Assessing the impact of concurrent replication with canceling in parallel jobs

Zhan Qiu, Juan F. Pérez
Department of Computing
Imperial College London
London, UK
{zhan.qiu11, j.perez-bernal}@imperial.ac.uk

***Abstract*—Parallel job processing has become a key feature of many software applications, e.g., in scientific computing. Parallelization allows these applications to exploit large resource pools, such as cloud or grid data centers. However, a job composed of a large number of parallel tasks will suffer a failure if any of its tasks fail, requiring reprocessing and additional delays. In this paper, we explore the effect that the replication of parallel jobs has on the job reliability and response time, as well as on resource utilization. The replication mechanism consists of concurrently processing replicas, at either the job or the task level, retrieving the results of the replica that finishes first, if any, and canceling any remaining replica in process. We propose a stochastic model that explicitly considers parallel job processing, replication at both the job and the task level, and handles general arrival processes. We develop a numerically-efficient algorithm to solve large-scale instances of the model and compute key performance metrics. We observe that the task cancellation mechanism offers an effective way of limiting the increase in resource utilization, allowing the use of replicas that not only increase the job reliability, but have the potential to reduce the response times.**

I. Introduction

In the past decade, parallel computing has become a pervasive technology that finds applications in a large number of software systems. In particular, scientific computing applications heavily rely on parallelization to process large amounts of data and to perform a large number of operations, exploiting grid computing resources. Furthermore, many applications now run parallel subtasks to take advantage of large resource pools, as those available in private and public data centers, to offer data-intensive services, e.g. business analytics. Although each subtask in a parallel job may be highly reliable, the job reliability decreases with the number of tasks, and a single subtask failure will make the whole job fail [1].

To overcome this, a common approach is to retry the execution of a failed subtask, typically after a timeout handled by a central scheduler. While this approach improves the job reliability, it generates additional delays that degrade the quality of service (QoS) offered by the application, a major issue for latency-sensitive applications [2]. An alternative is to process the replicas *concurrently*, improving the job reliability and limiting the latency. In fact, this approach has been considered [3]–[6] to reduce the latency of highly parallel applications, issuing replicas to diverse resources, and using the result from whichever replica responds first. Heavy concurrent replication is also appealing in the light of the low utilization common in data centers, for example, traces released by Facebook reveal median CPU and memory utilization under 20% [3]. However, the execution of concurrent replicas may increase the resource utilization, and the response times, beyond desirable levels. An approach to limit this effect is to cancel any outstanding replicas, once the first one completes [5], [7]. In this paper, we study how this approach, concurrent replication with canceling, affects both the reliability and latency of parallel jobs, and the interplay between these two performance measures.

The *contributions* of this paper are two-fold. First, we propose a stochastic model to analyze the performance of the concurrent replication approach with replica cancellation. The model employs Markovian Arrival Processes (MAPs) to represent arrival processes with general inter-arrival times and correlation. We devise a numerical solution, building on previous work by [8], that is able to tackle large instances where traditional methods, including [8], fail or require long computation times. Second, we explore how the proposed concurrent replication with canceling impacts the reliability and response times of parallel jobs. In particular, we find that this fault-tolerance approach, compared with replication *without* canceling, significantly reduces the resource utilization and delivers a shorter mean response time, making fault-tolerance via replication more applicable. The two replication levels considered in this paper, to clone at the granularity of jobs (job-level replication) or subtasks (task-level replication), are shown to improve the system reliability. However, task-level replication achieves a much higher reliability while introducing a lower load and delivering shorter response times. Furthermore, task-level replication has the potential to reduce the response times since the task effective execution time becomes the minimum of the concurrently-executing replicas.

This paper is organized as follows. Section II reviews recent related works related, while Section III introduces the reference model and the replication strategy. The job-service-time model and results for the single node case under Poisson arrivals are presented in Section IV. Section V introduces the general model with multiple nodes under MAP arrivals, and its efficient numerical solution. The fault-tolerance method proposed is evaluated in Section VI using a trace from a real parallel cluster. Section VII concludes the paper.

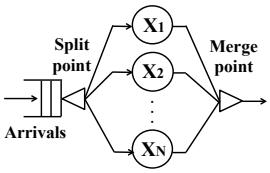


Fig. 1: Split-merge queue

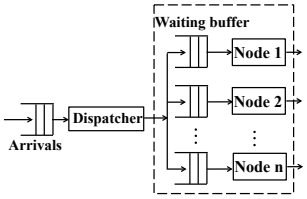
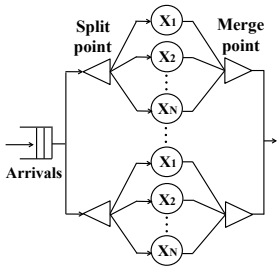
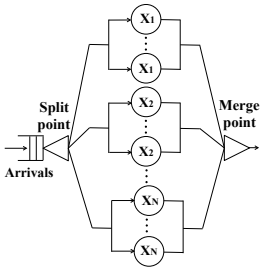


Fig. 2: Reference Model



(a) Job-level Replication



(b) Task-level Replication

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

II. Related Work

In parallel jobs, a subtask may fail during service due to various reasons, such as an error in the input data, the detection of a present or potential deadlock, communication failures [9], timeouts of resources with limited availability, or outputs failing to meet time constraints. Due to the scale and complexity of large-scale parallel systems, it is infeasible to eliminate all the possible failures [6]. A traditional fault-tolerance mechanism to handle request failures is to re-execute a request after failure, typically after a timeout [6], [10]. Although effective to handle failures, this approach introduces additional delays that degrade the QoS offered by the application. This effect is particularly critical for latency-sensitive application, for which all subtasks must complete within a strict deadline, in order for the application to feel responsive [5]. Several fault-tolerance approaches [1], [11], [12] have been proposed for processor failures, adopting redundancy in space and time to improve system reliability. Instead, the focus of this paper is on the analysis of systems subject to request failures. Concurrent replication has been studied recently [3]–[6] to improve latency and reliability. 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. [6] demonstrates replication to be an effective way to achieve robustness under unpredictable failures while limiting latency. To curb the additional load created by concurrent replication, [5] proposes to issue multiple job replicas, collect the results from the replica that responds first, and immediately cancel any outstanding replicas in process. This approach, referred to as replication with canceling, was analyzed for the case of single-task jobs in [7]. In this paper we propose an analytical model to evaluate this replication approach for parallel jobs consisting of multiple subtasks.

To model the concurrent execution of replicas for parallel jobs we make use of split-merge queues [13]. In this class of queues, as shown in Figure 1, an incoming job arriving at the split point is split into multiple subtasks if there is no previous job in the queue and the processors are idle, otherwise, the job will join the end of the waiting queue. Each subtask arrives simultaneously at one of the parallel processors to receive service. The processors are all blocked until all the subtasks finish service, rejoin and depart from the system. The analysis of split-merge queues has been focus on the computation of moments of maximum service time and of distribution of response time. For example, [14] determines analytically the maximum service time in a split-merge queue with random exponential service times, and presents an approximation for the case with general heterogeneous service times. [13] derived the response time distribution in a split-merge queue with general service times and Poisson arrivals, by means of order statistics. These results however do not consider job or task failures, which are the focus of this paper. Split-merge systems have been widely used to model systems where synchronization is central, such as the processing of logical I/O requests in a queueing model of RAID (Redundant Array of Independent Disks) systems [14], and in composite web service [15].

III. Background

We consider parallel systems comprised of multiple distributed computing nodes, and a central scheduler, as shown in Figure 2. Each node is composed of multiple execution units, or processors, and each node executes a parallel job by processing each of the job's subtasks in one of its processors. When a subtask fails during service, and no replication is adopted, the whole job fails. We assume that the processor is not affected by the failure, and continues serving a subtask of the next job. In order to increase the reliability of a system, we propose to *concurrently* process job replicas, and consider two levels of replication granularity.

A. Replication granularity

We consider the two replication granularity levels proposed in [3] for parallel jobs. One option is job-level replication, where for every job submitted to the cluster, multiple replicas are spawned, as shown in Figure 3(a). Once the result from the earliest job-replica is obtained, other remaining outstanding replicas are canceled immediately. Job-level replication is appealing due to its simplicity, but a job-replica is successful only if all its subtasks are successful. The alternative is to clone at subtask level, shown in Figure 3(b), where every subtask is cloned and the result of its earliest replica is returned, canceling any outstanding replicas in process. In this case it is enough that a single replica of each subtask succeeds to have a successful job completion.

B. Reference model

We consider a central dispatcher, as depicted in Figure 2, that assigns jobs to the processing nodes in a Round-Robin (RR) fashion. With the RR algorithm, the jobs are distributed

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

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

evenly among the nodes, and no communication is required between the nodes and the dispatcher about the nodes' state. A job joins the end of the queue in front of the corresponding node, if it is busy, and waits until all the jobs in front are processed. When a job starts service, each of its n subtasks is assigned to one of the node's processors. We assume all jobs are composed of n subtasks, and refer to this number as the *job size*. The subtasks processing times are assumed to be exponentially distributed with rate μ, while the time to failure of a single subtask follows an exponential distribution with parameter α. Notice that, although the subtask service and failure times are exponential, the *job* service and failure times no longer follow an exponential distribution. Consider a system with c nodes and cn processors in total, such that it can process c different jobs, each of size n, in parallel. If one replica is adopted for each job/subtask, then every two nodes, i.e. $2n$ processors, can be grouped together to serve a job and its replica in parallel, allowing the processing of $c/2$ concurrent jobs. In the next section we consider the single-node ($c = 1$) case under Poisson arrivals, and its generalization to $c > 1$ nodes and Markovian arrivals is treated in Section V.

IV. Single Node under Poisson Arrivals

In this section, we consider the case of a single processing node under Poisson arrivals. We model systems without replicas and with both task-level and job-level replications, considering a single replica only. A key step in the analysis of this and the next section is showing that the *job* service and failure times follow a Phase-Type (PH) distribution.

A. Phase-type distributions

A PH distribution [16] is the distribution of the absorption time X in a Markov Chain (MC) where the states $\{1, 2, \ldots, n\}$ are transient and state 0 is absorbing. We denote it as PH($\boldsymbol{\tau}, T$), where $\boldsymbol{\tau}$ is the $1 \times n$ vector of the initial probability distribution, T is the $n \times n$ sub-generator matrix, and the vector $\boldsymbol{t} = -T\boldsymbol{e}$ holds the absorption rates, where $\boldsymbol{e}$ is a column vector with ones. Its cumulative distribution function (CDF) is $F(x) = 1 - \boldsymbol{\tau} exp(Tx)\boldsymbol{e}$, for $x \geq 0$, and the expected absorption time is

$$E[X] = -\boldsymbol{\tau} T^{-1} \boldsymbol{e}. \tag{1}$$

In a single processing node, the job service time is the interval between the time that all its subtasks enter the node's processors, and the time it leaves the node, either with service completion or failure. In the next sections we show that, under the assumption of exponential service and failure times for the subtasks, the *job* service time follows a PH distribution. We consider two absorbing states, *S* and *F*, representing the cases where the job completes service successfully or encounters a failure, respectively. These absorption states have absorption vectors $\boldsymbol{t_S}$ and $\boldsymbol{t_F}$, respectively, such that $\boldsymbol{t} = \boldsymbol{t_S} + \boldsymbol{t_F}$.

B. No-replica model

In the no-replica case, if one subtask fails, the whole job fails. Let X_t be the number of subtasks in service at time t, and let n be the job size. The service time then follows a PH distribution with transient states, or phases, $\{1, 2, \ldots, n\}$. If a subtask of a job in phase n_1 completes service, it leads the job to phase $n_1 - 1$ with rate $n_1\mu$, for $n_1 \geq 1$. If $n_1 = 1$, the subtask service completion leads the job to phase S. Instead, if a subtask of a job in phase n_1 encounters a failure, the service process jumps to phase F with rate $n_1\alpha$. When a job completes service or fails, the next job is injected into the node's processors with initial service phase n.

C. Task-level replication

For task-level replication, we define the service state to be (X_t, Y_t), where X_t is the number of subtasks with both replicas still in service, and Y_t is the number of subtasks with one failed replica at time t. For a job with n subtasks, we have $n(n+3)/2$ transient states in total. The transition rates of the sub-generator matrix T are shown in Table I. For instance, from state (n_2, n_1), $n_2 \geq 1$, $n_1 \geq 0$ either one 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 $2n_2\alpha$. As an example, for a system with job size two, the generator matrix $Q = [T|\boldsymbol{t_S}, \boldsymbol{t_F}]$ of the service process is

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

where $\beta = \alpha + \mu$. Notice the upper-triangular structure of the matrix T, which results from ordering the state space lexicographically. This structure will be exploited in the numerical method proposed in Section V.

D. Job-level replication

For job-level replication, we track the number of subtasks in service for both the original job and its replica as (X_t, Y_t), at time t. To limit the state space, X_t is always the number of subtasks of the job-replica with more subtasks in service, resulting in $n(n+3)/2$ phases for n subtasks. Notice that when Y_t equals 0 either the job or its replica has failed while the other is still in service. For instance, if the service process is in state (n_2, n_1), $n_2 > n_1 \geq 1$, and one of the subtasks of

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

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	$n_2, 0$	$n_1\alpha$	for $n_2 > n_1 \geq 1$
n_2, n_1	$n_1, 0$	$n_2\alpha$	for $n_2 > n_1 \geq 1$
n_2, n_1	$n_2, 0$	$2n_1\alpha$	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 $n_2 = 1, n_1 = 0$

the job with the least number of subtasks in process fails, the service process jumps to state $(n_2, 0)$ with transition rate $n_1\alpha$. The transition rates for this case are summarized in Table II.

E. Performance measures

For Poisson arrivals we analyze the single-node system as an M/G/1 queue with FCFS scheduling [17]. Let X denote the effective service time of an arbitrary job, either failed or successful, and $E[X]$ its expected value, obtained as in Equation (1). Denoting $B^*(S)$ the service time Laplace transform, the waiting time Laplace transform $W_q^*(S)$ [18] is thus

$$W_q^*(S) = \frac{s(1-\rho)}{s - \lambda + \lambda B^*(S)},$$

where λ is the job arrival rate, and the utilization $\rho = \lambda E[X]$.

The probability P_S that a job completes service successfully is given by $P_S = \boldsymbol{\tau}(-T)^{-1}\boldsymbol{t_S}$, where the term $(-T)^{-1}\boldsymbol{t_S}$ captures the service process transitions and the eventual successful completion. The probability of failure P_F can be obtained similarly. The mean waiting time $E[W_{qi}]$ and mean response time $E[R_i]$ for successful and failed jobs ($i \in \{S, F\}$) are given by

$$E[W_{qi}] = -\frac{dW_q^*(s)}{ds}|_{s=0},$$
$$E[R_i] = E[W_{qi}] + 1/\mu_i,$$

where the service rates are given by $\mu_S = \boldsymbol{\tau}(-T)^2\boldsymbol{t_S}/P_S$ and $\mu_F = \boldsymbol{\tau}(-T)^2\boldsymbol{t_F}/P_F$, respectively.

V. Markovian Arrivals and Multiple Nodes

In this section we consider more general arrival processes and multiple computing nodes. We first focus on the single-node case where the arrivals follow a Markovian arrival process (MAP), and then extend this model to the multi-node case under RR scheduling. MAPs were introduced [19] as a generalization of PH distributions, to represent correlated point processes with PH inter-event distributions. Queueing systems with PH or MAP components give rise to generator matrices with a block structure. The model presented in this section falls in this category, specifically as a Quasi-Birth-and-Death (QBD) process [16]. In the next sections, we provide an introduction to QBDs, and model the single-node with MAP arrivals as a QBD. Next we introduce a method to solve large instances of the model, exploiting the inner block structure.

A. Quasi-Birth-and-Death processes

A continuous-time QBD process [16] is a two-dimensional MC $\{(N_t, J_t), t \geq 0\}$. N_t is the *level* variable, which takes values on $\mathbb{N}$, and J_t is the *phase* variable, which takes values on the set $\{1, 2, \ldots, m_0\}$ for level 0, and on $\{1, 2, \ldots, m\}$ for other levels. The one-step transitions are independent of the level n for $n > 1$, and restricted to states in level n', where $n' = n-1, n, n+1$. The QBD infinitesimal generator has the form

$$Q = \begin{bmatrix} B_1 & B_0 & 0 & 0 & \cdots \\ B_2 & A_1 & A_2 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & \cdots \\ 0 & 0 & A_0 & A_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

where the matrices A_0 and A_2 are non-negative and carry the transition rates from level i to level $i-1$ and $i+1$, for $i > 0$ and $i > 1$, respectively. The matrices A_1 and B_1 have non-negative off-diagonal elements, and strictly negative diagonals, holding the transition rates within level i, for $i > 0$ and $i = 0$, respectively. The matrices B_0 and B_2 are non-negative, and hold the transition rates from level 0 to level 1, and from level 1 to level 0, respectively. In the single-node system, the level variable N_t is the number of jobs in the node, waiting and in process, at time t, and the phase variable J_t carries the state of the service and the arrival processes.

B. Markovian Arrival Processes

The continuous-time MAP [16] is a marked Markov process with generator $D = D_0 + D_1$, where the elements of D_0, resp. D_1, represent transitions without, resp. with, arrivals. D_1 is a non-negative matrix, and D_0 has non-negative off-diagonal entries and strictly negative diagonals, with $(D_0 + D_1)\boldsymbol{e} = \boldsymbol{0}$. The arrival rate is given by

$$\lambda = \boldsymbol{\gamma} D_1 \boldsymbol{e}, \tag{2}$$

where $\boldsymbol{\gamma}$ is the stationary distribution of the underlying MC, i.e. $\boldsymbol{\gamma} D = 0$ and $\boldsymbol{\gamma e} = 1$. A MAP can represent a renewal process with PH$(\boldsymbol{\sigma}, S)$ inter-arrival times by setting $D_0 = S$ and $D_1 = \boldsymbol{s\sigma}$, where $\boldsymbol{s}$ is the exit vector $\boldsymbol{s} = -S\boldsymbol{e}$.

The single processing node with MAP arrivals can be modeled as a QBD with blocks

$$\begin{aligned} B_1 &= D_0, & B_0 &= \boldsymbol{\tau} \otimes D_1, \\ B_2 &= \boldsymbol{t} \otimes I_{m_a}, & A_0 &= \boldsymbol{t\tau} \otimes I_{m_a}, \\ A_1 &= T \otimes I_{m_a} + I_{m_s} \otimes D_0, & A_2 &= I_{m_s} \otimes D_1, \end{aligned} \tag{3}$$

where $\otimes$ stands for the Kronecker product [20], and $(\boldsymbol{\tau}, T)$ are the parameters of the service PH distribution, as described in Section IV. The *blocks size*, i.e. the size of the phase space, is $m = m_a m_s$, where m_a and m_s are the number of phases in the arrival and service processes, respectively.

C. Computing the stationary distribution

Let $\boldsymbol{\pi}$ be the steady state distribution of the MC with generator Q, which we partition conformally with Q, i.e., $\boldsymbol{\pi} = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \dots)$. The stationary distribution, if it exists, can be found as $\boldsymbol{\pi_i} = \boldsymbol{\pi_i} R$, where R can be obtained from $R = A_2(-A_1 - A_2 G)^{-1}$, and G is the minimal non-negative solution of the matrix equation [16]

$$A_0 + A_1 G + A_2 G^2 = 0. \tag{4}$$

Many iterative algorithms have been developed to solve this equation, including Functional Iterations (FI), Logarithmic Reduction, and Cyclic Reduction (CR) [16], [21]. However, if the phase space is large, the solution of these equations becomes computationally expensive, as each iteration requires $O(m^3)$ time. In the next section we introduce a solution method that exploits the inner structure of the blocks A_0, A_1, and A_2, to reduce the computation and memory requirements, allowing the solution of large-scale models.

D. Exploiting restricted transitions

The solution method we propose to solve the QBD model with blocks described in Equation (3) relies on three observations. First, a job always starts service in the first phase of the service process, such that the initial probability vector is $\boldsymbol{\tau} = [1, 0, \cdots, 0]$, thus the downward matrix A_0 has only $r = m_a$ nonzero columns. Second, the service matrix T has an upper-triangular structure, which causes the matrix A_1 to have a block-upper-triangular structure, where each block is of size r. Third, the independence of the arrival and service processes implies that the matrix A_2 has a block-diagonal structure, where each block is of size r and equal to D_1. Based on these observations, we build upon the solution method proposed in [8] to exploit the inner structure of the QBD blocks. The work in [8] exploits the structure of matrix A_0 (r nonzero columns) by partitioning the phase space into two sets, of size r and $m - r$ respectively. While the reduction in computation time [8] is significant, it still has to operate on matrices of size $m - r$, which can be very large. Our proposal consists of exploiting not only the structure of A_0, but also that of A_1 and A_2, by partitioning the phase space into m/r sets of size r. As a result, the method operates on small matrices of size r, significantly reducing the time and memory requirements.

We partition the phase space as $S = \{S^+, S^1, S^2, \cdots, S^w\}$, where $w = (m - r)/r$, and write the QBD blocks as

$$A_0 = \begin{bmatrix} A_0^{++} & 0 \\ A_0^{1+} & 0 \\ A_0^{2+} & 0 \\ \vdots & 0 \\ A_0^{w+} & 0 \end{bmatrix}, A_i = \begin{bmatrix} A_i^{++} & A_i^{+1} & A_i^{+2} & \cdots & A_i^{+w} \\ 0 & A_i^{11} & A_i^{12} & \cdots & A_i^{1w} \\ 0 & 0 & A_i^{22} & \cdots & A_i^{2w} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_i^{ww} \end{bmatrix}, \tag{5}$$

for $i = 1, 2$, where each sub-block is an $r \times r$ square matrix. Notice that, although in our problem, the matrix A_2 in Equation (3) is block triangular, we consider the more general block upper-triangular case, as this is sufficient to obtain the most significant gains in computation times. Further, from [8] we know that, due to the structure of A_0, the matrix G has the structure

$$G = \begin{bmatrix} G_+ & 0 \\ G_1 & 0 \\ \vdots & \vdots \\ G_w & 0 \end{bmatrix}, \tag{6}$$

where G_+ and G_p ($p = 1, \cdots, w$) are $r \times r$ matrices. In the next section we show how to compute G_+.

1) Computing G_+: As in [8], we define a new process by observing the QBD process only when the phase variable J_t is in S^+. Since A_0 has only r nonzero columns, any downward transition triggers the phase to a state in S^+. Therefore the new process can only move one level down, but several levels up, in each transition. Hence, the new process is of the M/G/1 type [22], which can be seen as a generalization of a QBD MC, where the level is allowed to skip several levels in the upward direction, but to decrease only to its adjacent level in a single transition. The generator matrix $\bar{Q}$ of the new process is of the form

$$\bar{Q} = \begin{bmatrix} \bar{B}_0 & \bar{B}_1 & \bar{B}_2 & \bar{B}_3 & \cdots \\ \bar{A}_0 & \bar{A}_1 & \bar{A}_2 & \bar{A}_3 & \cdots \\ & \bar{A}_0 & \bar{A}_1 & \bar{A}_2 & \cdots \\ 0 & & & \ddots & \ddots \end{bmatrix}.$$

The stationary distribution of this process can be found using Ramaswami's formula [23], which depends on the matrix $\bar{G}$, i.e. the minimal non-negative solution of the matrix equation

$$\bar{G} = \sum_{k=0}^{\infty} \bar{A}_k \bar{G}^k. \tag{7}$$

The matrix G_+ is equal to $\bar{G}$, since both hold, in their (i, j)th entry, the probability that the original QBD MC visits level n for the first time, starting from state $(n + 1, i)$, by visiting state (n, j), with $i, j \in S^+$.

To specify the blocks of the M/G/1-type MC, let the (i, j)-th entry of the $r \times r$ matrix $K_{l,p}$ hold the probability that, given that the original process starts in state (n, i), with $i \in S^p$, $p \in \{1, 2, \cdots, w\}$, its first transition to a state with phase in S^+ is to the state $(n + l, j)$, for $j \in S^+$, $n > 1$ and $l \in \{-1, 0, 1, \cdots\}$. The matrices $(K_{l,p})_{l \geq -1}$ are given by

$$K_{-1,p} = (-A_1^{pp})^{-1} \left(A_0^{p+} + \sum_{q=p+1}^{w} A_1^{pq} K_{-1,q} \right),$$

$$K_{0,p} = (-A_1^{pp})^{-1} \left(A_1^{p+} + \sum_{q=p}^{w} A_2^{pq} K_{-1,q} + \sum_{q=p+1}^{w} A_1^{pq} K_{0,q} \right),$$

$$K_{1,p} = (-A_1^{pp})^{-1} \left(A_2^{p+} + \sum_{q=p}^{w} A_2^{pq} K_{0,q} + \sum_{q=p+1}^{w} A_1^{pq} K_{1,q} \right),$$

$$K_{l,p} = (-A_1^{pp})^{-1} \left(\sum_{q=p}^{w} A_2^{pq} K_{l-1,q} + \sum_{q=p+1}^{w} A_1^{pq} K_{l,q} \right), l \geq 2.$$

$K_{-1,p}$ is the probability that the original process starts in level n and phase in S^p and moves to state $(n - 1, +)$ after

spending some time in the states of this level with phases in $\{S^p, \cdots, S^w\}$. Although the chain could visit a state in level $n+1$ with phase in $\{S^p, \cdots, S^w\}$, for the process to visit level $n-1$, it first needs to go back to level n by visiting a state with phase in S^+. This would make the first visit to a state with phase in S^+ to be in level n, and not in level $n-1$ as required. The other matrices can be defined similarly. The matrices $K_{l,p}$ are sequentially computed from $p = w$ to 1.

We now define the block matrices $(\bar{A}_k)_{k\geq 0}$ using the $(K_{l,p})_{l\geq -1}$ matrices. Let the (i,j)-th entry of $\bar{A}_k$ hold the probability that, given that the original process starts in state (n,i), with $i \in S^+$ and $n > 1$, its first transition to another state with phase in S^+ is to $(n+k-1, j)$, with $k \in \{0, 1, 2, \cdots\}$. Hence,

$$\bar{A}_0 = A_0^{++} + \sum_{p=1}^{w} A_1^{+p} K_{-1,p},$$

$$\bar{A}_1 = A_1^{++} + \sum_{p=1}^{w} A_1^{+p} K_{0,p} + \sum_{p=1}^{w} A_2^{+p} K_{-1,p}$$

$$\bar{A}_2 = A_2^{++} + \sum_{p=1}^{w} A_1^{+p} K_{1,p} + \sum_{p=1}^{w} A_2^{+p} K_{0,p}$$

$$\bar{A}_k = \sum_{p=1}^{w} A_1^{+p} K_{k-1,p} + \sum_{p=1}^{w} A_2^{+p} K_{k-2,p}, \quad k \geq 3.$$

To define $\bar{A}_0$, we observe that the transition from state (n,i) to state $(n-1,j)$, with $i,j \in S^+$, can only happen in two ways: the chain can go directly to level $n-1$ by visiting a state in phase S^+; or the chain can go to a state in level n with phase in $\{S^1, \cdots, S^w\}$, and after a sojourn within these states, it moves downward to a state with phase in S^+. An upward transition to level $n+1$ is not allowed as it would require an intermediate visit to a state in level n and phase in S^+. The other matrices can be defined similarly. The matrices $\bar{A}_k$ are sequentially computed from $k = 0$ to c, where c is the smallest positive integer that satisfies $\sum_{i=0}^{c} \bar{A}_k \boldsymbol{e} > (1-\epsilon)\boldsymbol{e}$, where $\epsilon = 10^{-14}$. Notice that it suffices to store two sets of $K_{k,p}$ matrices at a time in order to compute $\bar{A}_k$, with $p \in \{1, 2, \cdots, w\}$.

2) Computing G_p ($p = 1, \cdots, w$): Given the structure of A_0, A_1, A_2, and G, shown in equations (5) and (6), Equation (4) can be written block-wise as

$$G_p + (A_1^{pp})^{-1} A_2^{pp} G_p G_+ = (-A_1^{pp})^{-1} \left(A_0^{p+} \sum_{q=p+1}^{w} (A_1^{pq} G_q + A_2^{pq} G_q G_+) \right),$$

for $p = 1, 2, \ldots, w$. As a result, we can find the matrices G_p by solving these Sylvester matrix equations, starting from $p = w$ and operating backwards. Since there are w equations, and each of them can be solved in $O(r^3)$ time with the Hessenberg-Schur method [24], the overall computation time to find the G_p matrices is $O(wr^3)$. Instead, the method in [8] requires $O(w^3 r^3)$ time, which limits the values of w that can be considered. In fact, under replication, w grows quadratically with the number of substasks, making necessary the analysis of instances with large w.

Remark 1: Notice that the solution method proposed here is not restricted to the case of parallel jobs considered in this paper. For instance, using the results in [25], any acyclic PH distribution (APH) can be transformed to have an entry vector τ with all the mass in the first phase, and an upper-triangular matrix T. Thus, any MAP/APH/1 queue will fit within our assumptions regarding the structure of the QBD blocks. Also, the block upper-triangular structured assumed for A_1 and A_2 can be exchanged by a block lower-triangular structured, without losing the computation gains obtained.

E. Performance measures

After obtaining the matrix G, we can compute the matrix R and the stationary distribution $\{\boldsymbol{\pi}_i\}_{i\geq 0}$. The average number of jobs in the node $E[N]$ is then

$$E[N] = \pi_1 (I - R)^{-2} \boldsymbol{e}.$$

From Little's law [18], the mean number of jobs waiting in the queue, $E[N_q]$, and the expected response time of successful and failed jobs, $E[R_i]$ ($i \in \{S, F\}$) can be computed as

$$E[N_q] = E[N] - E[S]/E[A],$$
$$E[R_i] = E[N_q]E[A] + 1/\mu_i.$$

Here $E[A] = 1/\lambda$, with λ as defined in Equation (2), is the mean inter-arrival time, and $E[S]$ is the mean job service time, obtained with Equation (1).

F. Computation times

To illustrate the behavior of the proposed solution approach, named **ST**, we compare its computation times with traditional methods, such as FI and CR, and with the method in [8], referred to as RT. The experiments were performed in MATLAB, using the SMCSolver tool [21], using 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 Erlang-2 (E_2) and hyper-exponential with two phases (H_2) inter-arrival distributions, and general order-2 MAPs (MAP_2). We thus cover a broad range of behaviors in terms of variability, measured by the squared coefficient of variation (SCV), defined as $C_X^2 = Var[X]/E^2[X]$, for a random variable X. The E_2 and H_2 distributions represent the $C_X^2 < 1$ and $C_X^2 \geq 1$ cases, respectively, while the MAP_2 considers the autocorrelated case. The parameters of the H_2 distribution are computed using the moment-matching method in [26], while the matrices D_0 and D_1 of the MAP are obtained with the method in [27]. We consider MAPs with decay rate of the autocorrelation function equal to 0.9, and SCV equal to 2 and 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 case.

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

Arrivals	ρ	CR	FI	B1		G_p		Total		Ratio	
				RT	ST	RT	ST	RT	ST	RT	ST
E_2	0.2	85.4	87.72	3.0	0.2	4.8	0.2	7.9	0.4	10.8	**213.5**
	0.5	95.2	145.1	3.1	0.4	4.8	0.2	7.9	0.6	12.1	**158.7**
	0.9	95.2	665.5	3.2	0.5	4.8	0.2	8.0	0.7	11.9	**136.0**
H_2 ($C_X^2 = 2$)	0.2	83.1	132.5	3.1	0.3	4.9	0.2	8.0	0.5	10.4	**166.2**
	0.5	107.5	273.9	3.2	0.5	4.8	0.2	8.0	0.7	13.4	**153.6**
	0.9	107.7	1702.3	3.2	0.68	4.8	0.2	8.1	0.9	13.3	**119.7**
MAP_2 ($C_X^2 = 2$)	0.2	82.9	132.8	3.3	0.4	4.9	0.2	8.2	0.5	10.1	**165.8**
	0.5	107.3	273.9	3.3	0.5	4.9	0.2	8.1	0.7	13.2	**153.3**
	0.9	107.7	1707.3	3.2	0.76	4.9	0.16	8.2	1.0	13.1	**107.7**
H_2 ($C_X^2 = 10$)	0.2	95.1	160.6	3.1	0.41	4.9	0.2	8.0	0.6	11.9	**158.5**
	0.5	119.9	579.3	3.2	0.6	4.9	0.2	8.1	0.8	14.8	**149.9**
	0.9	107.4	2156.9	3.4	0.9	4.8	0.2	8.3	1.1	12.9	**97.6**
MAP_2 ($C_X^2 = 10$)	0.2	95.3	160.6	3.3	0.4	4.8	0.2	8.1	0.6	11.8	**158.8**
	0.5	119.7	578.9	3.2	0.6	4.9	0.2	8.1	0.8	14.8	**149.6**
	0.9	107.6	7337.4	3.3	0.84	4.8	0.17	8.1	1.1	13.3	**97.8**

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

Arrivals	ρ	B1		G_p		Total		Ratio
		RT	ST	RT	ST	RT	ST	
E_2	0.2	159.8	1.9	270.7	1.1	430.5	3.1	**138.9**
	0.5	159.6	3.6	278.8	1.1	438.4	4.7	**93.3**
	0.9	161.4	5.1	283.9	1.1	445.3	6.3	**70.7**
H_2 ($C_X^2 = 2$)	0.2	159.3	3.6	281.1	1.1	440.7	4.3	**102.5**
	0.5	160.6	4.8	281.6	1.1	442.2	5.9	**74.9**
	0.9	162.5	7.1	284.5	1.1	447.1	8.3	**53.9**
MAP_2 ($C_X^2 = 2$)	0.2	159.3	3.4	280.6	1.1	439.9	4.5	**97.8**
	0.5	160.2	5.8	280.1	1.1	440.4	7.0	**62.9**
	0.9	163.3	7.2	296.2	1.1	459.6	8.6	**53.4**
H_2 ($C_X^2 = 10$)	0.2	160.0	3.6	280.7	1.1	440.8	4.7	**93.8**
	0.5	160.8	5.6	280.6	1.1	441.4	6.7	**65.9**
	0.9	163.5	7.7	289.1	1.1	452.6	8.8	**51.4**
MAP_2 ($C_X^2 = 10$)	0.2	160.1	4.1	280.7	1.1	440.7	5.2	**84.8**
	0.5	161.2	6.6	282.7	1.1	443.9	7.7	**57.6**
	0.9	166.2	7.7	290.4	1.1	456.9	8.8	**51.9**

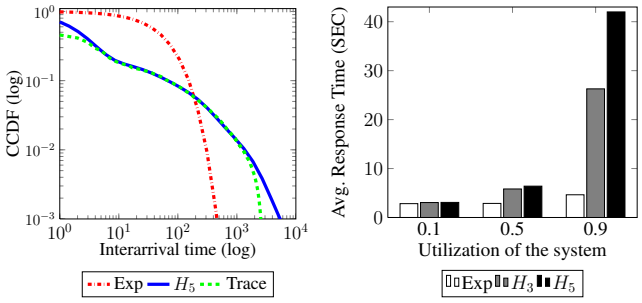
Table III summarizes the results for the case of 50 subtasks. The rows in Table III consider the different arrival processes and load levels ρ. The first two columns show the time to compute G using CR and FI on the full-size QBD. The next two columns show the time to compute the M/G/1-type blocks using the RT and ST approaches. We observe that ST offers shorter computation times, although the largest gain in this case is shown in the next two columns, which show the time to find the G_p matrices with the RT and ST methods. Here we observe a significant gain, as the computation times with our ST method are one order of magnitude smaller than with the RT method. The next two columns summarize the total computation times required by the RT and ST methods, showing the significant advantage of ST over RT. These results are compared with the traditional CR in the last two columns, where we depict the ratio between the column CR and the (Total) columns RT (R-RT), and ST (R-ST), respectively. The computation times with the ST method are 97-214 times shorter than with the CR method, while the RT method offers a reduction of only 10-15 times.

Table IV shows similar results for the 100 subtask case, although in this case no results are reported for the CR and FI algorithms as they run out of memory due to the large size of the QBD blocks. Notice that these and all the algorithms are implemented in MATLAB using the same data structures. We show instead, in the last column, the ratio between the columns RT and ST (R-RS). We observe that the ST method produces results 51-139 times faster than RT. Notice that in this case both the computation of blocks and of the G_p matrices show very important reductions when using ST instead of RT. Comparing Tables III and IV, we notice that when the number of subtasks increases, the time to compute the M/G/1-type blocks of both the RT and ST methods, and the time to find the G_p matrices increase. This is caused by the increase in the phase space due to the larger number of subtasks. Also, when the utilization ρ increases, the FI algorithm and both methods for computing the M/G/1-type blocks require longer times, since a larger load increases the rate of upward transitions per time unit.

Another gain with the proposed approach is the significant reduction in memory requirements, since it divides the $m \times m$ matrices into small non-zero $r \times r$ blocks. On the machine described, this method is able to solve systems with block size up to 40000. We also highlight that the proposed approach offers a good numerical behavior, which we measure with the residual error of Equation (4), $\|A_2 + A_1 G + A_0 G^2\|_\infty$, using the infinity norm. In all the experiments conducted, the residual error was below 10^{-14}, revealing the good numerical behavior of the proposed approach. This behavior is expected since the CR and the Hessenberg-Schur methods on which the approach relies are numerically stable, and all the steps are based on sums and products of positive quantities.

G. The multi-node system

Based on the analysis of a single node, we now consider the case of multiple nodes with a central RR scheduler. The key observation is that the RR scheduling affects the arrival process, but this process is still a MAP. As a result, we can focus on a single tagged processing node, by appropriately modifying its arrival process. The new process is a MAP with



(a) Inter-arrival times CDF (b) Response time

Fig. 4: The effect of different arrival process

matrices C_0 and C_1 given by

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

These matrices are of size cm_a, where c is the number of nodes, and such that a new arrival to the tagged node will only occur after $c-1$ jobs have been assigned to each of the other nodes. Notice that now the QBD blocks size increase by a factor c, making more relevant the use of the proposed method to analyze large instances, as traditional methods would be limited by the block size.

VI. Results and Discussion

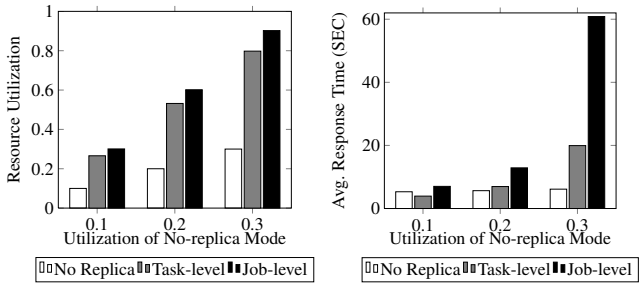
A. Case Study: RICC Log

In this section, we evaluate the proposed fault-tolerance approach using the workload log from a real parallel cluster. We make use of the log of RICC (RIKEN Integrated Cluster of Clusters) [28], available on the Parallel Workloads Archive [29]. This log contains a total of 447,794 records of jobs submitted to the RICC installation in Japan from May to September 2010. We make use of the trace to parameterize our model, estimating a subtask service rate of $\mu = 0.9015$ and a failure rate of $\alpha = 0.0171$. Moreover, given the high variability observed in the job inter-arrival-times, we consider PH-distributed inter-arrival times. Specifically, we use the method in [30], as implemented in the jPhase package [31], to find a hyper-exponential distribution with r phases (H_r). Table V shows the mean relative error between the trace empirical CDF and the fitted CDF of the exponential and hyper-exponential distributions. We observe how the error decreases with the number of phases, but the improvement is limited for 6 or more phases. We therefore choose an H_5 representation, and show in Figure 4(a) the complementary CDF (CCDF) of the real trace, the exponential and the H_5 distribution. Clearly, the H_5 distribution captures the trace behavior much better than the exponential, especially its long tail.

To illustrate the impact of the hyper-exponential arrivals, in Figure 4(b) we compare the average response times of a system with task-level replication and three different inter-arrival distributions while keeping the mean inter-arrival time

TABLE V: Absolute error (%) of different arrival processes

Distribution	% Err	Distribution	% Err	Distribution	% Err
Exp	2.87	H_2	1.01	H_3	0.34
H_4	0.11	H_5	0.07	H_6	0.06



(a) Resource utilization (b) Average response time

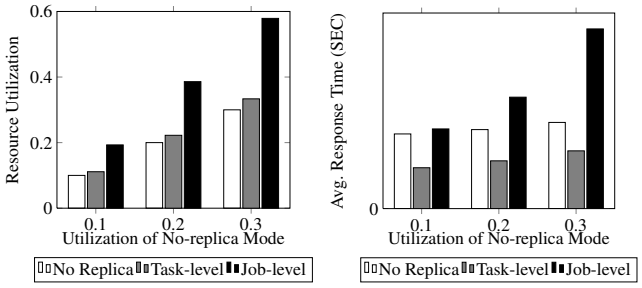
Fig. 5: Effect of replication with failure rate $\alpha = 1.7E^{-2}$

fixed. The response times are similar under low loads but significantly different for medium and large loads, highlighting the influence of the arrival process variability on the system performance. For this and the following results, we consider a system with 2048 processors grouped in 32 nodes, each with 64 processors, serving jobs with 64 subtasks. When introducing replicas, the processors are grouped into 16 nodes, each with 128 processors, serving jobs and their replicas.

In order to illustrate the behavior of the proposed fault-tolerance approach, we compare the reliability, utilization and average response time of the system without replication, and with task-level and job-level replications. As replication is expected to be adopted under underutilized conditions, we scale the mean arrival rate such that the system without replicas has three utilization levels: 0.1, 0.2 and 0.3.

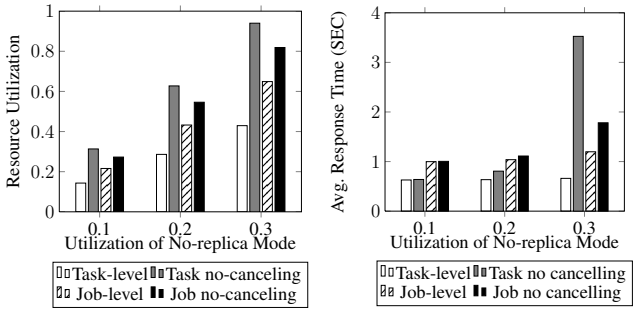
The reliability of the system without replica, and with job and task level replications is 30.04%, 51.05% and 97.81%, respectively, i.e., the reliability improves by 226% using task-level replication, and 70% using job-level replication. Although the failure rate of a single subtask is relatively low, the reliability of the system without replicas or with job-level replication is low because a single subtask failure causes the failure of a whole job or its replica. This is not the case under task-level replication as the job only fails if both a subtask *and* its replica fail.

Figure 5(a) depicts how the utilization for job-level replication is larger than for task-level replication, and the impact this has on the response times is shown in Figure 5(b). We observe how the larger utilization for job-level replication causes the response times to shoot up when the utilization of the system without replica is 0.3. In fact, the response times are 3 times larger than for task-level replication. Notice that the no-replica case shows short response times because these correspond to jobs that complete service successfully, and given the low reliability of this scenario, only short jobs can complete without failure. Further, under a 0.1 utilization for the no-replica case, the response times for task-replication are 26% shorter than for the no-replica mode, but these become



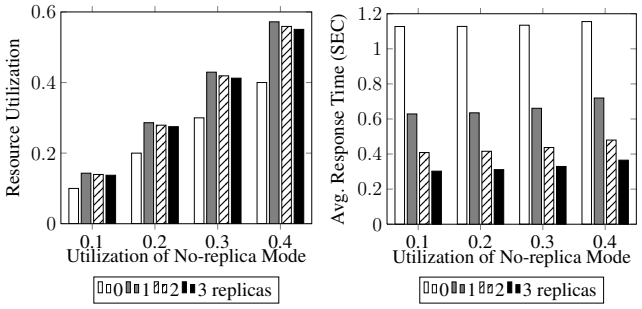
(a) Resource utilization (b) Average response time

Fig. 6: Effect of replication with failure rate $\alpha = 1.7E^{-3}$



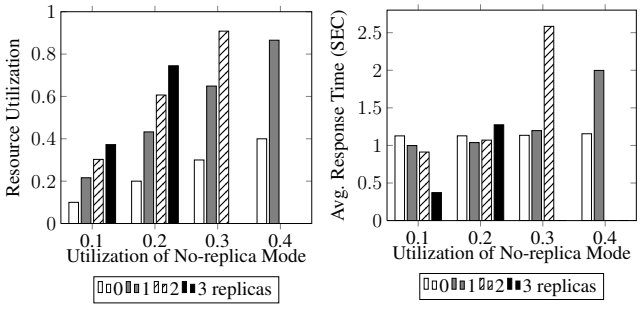
(a) Resource utilization (b) Avg. response time

Fig. 7: Replication with and without canceling



(a) Resource utilization (b) Response time

Fig. 8: Multiple replicas under **task-level replication**



(a) Resource utilization (b) Response time

Fig. 9: Multiple replicas under **job-level replication**

larger for larger utilizations. This is because the introduction of replicas has two effects: it increases the resource utilization, leading to an increase of the response times; and at the same time it allows the selection of the first replica that finishes, potentially reducing the response times. If the overall reliability is low, as is the case here for job-level replication, then one of the replicas will likely fail and the first effect will be dominant, causing an overall increase of the response times. In fact, the reduction in response times is more significant when the subtasks are more reliable, and under a relatively low load, as illustrated in Figure 6. Here we reduce the failure rate by an order of magnitude to $\alpha = 1.71E^{-3}$, increasing the reliability of the system without replica, and with task and job-level replications to 88.58%, 99.98% and 98.70%, respectively. As shown in Figure 6, the reduction in the mean response time for task-level replication, compared to the no-replica case, is more significant, being 45%, 40%, and 33% less for the three utilization levels considered, respectively.

B. Comparison with no canceling modes

We now compare the proposed replication methods with canceling with their counterparts without canceling. We found that, considering the setup based on the case study, the system with task-level replication, but *without* canceling, was already unstable when the no-replica system utilization was just 0.2. This highlights the importance of the replica cancellation scheme to limit the resource utilization. We therefore focus here on a simpler setup in the following results, with 12 nodes, job size 10, service rate $\mu = 2.5$, failure rate $\alpha = 0.1$ and Poisson arrivals. The results of the replication-without-canceling case are obtained using a simulation model implemented in Java, considering three different utilization levels for the system without replication: 0.1, 0.2 and 0.3.

Although both approaches, with and without canceling, have the same reliability, their response time and utilization differ, as illustrated in Figure 7. As expected, the utilization and average response time without canceling are higher than with canceling. In particular, we observe how, when the no-replica utilization is 0.3, the response time increases dramatically for the task-level replication without canceling. This is caused by the high resource utilization achieved under the no-canceling mode, which is about twice that of the canceling mode. Again, this highlights the impact that canceling has when implementing the concurrent replication scheme.

C. The effect of additional replicas

We now consider the possibility of adding more than one replica, and its effect on the system reliability and performance. Figure 10 shows how the reliability increases as the number of replicas increases. For task-level replication, the improvement in reliability is significant after adopting a single replica, but is limited when considering more replicas. Instead, in job-level replication, adopting more replicas considerably increases the system reliability. Figures 8 and 9 show the system utilization and average response time for varying number of replicas for task-level and job-level replications, respectively. The results not shown correspond to cases where the expected utilization with replication exceeds 1.0. This is actually the case for job-level replication when the no-replica utilization is 0.3 and 0.4. Interestingly, for task-level

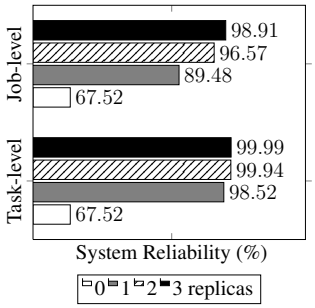


Fig. 10: Reliability under different replication techniques and multiple replicas

replication, we observe how the utilization and response times actually decrease with the addition of replicas. As mentioned before, this is caused by the high reliability achieved and the availability of multiple copies of the same subtask, from which the first one that finishes determines its completion time, and cancels its replicas. A different trend is observed under job replication, where the reliability is lower and the utilization higher, significantly increasing the response times.

Job-level replication is inefficient, albeit being simpler to implement. On the other hand, task-level replication is desirable when considering the significant improvement of reliability, resource efficiency and low latency. Given the efficiency of task-level replication in exploiting the concurrent replicas, it appears enough to adopt a single replica. Furthermore, the adoption of more replicas also implies a higher overhead to oversee their execution and proceed to cancel the outstanding replicas when the first of them completes its service.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have evaluated the ability of concurrent replication with canceling to improve the reliability of parallel systems subject to request failures. The exact analysis of this system is performed using a stochastic model, for which we develop a numerical solution method that is able to tackle large instances effectively. We demonstrate that task-level replication with canceling offers an effective way of improving the system reliability, with the potential to reduce latency, while limiting the increase in resource utilization. Since important performance metrics can be efficiently derived by using the proposed method, users or service providers can decide whether to adopt replicas or not depending on their service level agreements. In the future, we intend to investigate the performance of the proposed approach, and the overhead of the replication and cancellation mechanisms, by implementing it in a real parallel system.

REFERENCES

[1] A. Nguyen-Tuong, A. S. Grimshaw, and J. F. Karpovich, *Fault tolerance via replication in coarse grain data-flow*. Springer, 1996.

[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 *Proceedings of the 2011 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, no. 60, p. 80, 2012.

[4] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*, 2013.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[6] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 13–18.

[7] P. G. Harrison and Z. Qiu, "Performance enhancement by means of task replication," in *Computer Performance Engineering*. Springer, 2013, pp. 191–205.

[8] J. F. Pérez and B. Van Houdt, "Exploiting restricted transitions in quasi-birth-and-death processes," in *Quantitative Evaluation of Systems, 2009. QEST'09.* IEEE, 2009, pp. 123–132.

[9] M. T. Özsu and P. Valduriez, "Distributed and parallel database systems," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 125–128, 1996.

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

[11] P. Chan, M. R. Lyu, and M. Malek, "Reliable web services: Methodology, experiment and modeling," in *IEEE International Conference on Web Services, 2007. ICWS 2007.* IEEE, 2007, pp. 679–686.

[12] M. Treaster, "A survey of fault-tolerance and fault-recovery techniques in parallel systems," *arXiv preprint cs/0501002*, 2005.

[13] A. S. Lebrecht and W. J. Knottenbelt, "Response time approximations in fork-join queues," in *23rd UK Performance Engineering Workshop (UKPEW)*, 2007.

[14] P. Harrison and S. Zertal, "Queueing models of RAID systems with maxima of waiting times," *Performance Evaluation*, vol. 64, no. 7, pp. 664–689, 2007.

[15] D. Menasce, "Response-time analysis of composite web services," *Internet Computing, IEEE*, vol. 8, no. 1, pp. 90–92, 2004.

[16] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. Siam, 1999, vol. 5.

[17] N. Gautam, *Analysis of Queues: Methods and Applications*. CRC Press, 2012.

[18] W. J. Stewart, *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

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

[20] A. Graham, *Kronecker products and matrix calculus with applications*. Horwood Chichester, 1981, vol. 108.

[21] D. Bini, B. Meini, S. Steffé, and B. Van Houdt, "Structured markov chains solver: software tools," in *Proceeding from the 2006 workshop on Tools for solving structured Markov chains*. ACM, 2006, p. 14.

[22] M. F. Neuts, *Structured stochastic matrices of M/G/1 type and their applications*. CRC Press, 1989, vol. 5.

[23] V. Ramaswami, "A stable recursion for the steady state vector in markov chains of M/G/1 type," *Stochastic Models*, vol. 4, pp. 183–188, 1988.

[24] J. D. Gardiner, A. J. Laub, J. J. Amato, and C. B. Moler, "Solution of the sylvester matrix equation AXB T+ CXD T= E," *ACM Transactions on Mathematical Software (TOMS)*, vol. 18, no. 2, pp. 223–231, 1992.

[25] A. Cumani, "On the canonical representation of homogeneous markov processes modelling failure-time distributions," *Microelectronics Reliability*, vol. 22, no. 3, pp. 583–602, 1982.

[26] W. Whitt, "Approximating a point process by a renewal process, I : Two basic methods," *Operations Research*, vol. 30, no. 1, pp. 125–147, 1982.

[27] A. Heindl, "Inverse characterization of hyperexponential MAP(2)S," in *11th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA)*, vol. 31. Citeseer, 2004.

[28] "The RICC log." [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html

[29] "Parallel workloads archive." [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[30] R. El Abdouni Khayari, R. Sadre, and B. R. Haverkort, "Fitting world-wide web request traces with the EM-algorithm," *Performance Evaluation*, vol. 52, no. 2, pp. 175–191, 2003.

[31] J. F. Pérez and G. Riano, "jPhase: an object-oriented tool for modeling phase-type distributions," in *Proceeding from the 2006 workshop on Tools for solving structured Markov chains*. ACM, 2006, p. 5.