case. However, when p equals 1, the behavior of job-level replication is not equivalent to the case without replication as the successful completion of a subtask in one job replica does not cancel its partner in the other job replica, which can still fail and make its job replica fail as well. We study this issue experimentally in Section 6.

4.5 The Multi-node System

Based on the analysis of a single node, the extension to the multi-node case relies on the simple observation that each node operates independently and only interacts with the other nodes via the arrival process. If the job arrival process to the *cluster* is a MAP(m_a, D_0, D_1), the arrival process to a single node can also be described as a MAP. Under round-robin scheduling, the parameters are

$$C_0=\begin{bmatrix} D_0 & D_1 & . & \cdots \\ . & D_0 & D_1 & \cdots \\ \vdots & \vdots & \ddots & \ddots \\ . & . & \cdots & D_0 \end{bmatrix}, \quad C_1=\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ D_1 & 0 & \cdots & 0 \end{bmatrix}. \tag{3}$$

These matrices are of size cm_a, where c is the number of nodes, such that $c-1$ jobs are assigned to the non-tagged nodes, as captured by C_0, between two consecutive arrivals to the tagged node, captured by C_1. Also, under random scheduling, the arrival process has parameters $C_0=D_0+(1-c^{-1})D_1$, $C_1=c^{-1}D_1$.

TABLE 3: Transition rates for **job-level** replication with correlated failures

From	To	Rate	Range
(n_2,n_1)	(n_2,n_1-1)	$n_1\mu$	for $n_2>n_1\geq 2$
(n_2,n_1)	(n_2,n_1-1)	$2n_1\mu$	for $n_2=n_1\geq 2$
(n_2,n_1)	(n_2-1,n_1)	$n_2\mu$	for $n_2>n_1\geq 1$
(n_2,n_1)	$(n_2,0)$	$n_1\alpha(1-n_2p/n)$	for $n_2>n_1\geq 1$
(n_2,n_1)	$(n_1,0)$	$n_2\alpha(1-n_1p/n)$	for $n_2>n_1\geq 1$
(n_2,n_1)	(F)	$2n_1n_2\alpha p/n$	for $n_2\geq n_1\geq 1$
(n_2,n_1)	$(n_2,0)$	$2n_1\alpha(1-n_2p/n)$	for $n_2=n_1\geq 1$
$(n_2,0)$	$(n_2-1,0)$	$n_2\mu$	for $n_2\geq 2, n_1=0$
$(n_2,0)$	(F)	$n_2\alpha$	for $n_2\geq 1, n_1=0$
$(n_2,1)$	(S)	μ	for $n_2\geq 2, n_1=1$
(1,1)	(S)	2μ	for $n_2=1, n_1=1$
(1,0)	(S)	μ	for $n_2=1, n_1=0$

5 Efficient Response-Time Computation

In Section 4 we obtained the PH representation $(\boldsymbol{\alpha}_{\text{ser}}, S_{\text{ser}})$ of the service-time distribution. Assuming we have a PH representation $(\boldsymbol{\alpha}_{\text{wait}}, S_{\text{wait}})$ for the waiting-time distribution, and given the independence between service and waiting times, we can obtain the PH representation $(\boldsymbol{\alpha}_{\text{res}}, S_{\text{res}})$ of the response-time distribution as

$$\boldsymbol{\alpha}_{\text{res}}=[\boldsymbol{\alpha}_{\text{wait}} \quad (1-\boldsymbol{\alpha}_{\text{wait}}\mathbf{1})\boldsymbol{\alpha}_{\text{ser}}], \quad S_{\text{res}}=\begin{bmatrix} S_{\text{wait}} & (-S_{\text{wait}}\mathbf{1})\boldsymbol{\alpha}_{\text{ser}} \\ \mathbf{0} & S_{\text{ser}} \end{bmatrix}.$$

This representation captures that for some jobs the response time is made of a first period of waiting followed by a period of service, while others start service directly. With this representation we can readily compute response-time percentiles and moments. This section therefore focuses on computing the PH representation $(\boldsymbol{\alpha}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, for which we propose a numerically-efficient algorithm that allows us to consider parallel jobs with a large number of subtasks. In fact, this method is applicable for general MAP/APH/1 queues, where APH stands for acyclic PH distributions. APH distributions are characterized by having an upper-triangular generator matrix, which is a key feature of the job service time representations PH$(\boldsymbol{\alpha}_{\text{ser}}, S_{\text{ser}})$ introduced in the previous section.

5.1 The Waiting-time Distribution

The waiting time of a job is the time period between its arrival and the time its service starts. To determine the waiting-time distribution, we follow the approach in [20], observing the queue only during the busy periods, and

defining a bivariate Markov process $\{(X(t), J(t))|t\geq 0\}$. Here the *age* $X(t)$ is the total time-in-system of the job in service, and the *phase* $J(t)=(A(t), N_2(t), N_1(t))$ holds the joint state of the arrival process $A(t)$ and the service process $(N_2(t), N_1(t))$, as defined in Section 4. The phase $J(t)$ thus takes values in a set of size $m=m_a m_s$. Recall that, for job-size n, in the no-replication case $m_s=n$, while under task- and job-level replication $m_s=n(n+3)/2$. Instead, the age $X(t)$ takes values in $[0,\infty)$, increasing linearly with rate 1 if no service completions occur. In case of a service completion, $X(t)$ suffers a downward jump, with its new value being equal to the waiting time of the job starting service. The service completion or failure of the *job* in service triggers the job in front of the queue to start service, while the service completion or failure of a *subtask* does not necessarily trigger the start of a new job service.

To determine the PH representation $(\boldsymbol{\alpha}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, we rely on the stationary distribution $\boldsymbol{\rho}(x)$ of the $(X(t), J(t))$ process, which has a matrix-exponential representation [20] $\boldsymbol{\rho}(x)=\boldsymbol{\rho}(0)\exp(Tx)$, for $x>0$. The $m\times m$ matrix T can be found as the solution to the matrix integral equation

$$T = Q \otimes I_{m_a} + \int_0^\infty \exp(Tt)(\Pi \otimes \exp(D_0 t) D_1) dt, \quad (4)$$

where Q and Π are $m_s \times m_s$ matrices that describe the evolution of the service phase process. The off-diagonal entries of Q hold the transition rates, between the job service phases N_J, not associated with the start of a new job service, while Π holds the transition rates associated with a job service completion or failure, during a busy period. The diagonal entries of Q are set such that $Q+\Pi$ is the generator of the service phase process [20], [21]. From the definition of Q we see that it is in fact equal to S_{ser} since it holds transition rates that are not accompanied by a job service completion or failure. Similarly, $\Pi=-(S_{\text{ser}}\mathbf{1})\boldsymbol{\alpha}_{\text{ser}}$, as this matrix holds the rates with which a job completes service or fails, and a new one starts with phase according to $\boldsymbol{\alpha}_{\text{ser}}$. Notice that, since $\boldsymbol{\alpha}_{\text{ser}}$ has a single nonzero entry, Π has a single nonzero column. Also, as pointed out in Remark 1, the matrix S_{ser}, thus Q, has an upper-triangular structure. We will exploit these features in developing an efficient algorithm to solve Eq. (4) to find T, as described in Section 5.2.

To complete the matrix-exponential representation $\boldsymbol{\rho}(x) = \boldsymbol{\rho}(0)\exp(Tx)$, we need to find the vector $\boldsymbol{\rho}(0)$, which is the stationary distribution of the phase at the beginning of a busy period. This vector can be found as the solution to the equation

$$\boldsymbol{\rho}(0) = \boldsymbol{\rho}(0) \int_0^\infty \exp(Tt)((S_{\text{ser}}\mathbf{1}) \otimes \exp(D_0 t)) dt (-D_0^{-1})(\boldsymbol{\alpha}_{\text{ser}} \otimes D_1). \quad (5)$$

This equation describes how the phase starts a busy period according to $\boldsymbol{\rho}(0)$, evolves with T and at some time t a service completion occurs with rate $S_{\text{ser}}\mathbf{1}$, which terminates the busy period. Next, the process simply keeps track of the arrival phase, which evolves according to D_0, until a new job arrives with rates in D_1, and starts service according to $\boldsymbol{\alpha}_{\text{ser}}$. To solve this equation we define the matrix $P=\int_0^\infty \exp(Tt)(I_{m_s} \otimes \exp(D_0 t))dt$, and integrate by parts to obtain

$$TP + P(I_{m_s} \otimes D_0) = -I_m.$$

This is a Sylvester matrix equation that can be solved in $O(m^3)$ time with the Hessenberg-Schur method [22]. After finding P we can re-write (5) as the linear system

$$\boldsymbol{\rho}(0)=\boldsymbol{\rho}(0)P((S_{\text{ser}}\mathbf{1}) \otimes I_{m_a})(-D_0^{-1})(\boldsymbol{\alpha}_{\text{ser}} \otimes D_1),$$

which we can readily solve to find $\boldsymbol{\rho}(0)$.

Relying on $(\boldsymbol{\rho}(0), T)$ we can obtain the PH representation $(\boldsymbol{\alpha}_{\text{wait}}, S_{\text{wait}})$ of the waiting time distribution as follows. Let $\boldsymbol{\sigma}$ be the steady-state marginal distribution of the phase process $J(t)$, thus [20]

$$\boldsymbol{\sigma} = \int_0^\infty \boldsymbol{\rho}(t) dt = -\boldsymbol{\rho}(0) T^{-1}.$$

Also, let $\boldsymbol{\varphi}$ be the probability that when the current state is $(t+dt, i)$, the next return to a state with age in the interval $[t, t+dt]$ is by way of a downward jump, thus [21]

$$\boldsymbol{\varphi}= \int_0^\infty \exp(Tt)(\Pi \otimes \exp(D_0 t) D_1) dt \mathbf{1} = (T - Q \otimes I_{m_a})\mathbf{1}.$$

The PH representation of the waiting time is given by [21]

$$\boldsymbol{\alpha}_{\text{wait}} = \upsilon \boldsymbol{\sigma} \circ \boldsymbol{\varphi} / ((\boldsymbol{\sigma} \circ \boldsymbol{\varphi})\mathbf{1}), \qquad S_{\text{wait}} = \Lambda^{-1} T' \Lambda,$$

where $\circ$ stands for the Hadamard product [23], $'$ denotes the matrix transpose, Λ is a diagonal matrix such that $\Lambda\mathbf{1}=\boldsymbol{\alpha}'_{\text{wait}}$, and υ is the probability that a job has to wait. For the sake of completeness, we recall the computation of υ, by first defining η_0 to be the number of service completions in a busy period, and η_1 the number of arrivals in a not-busy period, respectively. Their expected values can be obtained as [21, Section 7.2]

$$E[\eta_0] = -\boldsymbol{\rho}(0) T^{-1} (\Pi \otimes I_{m_a})\mathbf{1} / \boldsymbol{\rho}(0)\mathbf{1},$$

and $E[\eta_1]=1$, since an arrival during a not-busy period occupies the whole computing node, initiating a busy period. Thus the probability that a job has to wait is

$$\upsilon = (E[\eta_0]-1)/(E[\eta_0]-1+E[\eta_1]) = 1-1/E[\eta_0],$$

since, in a cycle made of one busy and one not-busy period, $E[\eta_0]-1$ is the expected number of jobs that have to wait, and $E[\eta_0]-1+E[\eta_1]$ is the expected number of job arrivals.

5.2 Computing the Matrix T

The standard method [20] to solve Eq. (4) is iterative. However, different from the numerical integration used in [20], we follow [24] to define the matrix

$$L = \int_0^\infty \exp(Tt)(I_{m_s} \otimes \exp(D_0 t)) dt,$$

such that Eq. (4) can be written as

$$T = Q \otimes I_{m_a} + L(\Pi \otimes D_1). \quad (6)$$

Integrating L by parts we obtain the equation

$$TL + L(I_{m_s} \otimes D_0) = -I. \quad (7)$$

Thus, we compute the T matrix iteratively starting with $T_0=Q \otimes I_{m_a}$, and in the k^{th} iteration solving

$$T_k L_k + L_k (I_{m_s} \otimes D_0) = -I \quad (8)$$

to find L_k. The iteration continues with $T_{k+1}=Q \otimes I_{m_a} + L_k(\Pi \otimes D_1)$ until convergence is reached. Eq. (8) is a Sylvester matrix equation that can be solved in $O(m^3)$ time with the Hessenberg-Schur algorithm [22].

TABLE 4: Computation times (sec) for jobs with 50 subtasks

Arrivals (SCV)	U	Full	BS	BW	Ratio	
					R-BS	R-BW
HE_2 (SCV=2)	0.2	3583.6	91.8	1.7	39.0	2108.0
	0.5	7310.9	188.6	3.4	38.8	2150.3
	0.9	34634.0	878.0	15.8	39.4	2192.0
HE_2 (SCV=10)	0.2	4675.6	114.1	2.1	41.0	2226.5
	0.5	13041.3	325.5	6.0	40.1	2173.6
	0.9	/	915.2	15.9	/	/
MAP_2 (SCV=10) (decay = 0.1)	0.2	4613.5	121.3	2.0	38.0	2306.8
	0.5	13488.8	342.5	5.8	39.4	2325.7
	0.9	/	942.8	15.2	/	/
MAP_2 (SCV=10) (decay = 0.9)	0.2	5174.8	120.8	2.3	42.8	2249.9
	0.5	32208.6	742.8	13.2	43.36	2440.0
	0.9	/	3853.5	64.2	/	/

TABLE 5: Computation times (sec) for jobs with 100 subtasks

Arrivals (SCV)	U	BS	BW	Ratio
HE_2 (SCV=2)	0.2	5130.8	17.0	301.8
	0.5	8735.6	35.1	248.9
	0.9	/	156.8	/
HE_2 (SCV=10)	0.2	5219.7	20.3	257.1
	0.5	15237.1	56.3	334.2
	0.9	/	184.2	/
MAP_2 (SCV=10) (decay = 0.1)	0.2	5372.1	21.4	251.0
	0.5	18817.3	57.7	326.1
	0.9	/	177.9	/
MAP_2 (SCV=10) (decay = 0.9)	0.2	5696.3	22.4	254.3
	0.5	/	134.6	/
	0.9	/	595.2	/

5.3 Exploiting Restricted Transitions

As the time complexity to solve Eq. (8) is cubic in m, and m increases quadratically with the number of subtasks n, finding T becomes computationally expensive when considering instances with large job size n. We therefore introduce an efficient solution method to exploit the inner structure of the matrices Q and Π.

We consider a two-step approach. In the first step we exploit the structure in $\Pi{=}-(S_{\text{ser}}\mathbf{1})\boldsymbol{\alpha}_{\text{ser}}$, which has a single non-zero column since $\boldsymbol{\alpha}_{\text{ser}}$ has a single non-zero entry. This in turn implies that the term $L(\Pi{\otimes}D_1)$ in Eq. (6) has only $r{=}m_a$ non-zero columns. As a result, the iteration $T_{k+1}{=}Q\otimes I_{m_a}{+}L_k(\Pi\otimes D_1)$ only affects the first r columns of T. We can then proceed as in Appendix A to re-state Eq. (7) as the Sylvester matrix equation (10) where the unknown matrix Y is of size $m{\times}r$ rather than $m{\times}m$ as in Eq. (7). Thus, using the Hessenberg-Schur method [22], we can solve Eq. (10) in $O(5m^3/3)$ time, where this leading term is due to the Hessenberg decomposition of T. Since solving the original Eq. (7) requires $O(115m^3/6)$ time, this step can be expected to provide a significant reduction in computation time, especially for large block-size m.

In the second step we exploit the upper-triangular structure in $Q{=}S_{\text{ser}}$. The key idea is to replace the Hessenberg decomposition required to solve Eq. (10) by a more efficient method that solves a triangular system of size m and applies a rank-r correction by means of the Woodbury identity. The solution of such system requires $O(rm^2)$ time, and this is done to find each of the r columns of X in Eq. (11), thus taking $O(r^2m^2)$ time to solve Eq. (11), which is equivalent to Eq. (7), as detailed in Appendix B. This is a significant improvement compared to the standard method, which has a time complexity cubic in m, especially since r is n^2 times smaller than m, and n is the job size. The next section reports experiments that illustrate how significant the gains obtained with this method are.

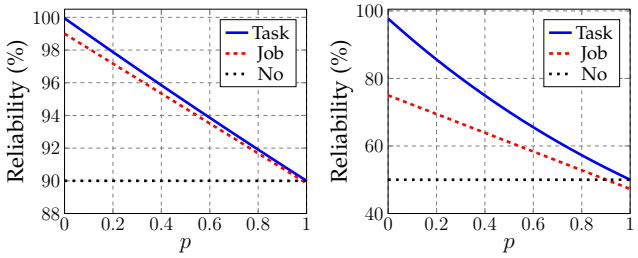
5.4 Evaluation

To illustrate the efficiency of the proposed approach to compute the matrix T we compare it against the solution with the standard method described in Section 5.2, which we label **Full**. As the proposed method relies on the two steps described in Appendices A and B, we present the results considering only the method in Appendix A, labeled **BS**, and the overall method, labeled **BW**.

The experiments were performed in MATLAB on an Intel Core i7-3770 machine, running at 3.4 GHz and with 16 GB of memory. We consider jobs with 50 and 100 subtasks, service rate $\mu{=}1$, and failure rate $\alpha{=}0.1$. For the arrival process, we consider renewal processes with two-phase Erlang (ER_2) and hyper-exponential (HE_2) inter-arrival distributions, and general order-2 MAPs (MAP). We thus cover a broad range of behaviors in terms of variability, measured by the squared coefficient of variation (SCV), defined as $C_X^2{=}\text{Var}(X)/E^2[X]$ for a random variable X. The ER_2 and HE_2 distributions represent the SCV$=0.5$ and SCV≥ 1 cases, respectively, while the MAP considers auto-correlated arrivals. The parameters of the HE_2 distribution are computed using the moment-matching method in [25], while the matrices D_0 and D_1 of the MAP are obtained with the method in [26]. We consider MAPs with decay rate of the auto-correlation function equal to 0.1 and 0.9, and SCV equal to 10. With these parameters, the size of the phase space is $m{=}2650$ for 50 subtasks and $m{=}10300$ for 100 subtasks, while the number of nonzero columns in A_0 is $r{=}2$ for both cases. We focus on task-level replication as the results are similar for the job-level replication case.

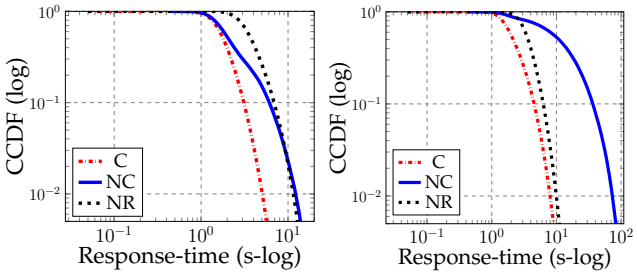
Table 4 summarizes the computation times to find the matrix T for the case of 50 subtasks. The rows in Table 4 consider different arrival processes and utilization levels U, as shown in the first two columns. We set a time limit of 10 hours, and the results not shown here correspond to cases where the computation times exceeded this limit. We observe that the **BS** method reduces the computation times by two orders of magnitude compared with the **Full** method, while the **BW** method offers even larger reductions, of three orders of magnitude compared to the **Full** method. This can also be observed in the last two columns (Ratio), where we report the ratio between the **Full** and the **BS** and **BW** methods. Specifically, the computation times with the **BW** method are 2100-2500 times shorter than with the **Full** method, while the **BS** method offers a reduction of only 35-45 times.

Table 5 shows similar results for the 100-subtask case, but the results for the **Full** method are not reported since its computation times exceeded 10 hours in all instances. We show in the last column the ratio of the computation times between the **BS** and the **BW** methods, where we observe that **BW** offers 240-340 shorter execution times than the **BS** method. Here and in the previous set of results we observe that the computation times of all the methods increase when the utilization, the arrival process variability and its auto-correlation increase. In the toughest case considered here, where the utilization is 0.9 and the arrival process SCV



(a) NR-reliability: 90% (b) NR-reliability: 50%

Fig. 4: Reliability under different replication modes



(a) NR-utilization:0.1 (b) NR-utilization:0.3

Fig. 5: The effect of canceling - NR-reliability:90%

and decay rate of the auto-correlation function are 10 and 0.9, respectively, only the **BW** method is able to provide a result, and it does so in just under 10 minutes. In most of the other instances, its computation times are under 1 minute, highlighting its ability to efficiently compute T by exploiting the structure in the matrices Π and Q.

Remark 2. Notice that the solution method proposed here is not restricted to the analysis of parallel jobs considered in this paper. For instance, using the results in [27], any acyclic PH distribution (APH) can be transformed to have an entry vector $\boldsymbol{\tau}$ with all the mass in the first phase, and an upper-triangular matrix S. Thus, any MAP/APH/1 queue will fit within our assumptions regarding the structure of the Q and Π matrices.

6 Evaluating the Impact of Replication

With the proposed model, and the efficient solution method introduced in the previous section, we are now in a position to evaluate the effect of replication on the reliability and response times offered by the parallel-job computing cluster. In particular, we are able to determine how replication impacts these metrics under different system conditions, such as the system utilization, the correlation across replicas, and the job arrival process.

In the following we make use of the no-replication (NR) case as the baseline. Thus we consider scenarios where we fix the reliability for the NR case (NR-reliability) and its utilization (NR-utilization). The mean service rate μ is set to 1, while the failure rate α is set to achieve a target NR-reliability. Similarly, the arrival rate is set to achieve a target NR-utilization. To make the comparison fair, the NR setup has as many servers as the setup with replication, which implies that, for job-size n, for every computing node with $2n$ servers in the scenarios with replication, there are two computing nodes in the NR case, each with n servers.

6.1 The Impact of Replication on Reliability

The reliability is measured as the probability P_S that a job completes service successfully. When the replicas fail independently, the reliability depends only on the job size n, the service rate μ, and the failure rate α. In the no-replication case, it is given by $(\mu/(\mu+\alpha))^n$, while under task-level replication it is equal to $(1-(\alpha/(\alpha+\mu))^2)^n$, and for job-level replication it is $1-(1-(\mu/(\alpha+\mu))^n)^2$. Instead, when the failures between replicas are correlated, the reliability is

$$P_S = -\boldsymbol{\alpha}_{\text{ser}} S_{\text{ser}}^{-1} \boldsymbol{s}_{\text{ser}},$$

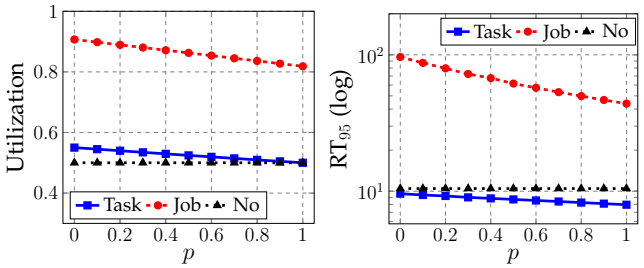
where $(\boldsymbol{\alpha}_{\text{ser}}, S_{\text{ser}})$ is the PH representation of the job service-time distribution, defined in Section 4.4, and $\boldsymbol{s}_{\text{ser}} = -S_{\text{ser}}\mathbf{1}$.

Figure 4(a) compares the job reliability under different replication modes, considering the case where the job-size n is 20, the NR-reliability is 90%, and the correlation p varies from 0 to 1. We observe that both task- and job-level replication improve the job-reliability under most values of p, with task-level having a slightly higher reliability. Also, the improvement in reliability decreases as the correlation p increases. Under task-level replication, the reliability decreases until it becomes equal to the case without replication when the correlation reaches 1. For instance, task-level replication improves the job reliability from 90% to 99.94% when p=0, but decreases to 94.85% when p=0.5, and drops back to 90% for p=1. This effect is more clear in Figure 4(b), where the NR-reliability is decreased to 50%. Here the reliability achieved with task-level replication is significantly larger for small to medium correlations, and it is always better than no-replication, except under perfect correlation. However, we observe that the reliability achieved under job-level replication is much lower than the one achieved under task-level replication. For instance, when p is 0.1, the reliability achieved under task-level replication is 91.46%, while under job-level replication it is just 72.22%. This is because under job-level replication, a single subtask failure causes the failure of a whole job-replica, while in task-level replication the job fails only if *both* replicas of a subtask fail.

Figure 4 also offers a less expected, and somewhat counter-intuitive, result. While job-level replication clearly improves the reliability under a broad range of the correlation, it actually performs *worse* than the no-replication case for correlations close to one. The cause of this behavior lies in the lack of synchronization between the two job replicas. While in task-level replication the successful completion of a task immediately cancels its sibling, in job-level replication this synchronization does not occur. As a result, the sibling task continues in execution and may eventually fail, causing the failure of its job replica. This situation is exacerbated when the failure rate and correlation are high, degrading the job reliability until it becomes worse to replicate at the job level than to not replicate. This effect however only arises when the correlation is close to one, which is not to be expected in application-level failures.

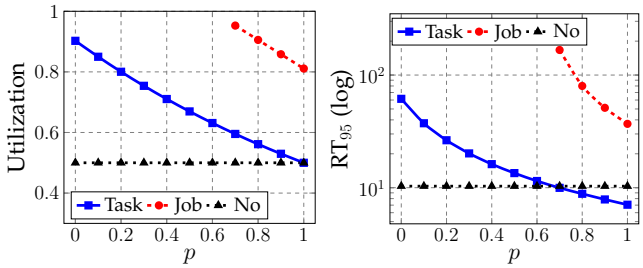
6.2 Comparing against Replication Without Canceling

To evaluate the effectiveness of canceling, we compare the response times obtained with and without replication,



(a) Utilization (b) RT_{95}

Fig. 6: The effect of the correlation - NR-reliability: 90%



(a) Utilization (b) RT_{95}

Fig. 7: The effect of the correlation - NR-reliability: 50%

considering uncorrelated failures (p=0) and HE_2 arrivals (SCV=2) as an example. Here we use the same notation for the arrival process (Exp, ER_2, HE_2 and MAP) introduced in Section 5.4. Figure 5 depicts the response-time complementary CDF (CCDF) for the system without replication (NR), with replication with canceling (C), and without canceling (NC). For brevity we consider task-level replication only. The results without canceling are obtained with a simulation model as this is not the focus of this paper. Here we consider an instance with job-size 20 and NR-reliability 90%. In Figure 5(a), where the NR-utilization is just 0.1, it can be seen that the introduction of replication may reduce the response-time tails compared to the no-replication case. This reduction is caused by the possibility of selecting the first replica that completes service. In addition, the introduction of canceling reduces the system load, significantly reducing the job response times, especially their tail. Further, increasing the NR-utilization to 0.3, which guarantees that the system with replication without canceling is stable, as shown in Figure 5(b), task-level replication with canceling is able to achieve a shorter response-time tail than without replication, while the system without canceling offers a much longer tail. The effect of canceling is thus more evident in this scenario as the baseline load is higher and the introduction of replication without canceling introduces enough additional load to cause large response times.

6.3 The Impact of Replication on the Response Times

We now look further into the impact of replication on the offered response times, and how this effect depends on the system parameters. In particular, we focus on the response-time percentiles, where the i^{th} percentile of the response-time distribution, RT_i, stands for the value x such that i% of the jobs experience a response time of at most x, for i=1, . . . , 99. We consider a case with job-size 20, HE_2 arrivals (SCV=2), NR-utilization of 0.5, and NR-reliability of 90%. Figures 6(a) and 6(b) show how the utilization and RT_{95}, respectively, change with an increasing failure correlation p. Clearly, both task- and job-level replication increase the system utilization, but job-level replication adds a much larger load, increasing the utilization from 0.5 to between 0.91 and 0.82. The effect of this additional load is clearly visible in Figure 6(b), which shows how the RT_{95} shoots up under job-level replication, being one order of magnitude larger than under no-replication and task-level replication.

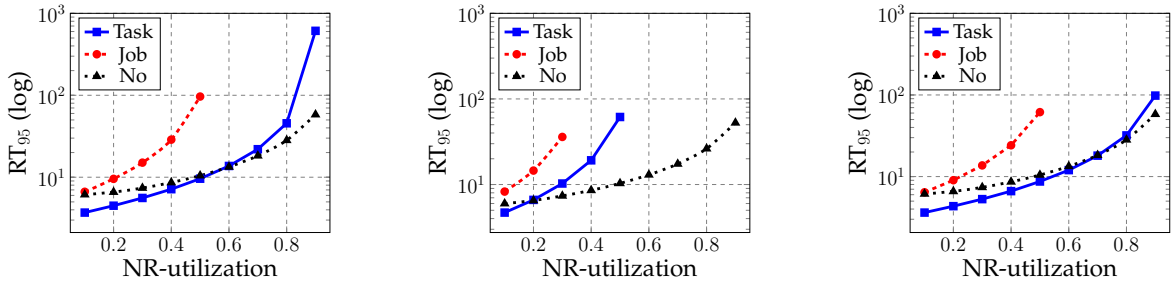
We now decrease the NR-reliability from 90% to 50%, and display the obtained utilization and RT_{95} in Figures 7(a) and 7(b). In this case both task- and job-level replication increase the utilization compared to the no-replication case. In fact, when the correlation p is below 0.7, the additional load under job-level replication makes the system unstable. Recall that the NR-utilization is 0.5 in this case. Moreover, Figure 7(b) shows that the RT_{95} achieved under task-level replication is larger than without replication when the correlation p is below 0.7. This clearly differs from the previous case where task-level replication improved the RT_{95} for every correlation level. This is because of two reasons:

- If the NR-reliability is low and no replication is used, only short jobs can complete service successfully before encountering a failure. This causes the response times under no-replication to be necessarily short.
- Introducing replicas has two effects: first, it increases the resource utilization, thus increasing the response times; second, it allows the selection of the first replica that finishes, potentially reducing the response times. If the NR-reliability is low, as in Figure 7, one of the replicas will likely fail and the first effect will be dominant, increasing the response times. Instead, if the NR-reliability is high, as in Figure 6, then both replicas are likely to finish service, allowing the selection of the first result.

Moreover, we notice that both the utilization and the RT_{95} decrease as the correlation p increases. Here again, as the correlation p increases it is more likely that the failure of a subtask leads to the failure of its replica, allowing only short jobs to complete service. Also, the larger correlation reduces the system load, allowing for shorter response times. In Figure 7(b) we also observe that, under large correlation values, even though the improvement in reliability is limited, task-level replication is still able to reduce the response times, thanks to its ability to select the first task-replica that completes.

6.3.1 *The Effect of the Baseline Utilization*

As replication, even with canceling, increases the system load, it is natural to expect that its benefits can only be exploited as long as the NR-utilization is not too high. We now explore some scenarios to study how high can the NR-utilization be for replication with canceling to effectively reduce the response times. Figure 8(a) compares the RT_{95} achieved under different NR-utilization levels, for the case with job-size 20, NR-reliability of 90%, HE_2 arrivals



(a) NR-reliability: 90%; p=0 (b) NR-reliability: 50%; p=0 (c) NR-reliability: 90%; p=0.5

Fig. 8: RT_{95} achieved with different NR-utilization

(SCV=2), and uncorrelated failures (p=0). Clearly, job-level replication achieves the highest RT_{95}, increasing the response times compared to no-replication, and causing the system to become unstable when the NR-utilization is larger than 0.5. Instead, task-level replication reduces the RT_{95} compared to no-replication as long as the NR-utilization is 0.6 or less. For larger values of the NR-utilization, task-level replication increases the RT_{95}, until the point that, under a 0.9 NR-utilization, the RT_{95} is actually one order of magnitude larger than without replication. It is still significant, though, that task-level replication causes just a small increase in the RT_{95} for an NR-utilization as high as 0.7, while offering a much better reliability (99.95%). If the NR-reliability decreases to 50%, as depicted in Figure 8(b), task-level replication is able to reduce the RT_{95}, but only for an NR-utilization of up to 0.2. For NR-utilizations between 0.3 and 0.5, task-level replication offers a larger RT_{95} than without replication, and higher values will make the system unstable. This is again the result of the low NR-reliability.

We now go back to the case with NR-reliability 90%, as in Figure 8(a), but introduce a failure correlation of p=0.5, and depict the results in Figure 8(c). Comparing Figures 8(a) and 8(c), we observe that task-level replication reduces the RT_{95} up to a higher NR-utilization (0.7) in the correlated case than in the uncorrelated case (0.6). This is because a larger correlation implies a higher probability that the failure of one replica causes its partner to fail, leading to a lower utilization under task-level replication, and therefore shorter response times. We thus observe that the impact of replication not only is affected by the NR-utilization, but by the NR-reliability and the correlation among the task failures. The results indicate that under low to medium NR-utilizations, task-level replication is able to improve the response times, as long as the NR-reliability is high. A low NR-reliability scenario can benefit from replication in terms of reliability, but the room for improving the response times is more limited.

7 SLO-driven Resource Provisioning

As we have seen, while task-level replication is effective in improving the reliability, its impact on the response times offered depends on a number of factors. This is particularly relevant for a system that aims to fulfill a service-level objective (SLO) $RT_{\max}$ defined as a bound on a response-time percentile. Thus, the questions we want to answer in this section are *whether replication should be adopted or not*, and *what is the minimum number of computing nodes c^* necessary to achieve a given SLO RT_{max}*. Recall that, in the case without replication n servers are needed to process a job of size n, while task-level replication requires $2n$. As defined in Section 3.2, in the case without replication, each computing node is able to process one parallel job of size n. Under task-level replication, instead, two computing nodes are needed to process each job, as a total of $2n$ servers are needed in this case. Also, we assume that a maximum of $nc_{\max}$ servers can be used, thus making up at most $c_{\max}$ computing nodes. Round-robin or random scheduling is used to allocate the incoming jobs to the computing nodes. Relying on the proposed analytical model, this resource provisioning problem can be formulated as:

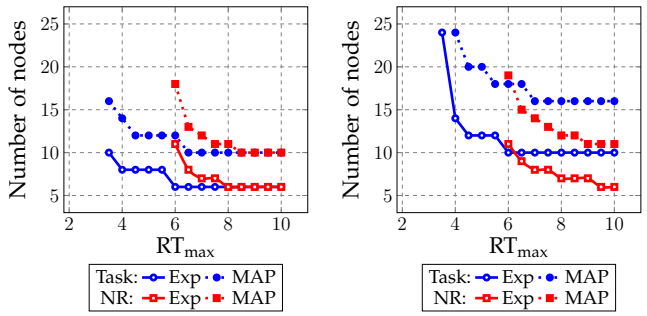
$$\begin{aligned} c^* = \text{Min.} \quad & c, \\ \text{s.t.} \quad & RT_i(c, R_{\text{choice}}) \leq RT_{max}, \\ & \varrho(c, R_{\text{choice}}) < 1, \\ & c \in \{1, 2, \ldots, c_{max}\}, \\ & R_{\text{choice}} \in \{0, 1\}, \end{aligned} \tag{9}$$

where the binary variable R_{choice} is equal to one if task-replication is adopted, or to zero if no-replication is implemented. Also, $RT_i(c, R_{\text{choice}})$ is the i^{th} percentile of the response-time distribution obtained with c computing nodes and R_{choice} replication mode. To solve this problem, we consider two separate problems, one for each value of R_{choice}. Since, for each of these problems, the percentile $RT_i(c, R_{\text{choice}})$ is a non-increasing function in c, which is the objective function, Algorithm 1 shows how each of these problems can be solved with a binary search algorithm [28] restricted to the integers $\{1, 2, \ldots, c_{max}\}$. We then choose the replication strategy that requires the least number of computing nodes. In case of a draw, we always choose to replicate to achieve a higher reliability.

We first consider an instance with round-robin scheduling, exponential (Exp) inter-arrival times, mean arrival rate of 1.2, job-size 20, NR-reliability 90%, and a total of c_{max}=25 nodes, i.e., 500 servers in total. As depicted in Figure 9(a), we evaluate different RT_{max} objectives in the set $[2, 2.5, \ldots, 10]$. The missing results correspond to cases where the required RT_{max} cannot be achieved with the given maximum of 25 nodes. The first obvious trend is that the minimum number of nodes required decreases as the SLO RT_{max} increases. More interestingly, we observe a significant difference between no-replication and task-level replication. In fact, tight RT_{max} objectives can only be achieved with replication.

Algorithm 1 Computing the optimal number of servers

Require: $c_{\max}$, R_{choice}, $RT_{\max}$, i
if $RT_i(c_{\max}, R_{\text{choice}}) > RT_{\max}$ **then**
  return The required $RT_{\max}$ cannot be achieved.
else
  $c_{\min} = 1$
  while $c_{\max} - c_{\min} > 1$ **do**
    $c = \lfloor (c_{\min} + c_{\max})/2 \rfloor$
    if $RT_i(c, R_{\text{choice}}) > RT_{\max}$ **then**
      $c_{\min} = c$
    else
      $c_{\max} = c$
    end if
  end while
end if
return c



(a) NR-reliability: 90% (b) NR-reliability: 50%

Fig. 9: Minimum number of computing nodes needed to achieve a response-time SLO under different replications modes and arrival processes

For instance, the system with task-level replication can achieve an RT_{max} as low as 3.5, while the system without replication cannot achieve an RT_{max} below 6.0. In addition, under the tightest achievable SLO by all replication modes, RT_{max}=6.0, task-level replication requires 12 computing nodes, while no-replication needs 18 nodes, a 33% more. As expected, these differences decrease as the objective RT_{max} becomes less strict.

Figure 9(a) also depicts the minimum number of nodes required under MAP arrivals, assuming an SCV of 10 and a decay rate of the auto-correlation function of 0.9. Either with or without replication, many more nodes are required to handle the bursty workload introduced by MAP arrivals, compared to the numbers needed under Exp arrivals, to achieve the same response-time SLO. For instance, under the tightest achievable SLO with task-level replication, RT_{max}=3.5, 10 computing nodes are required to handle exponential IATs, while 16 nodes are needed to cope with the variability and correlation in the workload introduced by the MAP arrivals, a 60% more.

In Figure 9(b) we decrease the NR-reliability to 50%. In contrast with the results in Figure 9(a), we observe that a loose response-time SLO can be achieved without replication with fewer computing nodes than with task-level replication. In this specific case, this occurs for SLOs larger than 6.5. This is different from Figure 9(a), where task-level replication always requires less or the same number of computing nodes than without replication. This difference is due to the low NR-reliability (50%) considered in Figure 9(b), as in this case only short jobs can complete service in the system without replication. The benefits of replication in this case are thus limited to the improvement in reliability. However, if a tight SLO is considered, in this case RT_{max} below 6, only task-level replication is able to achieve it.

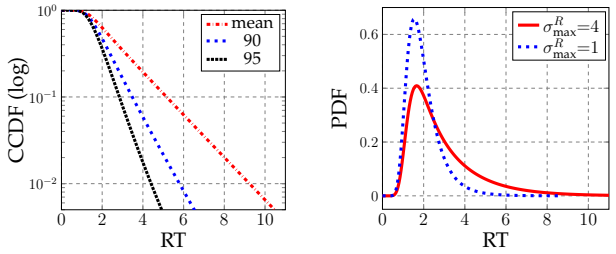
TABLE 6: Solution times for the optimization problem (sec)

Arrival	Mode	Achievable?	Mean	Std	Min	Max
Exp	NR	No	0.30	0.20	0.19	0.76
Exp	Task	No	5.13	0.09	5.01	5.53
Exp	NR	Yes	16.42	0.18	16.22	18.83
Exp	Task	Yes	11.79	0.76	11.36	14.74
MAP	NR	Yes	7.90	2.95	6.21	18.70
MAP	Task	Yes	229.34	14.68	195.77	274.92

Table 6 summarizes the computation times required to solve the optimization problem on the machine described in Section 5.4, solving each scenario in Figure 9(a) 50 times. In the cases where the $\text{RT}_{\max}$ objective is not achievable, the computation times are much lower as the algorithm only solves the model once to check if the $\text{RT}_{\max}$ limit is achievable. The first two rows in Table 6 illustrate this, as these times are much lower than the ones in rows 3-4, where the optimization method actually performs the binary search. Here we also observe that the mean computation times under task-level replication and Exp arrivals is lower than under no-replication. This is because in the no-replication mode, the maximum number of nodes available is twice as much as for task-level mode, thus requiring more iterations when applying the binary-search algorithm. Rows 5-6 show the total computation times under MAP arrivals. Different from the scenarios with Exp arrivals, the computation times under task-level replication are much larger than under no-replication, which is caused by the longer model evaluation time required at each iteration. Thanks to the efficient evaluation proposed for the model and the quadratic convergence of the binary-search method, the resource allocation method requires fairly short times to find the optimal number of resources and whether replication is to be adopted or not.

7.1 Managing the Response-Time Distribution

The provisioning problem (9) allows us to determine the minimum number of computing nodes needed to achieve a certain response-time percentile i. This feature can therefore be used to manage the response-time distribution, i.e., it is possible to define SLOs for different percentiles of the distribution, which can result from a set of requirements specified by the system users. In addition, the first constraint in (9) can refer to other measures on the response-time distribution, such as the mean or the standard deviation. To illustrate this we consider a similar case as in the previous section, with MAP arrivals, and we solve (9) setting three different response-time SLOs: mean response-time of 4.0, RT_{90} of 4.0, and RT_{95} of 3.5. Under these three SLOs, the minimum number of nodes required are 10, 12 and 16, respectively. Figure 10(a) compares the response-time CCDF obtained in each of these cases. While each of the SLOs is attained in each of the three cases, we see how the overall response-time distribution changes, particularly reducing the tail as the SLO is set on the 90^{th} percentile instead of the mean, and further when it is set on the 95^{th} percentile.



(a) Response-time CCDF (b) Response-time PDF

Fig. 10: Response-time CCDF and PDF achieved under different response-time SLOs

IAT	p-value	KS
Exp	$8.3217E{-}10$	0.6519
HE_2	$2.7839E{-}5$	0.4690
HE_3	0.0623	0.2612
HE_4	0.0623	0.2612
HE_5	0.0623	0.2612
HE_6	0.0623	0.2612

TABLE 7: KS test

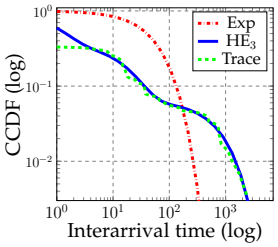


Fig. 11: IATs CCDF

While the interest in the response-time percentiles is based on offering an upper bound on the response time faced by a large proportion of the jobs, other response-time metrics are also relevant. In particular, the standard deviation σ^R provides a measure of the variability of the response times around their mean. Thus, when provisioning resources it may be relevant to ensure that the standard deviation of the response times is bounded by a limit $\sigma^R_{\max}$. To show the impact of using σ^R to size the cluster, in Figure 10(b) we compare the response-time density function (PDF) obtained by setting an SLO $\sigma^R_{\max}$ of 4 and 1 on the standard deviation, while keeping an SLO on the mean response time of 1 for both cases. The minimum number of nodes needed are 10 and 14, leading to a σ^R of 1.76 and 0.82, respectively. The difference is clearly shown in Figure 10(b), where the response times obtained with 10 nodes are spread out over a larger range of values than when 14 nodes are used. The response times offered by the cluster in the second case can thus be expected to be more consistent than in the first scenario. The resource provisioning scheme is therefore able to consider a number of system conditions, such as NR-reliability and NR-utilization, and how they affect the achievement of response times that comply with different SLOs, defined as percentiles or distributional moments.

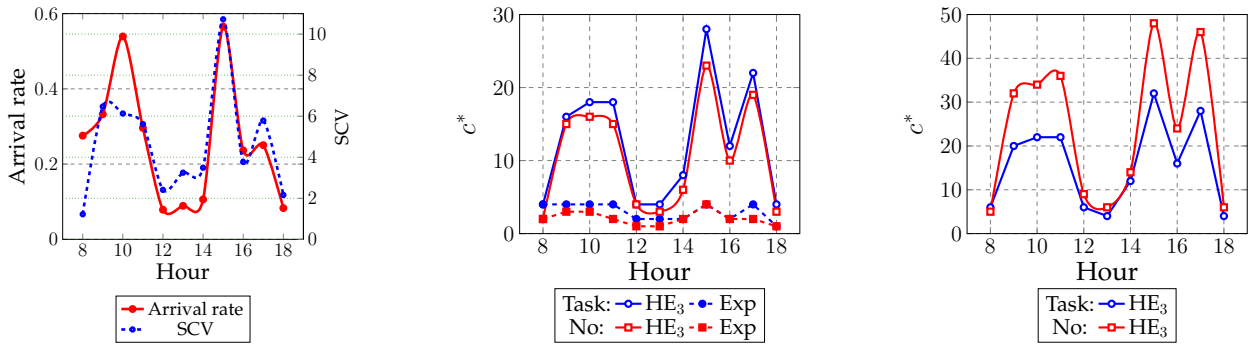
8 Case-study: the RICC logs

In this section, we evaluate the proposed model and the resource provisioning strategy by considering realistic traffic patterns from a parallel cluster. We make use of logs from the RIKEN Integrated Cluster of Clusters (RICC), available on the Parallel Workloads Archive [9]. We parameterize the service process of our model with single-task jobs that completed service successfully, estimating a subtask service rate μ of 0.3245 tasks per hour. We also estimate a failure probability of 3.63%, based on which we obtain a failure rate α of 0.0122 tasks per hour. We observe 112 classes of jobs with job sizes ranging from 1 to 8192. Among multi-task jobs, those with job-size 32 are the most common, accounting for 30.3% of all multi-task jobs. As job arrivals show a strong daily and hourly cycle, we divide the arrival data into sets according to the day of the week and the hour of the day. As an example, we consider the samples from 11:00 to 12:00 on Wednesday, where the inter-arrival times (IATs) have a squared coefficient of variation (SCV) of 4.69. We use the method in [29], as implemented in jPhase [30], to find a hyper-exponential distribution with r phases (HE_r). We test the goodness-of-fit of the fitted CDFs with the Kolmogorov-Smirnov (KS) test [31], and show in Table 7 the p-value and the maximum absolute difference between the CDFs for the exponential (Exp) and various HE_r distributions. The Exp and HE_2 assumptions fail the Kolmogorov-Smirnov test, while the HE_r with 3 or more passes the test under typical significance levels. Since the improvement in the p-value and the KS statistic is limited for 4 or more phases, we choose the HE_3 representation, and compare its CCDF with that of the real trace and the fitted Exp in Figure 11. Clearly, the HE_3 distribution captures the characteristics of the trace much better than the exponential, especially its long tail along a wide range of values.

To further emphasize the importance of considering the variability of the IATs in sizing the cluster, we use the provisioning method in Section 7 to determine the number of computing nodes to achieve an RT_{95} of 10 hours. We consider the RICC logs for the period from 07:00 to 18:00 on Tuesdays, using both HE_3 and Exp distributions to represent the IATs for each hour. In Figure 12(a) we show the observed arrival rate and SCV, and in Figure 12(b) the minimum number of nodes needed to satisfy an RT_{95} SLO of 10 hours, under both no-replication and task-level replication. The first clear observation is the huge difference in the number of computing nodes needed between HE_3 and Exp arrivals. The exponential assumption requires significantly fewer nodes to satisfy the response-time SLO, as more nodes are needed to handle the variability in the IATs introduced by the HE_3 arrivals. As a result, if the exponential assumption is used to dimension the system, we may expect a large violation of the response-time SLO, as the resources are not enough to cope with the arrival process variability.

Now, focusing on the HE_3 case, the results show that the effect of the arrival rate is dominant as the trend of the minimum number of computing nodes is similar to the trend of the arrival rate. For instance, the largest number of servers is required during the hour 15, which has the highest arrival rate. However, we observe that the SCV has a very significant effect too. For example, comparing hours 8 and 11 under HE_3 arrivals and task-level replication, the arrival rate slightly increases by 7.44% from hour 8 to hour 11, but the minimum number of nodes needed increases dramatically from 4 to 18. This is caused by the large SCV during hour 11, which is 5.61, while that of hour 8 is only 1.22. If we look at the provisioning under the exponential assumption, the number of nodes needed for both hours 8 and 11 is 4, as the IAT variability is completely ignored.

In Figure 12(b) we observe that task-level replication requires more computing nodes than no-replication. However, if we tighten the response-time SLO from 10 to 7.1, as in Figure 12(c), task-level replication requires fewer nodes



(a) Arrival rate and SCV (b) Min number c^* (RT$_{max}$=10) (c) Min number c^* (RT$_{max}$=7.1)

Fig. 12: Time dependent resource provisioning

than no-replication. Also, task-level replication increases the reliability from 54.8% to 98.9%. Thus, on top of the reliability gains, task-level replication has the potential to reduce the resource requirements under tight response-time SLOs.

9 Conclusion

In this paper we evaluate the ability of concurrent replication with canceling to improve the reliability and response times of parallel jobs subject to failures. We propose a stochastic model to obtain the response-time distribution, for which we derive an efficient solution method to consider jobs with a large number of subtasks. We demonstrate that task-level replication with canceling has the potential to reduce latency under a low to medium utilization, as long as the task reliability is high. Scenarios with a low task reliability can benefit from replication in terms of reliability, but the room for improving the response times is more limited. When the task failures are correlated, the improvement in reliability is more limited, but task-level replication is still able to reduce the response times. Building on the proposed model, we develop a resource-provisioning strategy that determines whether replication should be adopted or not, and provides the minimum number of computing nodes needed to comply with an SLO on the response times. The results highlight that ignoring the statistical characteristics of the arrival process may lead to large SLO violations. We also observe that, in addition to the gains in reliability, task-level replication has the potential to reduce the resource requirements for tight response-time SLOs. While the focus in this paper has been on parallel jobs that are processed synchronously, future work will consider the case of asynchronous parallel jobs, where we expect the benefits of replication to be stronger, as the additional load on the system should decrease due to the finer granularity with which resources become available.

Appendix A
Exploiting the Structure in Π

As the term $L(\Pi \otimes D_1)$ in (6) has only $r{=}m_a$ non-zero columns, the iteration $T_{k+1}{=}Q \otimes I_{m_a}{+}L_k(\Pi \otimes D_1)$ only affects the first r columns of T, the last $m-r$ columns being given by the corresponding columns of $T_0{=}Q \otimes I_{m_a}$. To formalize this observation, we write $\Pi \otimes I_{m_a}{=}\Gamma K$, where the $m{\times}r$ matrix Γ and the $r{\times}m$ matrix K are given by

$$\Gamma = -S_{\text{ser}}\mathbf{1} \otimes I_r, \text{ and } K = \begin{bmatrix} I_r & 0_{r\times m-r} \end{bmatrix}.$$

Letting $Y{=}L\Gamma$, we can rewrite Eq. (6) as

$$T{=}Q \otimes I_{m_a}{+}L\Gamma K(I \otimes D_1){=}Q \otimes I_{m_a}{+}Y \begin{bmatrix} D_1 & 0_{r\times m-r} \end{bmatrix}.$$

Therefore, it is enough to compute the $m{\times}r$ matrix Y, instead of finding the $m{\times}m$ matrix L in each iteration. Focusing on the first r columns of this equation, we write

$$\tilde{T} = \boldsymbol{Q}_1 \otimes I_r + YD_1,$$

where $\tilde{T}$ refers to the first r columns of the matrix T, and $\boldsymbol{Q}_1$ is the first column of Q. To find Y we solve

$$TY + YD_0 = -\Gamma, \tag{10}$$

which is obtained by post-multiplying (7) by Γ.

Appendix B
Exploiting the Structure in Q

To exploit the upper-triangular structure in Q, we recall that the Hessenburg-Schur method to solve Eq. (10) requires finding the Schur decomposition of D_0, i.e., a quasi-upper-triangular matrix S and an orthogonal matrix V such that $D_0{=}VSV'$. Thus, assuming this is the k^{th} iteration, and post-multiplying Eq. (10) by V, we have

$$T_kYV + YVV'D_0V = -\Gamma V.$$

Letting $X{=}YV$, $F{=}{-}\Gamma V$, and replacing T_k by its value from the previous iteration, we obtain

$$\left(Q \otimes I_{m_a} + L_{k-1}(\Pi \otimes D_1)\right)X + XS = F. \tag{11}$$

Following [22], the above equation can be solved by columns, by writing a linear system that involves finding one or two columns at a time, thanks to the quasi-upper-triangular nature of S. For the sake of clarity, we assume that S is upper-triangular, such that each step involves finding a single column. The more general case of a quasi-upper-triangular S can be treated similarly.

Using the upper-triangular structure in S we can find the p^{th} column of X, $\boldsymbol{X}_p$, by solving the system

$$\left(Q \otimes I_{m_a} + L_{k-1}(\Pi \otimes D_1) + S_{p,p}I_m\right)\boldsymbol{X}_p = \boldsymbol{F}_p, \tag{12}$$

where $\hat{\boldsymbol{F}}_p \equiv F_p - \sum_{i=1}^{p-1} \boldsymbol{X}_i S_{i,p}$. Notice that the right-hand side depends on the first $p-1$ columns of X, thus we can start with the first column and proceed forward finding one column at a time. This is a linear system of size m and becomes the bottleneck of the Hessenberg-Schur method, in which a Hessenberg decomposition of the matrix $(Q \otimes I_{m_a} + L_{k-1}(\Pi \otimes D_1))$ is used to find $\boldsymbol{X}_p$. Instead, we propose to take advantage of the structure in Q. As this matrix is upper-triangular, the matrix $R = Q \otimes I_{m_a} + I_m S_{p,p}$ is upper-triangular as well. Moreover, we know from the previous section that the matrix $L_{k-1}(\Pi \otimes D_1)$ can be written as $L_{k-1}\Gamma K(I \otimes D_1) = Y_{k-1} D_1 K$. The left-hand side matrix in (12) can then be written as $R + Y_{k-1} D_1 K$, and since Y_{k-1} is an $m \times r$ matrix and $D_1 K$ is an $r \times m$ matrix, this expression is a rank-r correction of the upper-triangular matrix R. We can thus use the Woodbury matrix identity [32] to write the inverse of the matrix in (12) as

$$(R + L_{k-1}(\Pi \otimes D_1))^{-1} = R^{-1} - R^{-1} Y_{k-1} (I_r + D_1 K R^{-1} Y_{k-1})^{-1} D_1 K R^{-1}.$$

Further, post-multiplying by $\hat{\boldsymbol{F}}_p$ and recalling that $K = [I_r \; 0_{r \times m-r}]$, we can write

$$\begin{aligned} \boldsymbol{X}_p &= (R + Y K (I_{m_s} \otimes D_1))^{-1} \hat{\boldsymbol{F}}_p \\ &= R^{-1} \hat{\boldsymbol{F}}_p - R^{-1} Y_{k-1} (I_r + D_1 \tilde{R}_Y)^{-1} D_1 \tilde{R}_F, \end{aligned}$$

where $\tilde{R}_Y$ holds the first r rows of the $m \times r$ matrix $R^{-1} Y_{k-1}$, and $\tilde{R}_F$ the first r rows of the vector $R^{-1} \hat{\boldsymbol{F}}_p$.

As a result, the key step in finding the column $\boldsymbol{X}_p$ is the solution of the upper-triangular system $RZ = [\hat{\boldsymbol{F}}_p \; Y_{k-1}]$. With the Woodbury identity, this can be done in $O(rm^2 + r^2 m + rm + r^3)$ time due to the triangular structure of R.

REFERENCES

[1] J. Dean and L. A. Barroso, "The tail at scale," *CACM*, vol. 56, pp. 74–80, 2013.

[2] L. T. X. Phan, Z. Zhang, Q. Zheng, B. T. Loo, and I. Lee, "An empirical analysis of scheduling techniques for real-time cloud-based data processing," in *IEEE SOCA*, 2011.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Why let resources idle? aggressive cloning of jobs with Dolly," *Memory*, vol. 40, p. 80, 2012.

[4] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *Allerton*, 2013.

[5] A. Vulimiri, P. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *CoNEXT*, 2013.

[6] B. Snyder, "Server virtualization has stalled, despite the hype," http://www.infoworld.com/print/146901, 2010.

[7] Z. Qiu and J. F. Pérez, "Assessing the impact of concurrent replication with canceling in parallel jobs," in *IEEE MASCOTS*, 2014.

[8] R. Sturm, W. Morris, and M. Jander, *Foundations of Service Level Management*. Sams publishing, 2000.

[9] "Parallel workloads archive," http://www.cs.huji.ac.il/labs/parallel/workload/, 2015.

[10] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *HotNets*, 2012.

[11] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Morgan Kaufmann, 2010.

[12] V. Stantchev and M. Malek, "Addressing web service performance by replication at the operating system level," in *IEEE ICIW*, 2008.

[13] V. Stantchev, "Effects of replication on web service performance in websphere," ICSI, Tech. Rep., 2008.

[14] P. G. Harrison and Z. Qiu, "Performance enhancement by means of task replication," in *EPEW*, 2013.

[15] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *IEEE Allerton*, 2012.

[16] P. Harrison and S. Zertal, "Queueing models of RAID systems with maxima of waiting times," *Perform. Eval.*, vol. 64, pp. 664–689, 2007.

[17] A. S. Lebrecht and W. J. Knottenbelt, "Response time approximations in fork-join queues," in *UKPEW*, 2007.

[18] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. SIAM, 1999.

[19] M. Neuts, "A versatile Markovian point process," *JAPROB*, vol. 16, pp. 764–779, 1979.

[20] B. Sengupta, "Markov processes whose steady state distribution is matrix-exponential with an application to the GI/PH/1 queue," *AAP*, vol. 21, pp. 159–180, 1989.

[21] S. Asmussen and J. R. Møller, "Calculation of the steady state waiting time distribution in GI/PH/c and MAP/PH/c queues," *Queueing Syst.*, vol. 37, pp. 9–29, 2001.

[22] G. Golub, S. Nash, and C. Van Loan, "A Hessenberg-Schur method for the problem AX+ XB= C," *IEEE TACON*, vol. 24, 1979.

[23] R. A. Horn, "The Hadamard product," in *Proc. Symp. Appl. Math*, vol. 40, 1990, pp. 87–169.

[24] Q. He, "Analysis of a continuous time SM[K]/PH[K]/1/FCFS queue: Age process, sojourn times, and queue lengths," *JSSC*, vol. 25, pp. 133–155, 2012.

[25] W. Whitt, "Approximating a point process by a renewal process, I : Two basic methods," *Oper. Res.*, vol. 30, pp. 125–147, 1982.

[26] A. Heindl, "Inverse characterization of hyperexponential MAP(2)S," in *ASMTA*, 2004.

[27] A. Cumani, "On the canonical representation of homogeneous markov processes modelling failure-time distributions," *MiRe*, vol. 22, pp. 583–602, 1982.

[28] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to algorithms*. MIT press, 2001.

[29] R. El Abdouni Khayari, R. Sadre, and B. R. Haverkort, "Fitting world-wide web request traces with the EM-algorithm," *Perform. Eval.*, vol. 52, pp. 175–191, 2003.

[30] J. F. Pérez and G. Riano, "jPhase: an object-oriented tool for modeling phase-type distributions," in *ACM SMCtools*, 2006.

[31] F. J. Massey Jr, "The Kolmogorov-Smirnov test for goodness of fit," *JASA*, vol. 46, pp. 68–78, 1951.

[32] W. W. Hager, "Updating the inverse of a matrix," *SIAM review*, vol. 31, pp. 221–239, 1989.

Zhan Qiu is a PhD student in computer science at Imperial College London. She received a MSc in Computer Science from Imperial College London in Oct 2012. Her research interests include performance and reliability modeling and evaluation of computer and communication systems, parallel processing, performance optimization and resource provisioning. She is a member of the IEEE, and the IEEE Communications Society.

Juan F. Pérez is a Research Fellow at the University of Melbourne, Department of Mathematics and Statistics. He obtained a PhD in Computer Science from the University of Antwerp, Belgium, in 2010, and was a Research Associate in performance analysis at Imperial College London, Department of Computing. His research interests center around the performance analysis of computer systems, especially on cloud and cluster computing and optical networking.