2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

DieHard: reliable scheduling to survive correlated failures in cloud data centers

Mina Sedaghat[a], Eddie Wadbro[a], John Wilkes[b], Sara De Luna[c], Oleg Seleznjev[c], and Erik Elmroth[a]
[a] Dept. of Computing Science, Umeå University, Sweden, {mina, eddiew, elmroth}@cs.umu.se
[b] Google Inc., johnwilkes@google.com
[c] Dept. of Mathematics and Mathematical Statistics, Umeå University, Sweden
{sara.de.luna, oleg.seleznjev}@math.umu.se

***Abstract*—In large scale data centers, a single fault can lead to correlated failures of several physical machines and the tasks running on them, simultaneously. Such correlated failures can severely damage the reliability of a service or a job.**

This paper models the impact of stochastic and correlated failures on job reliability in a data center. We focus on correlated failures caused by power outages or failures of network components, on jobs running multiple replicas of identical tasks. We present a statistical reliability model and an approximation technique for computing a job's reliability in the presence of correlated failures.

In addition, we address the problem of scheduling a job with reliability constraints. We formulate the scheduling problem as an optimization problem, with the aim being to achieve the desired reliability with the minimum number of extra tasks. We present a scheduling algorithm that approximates the minimum number of required tasks and a placement to achieve a desired job reliability. We study the efficiency of our algorithm using an analytical approach and by simulating a cluster with different failure sources and reliabilities. The results show that the algorithm can effectively approximate the minimum number of extra tasks required to achieve the job's reliability.

***Index Terms*—Cloud computing; Scheduling; Reliability; Fault tolerance; Correlated failures;**

I. Introduction

Data centers achieve high reliability through the use of failure tolerant hardware, network equipment, and architectures or via adopting sophisticated management solutions such as replication and recovery techniques. Many of these techniques cope with failures of a single individual component such as a machine or a software component, rather than providing overall reliability for jobs or services. However, it is not generally valid to assume that machines fail independently and component failures are uncorrelated. Correlated failures such as failures due to power outages or network component failure are rare [1] but have significant effects on system reliability [2], [3], [4], [5], [6]. Ignoring the impact of correlated failures can cause reliability to be overestimated by at least two orders of magnitude [2].

In this work we present a statistical model for job reliability in a cloud data center, in the presence of stochastic and correlated failures. The model quantifies the impacts of correlated failures on the overall reliability of a job comprising multiple identical tasks, which can be run in containers [7] or virtual machines. The job reliability is defined as the probability that at least a minimum number of tasks will continue running throughout the job's runtime. We do this to model batch jobs that need to have a certain number of workers to finish by a deadline, or long-running service jobs that need to have a minimum number of workers to meet some external load, such as a worst-case query rate.

We specifically focus on correlated failures caused by power outages and failures of network components. In our model, power nodes and network components have different failure rates and their failures affect different sets of machines, known as *failure domains*. For example, the machines that are affected by a power outage will not necessarily be the ones that are affected by a network component failure [8].

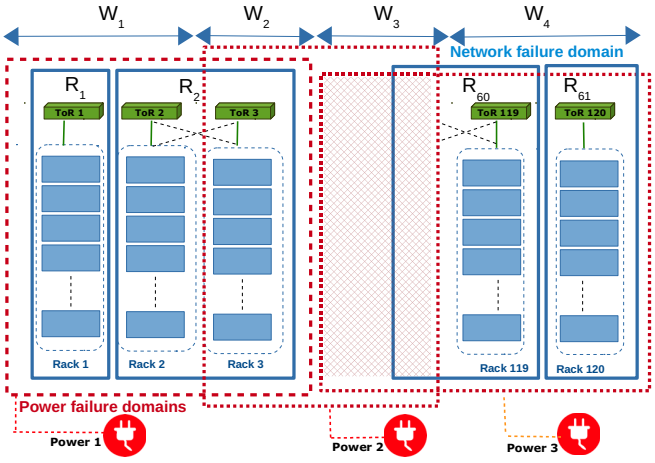

Fig. 1: A representative power and networking structure for a datacenter. The machines are connected via 120 ToR switches and they are plugged into to 3 power nodes. A set of ToR switches are backed up by a redundant switch. All machines are connected to at least one power node, while half of the machines are also supported by an extra power node. This topology forms 4 power failure domains (W_1, W_2, W_3, W_4), and 61 network failure domains $(R_1, ..., R_{61})$.

Delivering job reliability cannot be considered independently from job scheduling because job reliability is highly dependent on the placement of the tasks over different failure domains (e.g. different racks or power failure domains). To achieve job reliability, scheduling and allocation decisions should take into account the probabilities of failure for different machines, racks, and other potential failure domains,

978-1-5090-2453-7/16 $31.00 © 2016 IEEE
DOI 10.1109/CCGrid.2016.11

along with the impact of failures in these domains on the job.

Using the proposed reliability model, we introduce a scheduling algorithm to achieve a given level of reliability for a particular job. The objective of the algorithm is to achieve an overall reliability for a job while using the minimum necessary number of extra tasks (replicas), and to schedule the job across multiple different failure domains. Scheduling tasks over different failure domains makes jobs robust against the negative impact of a single failure, while running extra tasks enables the job to compensate for losses and deliver the targeted minimum reliability during recovery.

The scheduling algorithm approximates a minimum number of required replicas and selects a subset of machines in the cluster to run replicas of the tasks. Note that replicating a task within a failure domain cannot prevent it from being affected by a correlated failure of that domain, no matter how much redundancy is introduced. Similarly, spreading allocations over different failure domains alone is not sufficient to guarantee that the targeted minimum reliability will be achieved. Therefore, a combination of good placement and replication is required to ensure high overall reliability.

The contributions of this paper are:

1) A statistical reliability model for a job running multiple identical tasks, deployed in a cloud data center. The model captures the impacts of stochastic and correlated failures due to power or network node failure on job reliability. We also propose an approximation technique for estimating a given job's reliability.
2) A scheduling algorithm to approximate the number of tasks required to guarantee that a job will achieve the desired reliability, and to select a subset of machines to run those tasks.
3) Analytical proofs of the algorithm's validity and a simulation-based illustration and evaluation of the algorithm for a cluster with different failure sources and jobs with different target reliabilities. The evaluation shows that the algorithm can effectively approximate the minimum number of extra tasks required to achieve a job's reliability.

II. Problem formulation

A job J arrives at a data center and executes a group of identical tasks [9]. This is a common application model for datacenters, where multiple tasks (containers or Virtual Machines (VMs)) are needed to serve the job's demand. Each task has an expected compute and memory demand of C and M, respectively. The job starts at time t_1 and runs for time T. A job is successful if the probability that at least K tasks will be running at all times during the $[t_1, t_1+T]$ time interval is greater than or equal to a threshold $S_{\min}$. In addition to the required K tasks, a number of *extra tasks* can be created to increase the expected probability of success. A set of tasks may fail (and stop running) simultaneously if they share a common source of potential failure such as a common power node or a common network device.

The tasks are deployed in a cluster of physical machines. Each machine p has an available CPU and memory capacity of c_p and m_p, respectively. Machines are connected by a set of network nodes $\bar{R}$ and the electrical power is supplied by a set of power nodes $\bar{W}$. Failure of a power node causes failure on all the associated machines and the tasks deployed on those machines.

Given a fixed power and network topology, for each job J with desired number of running tasks K, we want to determine n, the required number of extra tasks, and $\mathbf{x}$, a placement of $N = K + n$ tasks on the machines, so at least K tasks are running at all times during the job runtime, with a certain probability threshold, $S_{\min}$. The job reliability is defined as the probability of the job operating for a certain amount of time, with at least K running tasks, at all times. In our model, each machine runs at most one task from each job. Tasks cannot be restarted and their failures are terminal.

Given a placement vector $\mathbf{x}$ representing the distribution of tasks over different failure domains, and $S(\mathbf{x})$ as the calculated reliability of the job, the goal is to find $n^* = \min \{n \mid S(\mathbf{x}) \geq S_{\min}\}$ during the job's runtime T, subject to the relevant capacity constraints.

A. Network and power topology

We model the dependencies among system components as a 2-level multi-rooted tree, where the leaves are the machines and the parents are power and network nodes. Each network component failure disconnects 60 to 120 machines (one or two racks), whereas a Power Distribution Unit (PDU) failure leads to the outage of 20 to 60 racks [10]. We assume that the power and network topology may contain redundancy, where each machine can be connected to an extra network node or be supported by an extra power node. In such cases, a machine is functional as long as at least one power node and one network node are available to it.

We also define a network failure domain R as the set of machines that share the same network components and are thus at risk of a concurrent failure. Similarly, we define a power domain W as the set of machines that share the same power nodes.

We define r_i as a random binary variable, where $r_i = 1$ if the network failure domain i is available and running, and $r_i = 0$ otherwise. The network failure domain is available as long as one network component is functional. Similarly, we define the random binary variable w_j, where $w_j = 1$ if the power failure domain j is available and $w_j = 0$ otherwise. The power failure domain is available as long as one power node is functional.

In addition, we define the failure domain F_l, as the set of tasks deployed in a power failure domain that are also connected via a single network failure domain. The binary random variable $f_l = 1$ if at least one network node and one power node in F_l are operational. Figure 1 shows a data center topology with respect to network and power failure domains, showing the redundancies in the system.

The network and power nodes break independently and are not repairable. The time to failure of each network node during

a job runtime T is exponentially distributed with a random failure rate of $0.000022 \leq \lambda \leq 0.000032$ per hour, given a Mean Time To Failure (MTTF) of 3.5 to 5 years. Similarly, the time to failure of the power nodes (i.e. the PDUs) during a job with runtime T is assumed to be exponentially distributed over time with a failure rate of $\lambda = 0.4 \times 10^{-6}$ per hour [11]. We also assume that all machines have an identical probability of hardware failure.

B. The reliability model

To model the impact of correlated failures on reliability, we specify the probability of each subset of tasks being unavailable. Let us assume that x_l is the number of tasks running in a failure domain F_l, $\mathbf{x} = [x_1, x_2, \ldots, x_L]$ is the placement vector of the tasks over the failure domains. The quantity x_l also happens to be the number of tasks that fail when failure domain F_l goes down. Moreover, $\mathbf{f} = [f_1, f_2, \ldots, f_L]$ is a failure state vector showing the availability or failure of the failure domains and L is number of failure domains. The number of running tasks can then be calculated as:

$$N(\mathbf{f}, \mathbf{x}) = \sum_{l=1}^{L} f_l x_l \tag{1}$$

Let

$$P^R(\mathbf{r}) = \prod_{\{i|r_i=1\}} A_i^R(T) \prod_{\{i|r_i=0\}} (1 - A_i^R(T)) \tag{2}$$

$$P^W(\mathbf{w}) = \prod_{\{j|w_j=1\}} A_j^W(T) \prod_{\{j|w_j=0\}} (1 - A_j^W(T)) \tag{3}$$

where $P^R(\mathbf{r})$ and $P^W(\mathbf{w})$ are the probabilities of network failure domains and power failure domains for failure state vectors $\mathbf{r} = [r_1, r_2, \ldots, r_R]$ and $\mathbf{w} = [w_1, w_2, \ldots, w_W]$, respectively. In addition, $A_i^R(T)$ and $A_j^W(T)$ are the reliabilities of network failure domain i and power failure domain j for a job with T runtime, respectively. The reliability of the job, $S(\mathbf{x}) \geq P(N(\mathbf{f}, \mathbf{x}) \geq K)$, can be calculated as the sum of the probabilities of all possible combinations of failure events when the number of running tasks $N(\mathbf{f}, \mathbf{x}) \geq K$. This can be written as:

$$S(\mathbf{x}) = \sum_{\{(\mathbf{r},\mathbf{w})|N(\mathbf{f},\mathbf{x}) \geq K\}} P^R(\mathbf{r}) P^W(\mathbf{w}) \tag{4}$$

In this formulation, the component failures are independent of each other. However, each component failure can terminate all tasks in one or more failure domains.

C. Approximating the reliability value

In practice, computing the exact reliability value is complex and computationally expensive. Therefore, we propose a method for approximating the reliability value.

The reliability function $S(\mathbf{x})$ is a step function; it is assumed that a job's reliability increases in discrete steps as the number of extra tasks is increased. However, different types and combinations of failures do not necessarily reduce the reliability value to the same extent. A network node has a much greater failure probability than a power node, but fewer extra tasks are required to mitigate the impact of a network node failure.

We reformulate equation (4) as a sum over the total number of failures in the system. To do this, for $i = 0, 1, \ldots, R$ and $j = 0, 1, \ldots, W$, we define:

$$S_{i,j}(\mathbf{x}) = \sum_{\left\{ (\mathbf{r},\mathbf{w}) | \begin{array}{c} \|\mathbf{r}\|_1 = R - i \\ \|\mathbf{w}\|_1 = W - j \\ N(\mathbf{f},\mathbf{x}) \geq K \end{array} \right\}} P^R(\mathbf{r}) P^W(\mathbf{w}) \tag{5}$$

In this equation, $S_{0,0}(\mathbf{x})$ is the reliability of the job if none of the network or power failure domains fail, and $S_{i,j}(\mathbf{x})$, $1 \leq i \leq R$ and $1 \leq j \leq W$ is the probability that the job has $N(\mathbf{f}, \mathbf{x}) \geq K$ tasks, given i network failures and j power failures. By combining definitions (4) and (5), we get the following expression for the reliability

$$S(\mathbf{x}) = \sum_{i=0}^{R} \sum_{j=0}^{W} S_{i,j}(\mathbf{x}) \tag{6}$$

The probability of failure of multiple components, $S_{i,j}(\mathbf{x})$, depends on the failure probabilities of the individual components, each of which is small. Considering the probability of failures of the components in our model, we can argue:

$$S_{0,0}(\mathbf{x}) \gg S_{1,0}(\mathbf{x}) \gg S_{2,0}(\mathbf{x}) \gg \ldots S_{0,1}(\mathbf{x}) \ldots \gg S_{R,W}(\mathbf{x}) \tag{7}$$

Therefore, as i and j increase, the improvement in system reliability, $S_{i,j}(\mathbf{x})$, becomes progressively smaller. There is thus an optimal number of extra tasks; adding further tasks above this threshold would not contribute substantially to the system's reliability. In other words, as the probability of a high number of failures during a job's lifetime decreases, so too does the need to plan and assign extra tasks to achieve reliability.

The fact that the reliability function increases in discrete steps is useful when approximating the reliability value and estimating the number of extra tasks required to achieve a given reliability. Each step in the reliability function corresponds to a failure arrangement, $(\mathbf{r}, \mathbf{w})$, which specifies the type and number of failures that the existing arrangement is sufficient to cover. A desired reliability can then be achieved by providing the minimum redundancy required to cover the corresponding failure arrangement.

The approximation becomes essential, because of the computational expense of calculating $S(\mathbf{x})$, which necessitates calculation of every possible combination of failures that satisfy $N(\mathbf{f}, \mathbf{x}) \geq K$. To reduce the computational complexity, we approximate the sum on the right hand side of equation (6) by discarding the terms $S_{i,j}(x)$ that correspond to failures of components (i, j) whose probability of failure is negligible with respect to S_{min}. This enables us to obtain approximate reliability values cheap, and also helps us estimate the number of extra tasks required to prevent failure events that have non-negligible effects on reliability relative to S_{min}.

III. Fault-aware Scheduling

To schedule a job in a way that achieves a given reliability, we must approximate the minimum necessary number of extra tasks and identify a placement that satisfy the reliability constraint $S(\mathbf{x}) \geq S_{min}$. Our approximation algorithm is based on the discussion in the previous section. We aim to determine which failures of (i, j) must be compensated for to achieve the target reliability S_{min}. Having identified this set of essential failures, we then provide the minimum level of redundancy necessary to cover them.

To identify the failures for which it is necessary to provide redundancy, we implement a decision tree. Each node in the tree corresponds to a failure of (i, j) network and power failure domains. For each node of the tree, we estimate the number of extra tasks required to compensate for the corresponding failures, n, and approximate the job reliability for the current arrangement. The algorithm starts at the root of the tree (0,0), and initially expands along the power branch (0,1). This is done because providing redundancy for power outages with appropriate task placement also automatically protects against some network failures. If the target reliability, S_{min}, cannot be achieved by covering for one power failure, no amount of additional redundancy with respect to network failure would be sufficient to compensate for this deficiency, so it is necessary to expand further along the power branch. However, if providing redundancy for one power failure domain's failure results in $S(\mathbf{x}) > S_{min}$, there is a chance of obtaining the desired reliability with fewer extra tasks by covering for just some additional network domain failures. If this is the case, we expand along the network branch and iteratively increase the number of possible component failures, (i, j), adding the necessary redundancy to cover these failures at each step. In each expansion, we re-compute the new $\mathbf{x}$ for new $N = K + n$ and its associated $S(\mathbf{x})$. We stop at the node that satisfies the target reliability S_{min}.

The n values of the last two nodes are the lower and upper bound estimates of the approximate minimum number of extra tasks required to achieve $S(\mathbf{x}) \geq S_{min}$. Having determined this interval, we can easily approximate the minimum by performing a bisection search over the n values that satisfy $S(\mathbf{x}) \geq S_{min}$. The number of iterations is limited and small because the bound is usually limited and small. This algorithm is outlined graphically in Figure 2 and more precisely in Algorithm 1.

A. Estimating the number of extra tasks

To estimate the number of extra tasks n, let us first assume that the desired reliability S_{min} can be guaranteed by covering a single network failure domain:

Lemma 1. *To provide full redundancy for one network domain failure, the required number of extra tasks is* $n = \left\lceil \frac{K}{R-1} \right\rceil$.

Proof. Let N be the total number of deployed tasks for the job. To ensure that at least K tasks are running if any single network failure domain fails, at least K tasks must run on each collection of $R-1$ network failure domains. By the

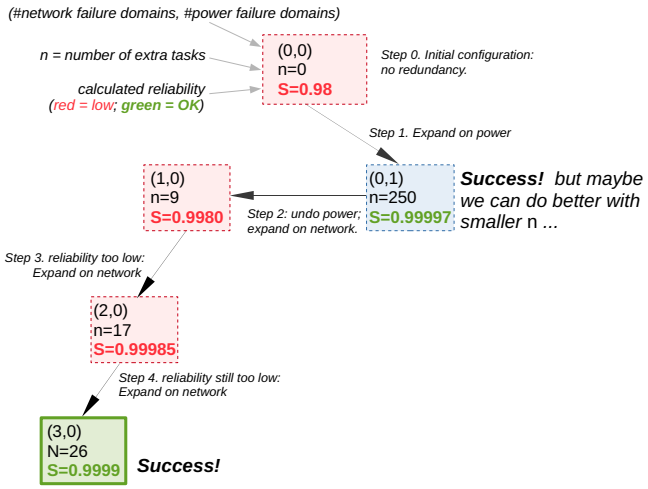

Fig. 2: Decision tree for identifying the required redundancy level, for a job with the minimum number of running task K=500 and minimum target reliability 0.9999.

Algorithm 1 Fault-aware scheduling algorithm

procedure SCHEDULING
  Calculate $S_{0,1}(\mathbf{x})$ assuming $n = \frac{K}{W-1}$, i.e. the reliability of a job in the event of one power failure
  if $(S > S_{min})$ **then**
    Branching($S_{1,0}(\mathbf{x})$); Expand along the network branch
  else $S < S_{min}$
    Branching($S_{0,2}(\mathbf{x})$); Expand along the power branch
  end if
end procedure

procedure BRANCHING($S_{LevelNetwork,LevelPower}(\mathbf{x})$)
  if $S > S_{min}$ **then return x**
  end if
  if LevelNetwork > 0 **then**
    calculate $n = \frac{K \times LevelNetwork}{R - LevelNetwork}$
    $\mathbf{x}$:= Schedule Rn tasks equally on R network failure domains, considering the capacity constraints
    Sort network failure domains according to their reliability
    Update $\mathbf{x}$ after removing $N^* - N$ tasks from the least reliable domains
    calculate job reliability S
  else
    if LevelPower > 0 **then**
      calc $n = \frac{K \times LevelPower}{W - LevelPower}$
      Schedule Wn tasks on W power failure domains
      Sort power failure domains according to their reliability
      remove $N^* - N$ from the least reliable domains
      calculate job reliability S
    end if
  end if
  if $S_{LevelNetwork,\ LevelPower}(\mathbf{x}) < S_{min}$ and (LevelNetwork > 0) **then**
    Branching($S_{LevelNetwork++,\ LevelPower}(\mathbf{x})$)
  else
    if $S_{(LevelNetwork,\ LevelPower)}(\mathbf{x}) < S_{min}$ and (LevelPower > 0) **then**
      Branching($S_{LevelNetwork,\ LevelPower++}(\mathbf{x})$)
    end if
  end if
end procedure

pigeon hole principle [12], at least one of these $R-1$ network failure domains has $n = \lceil \frac{K}{R-1} \rceil$ tasks. To ensure redundancy, the remaining $R-1$ network failure domains must have at least K tasks, so we need $K+n$ tasks to ensure redundancy.

Furthermore, we need to prove that this setting is realizable. In other words, we should be able to place $K+n$ tasks over the R network failure domains such that no domain has more than n tasks. One straightforward way of doing this is to put n tasks on each network failure domain, resulting in $R\lceil \frac{K}{R-1} \rceil \geq K+n$ tasks in total, and then remove any $(Rn)-K-n$ tasks. □

Lemma 2. *To provide full redundancy for $i > 0$ network domain failures, the number of required extra tasks is $n_r = i\lceil \frac{K}{R-i} \rceil$.*

Proof. Lemma 2 is an extension of the statement in Lemma 1. For a job to survive $i \geq 1$ network domain failures, we need for any combination of remaining $R-i$ domains to have at least K tasks. The simplest way is to assume that we have the same number of tasks n in all R domains. Then $Rn - in \geq K$ or $n \geq K/(R-i)$, that is when we can take the smallest number $n = \lceil K/(R-i) \rceil$, where $m = \lceil a \rceil$ is the smallest integer number such that $m \geq a$. Now we obtain

$$n = \left\lceil \frac{K}{R-i} \right\rceil \geq \frac{K}{R-i} \leftrightarrow Rn \geq K + in. \tag{8}$$

Therefore, we get the number of extra tasks

$$n_r = in = i\left\lceil \frac{K}{R-i} \right\rceil. \tag{9}$$

□

Using a similar argument, we can conclude that the number of extra tasks required to provide full redundancy in the event of $j > 0$ power failures is $n_w = \left\lceil \frac{Kj}{W-j} \right\rceil$. The only difference is that deploying $n_w \gg n_r$ tasks automatically covers for the failure of R^* network failure domains due to the deployment of a greater number of spare tasks. Using the equation in Lemma 2, we can find R^*, the number of network domain failures covered by deploying n_w extra tasks:

$$n_w = \left\lceil \frac{KR^*}{R-R^*} \right\rceil \rightarrow R^* = \left\lceil \frac{n_w}{K+n_w} \times R \right\rceil \tag{10}$$

Therefore, when estimating the increase in reliability it is necessary to consider the coverage of both the power failures and R^* network failure domain failures.

B. Placement Algorithm (DieHard)

In this section, we introduce our failure-aware placement algorithm, which we call DieHard (DH). To identify a suitable placement given the failure of (i, j) components, we must ensure that at least K jobs are running on each combination of failure domains. One intuitive approach is to evenly assign n tasks to each domain. Assuming that placement is done over network failure domains, this results in at most $Rn \geq K+n$ tasks in total given the capacity constraints. Next, we can remove $(Rn)-K-n$ tasks from the assigned tasks. Since n is derived independently from the failure probabilities of each failure domain, a lower n can be obtained by removing the tasks from the failure domains with highest probability of failure. We continue reducing the number of tasks from the least reliable failure domains as long as the reliability condition $S(\mathbf{x}) \geq S_{min}$ holds.

If the placement is to be done over power failure domains, we first assigns tasks to each power failure domain and then distribute the tasks over the network failure domains within the power failure domain to cover the failures of network failure domains.

IV. Experimental Setup

We implemented a Java-based simulator to simulate a cluster with 3 power nodes and 9600 machines connected through 120 Top of Rack (ToR) switches. Depending on the types of machines, a rack can contain between 10 and 80 machines, and each power node can support between 20 and 60 racks [10]. For our simulation, we chose racks with 80 machines and assigned 4800 machines, (60 racks) to each PDU. Each machine has 4 CPU cores and 16 GB of memory, and a background load that is a random value uniformly chosen within the available capacity range. Each ToR switch is backed up by a redundant switch, as shown in Figure 1. In addition, half of the machines are connected to two PDUs for a more resilient power supply. The cluster's machines are organized into 61 network failure domains and 4 power failure domains.

A job arrives at the system and runs for T hours. The value for the T is introduced in the experiments. All tasks have identical CPU and memory demands, uniformly chosen from 1 to 4 cores for the CPU and 1 to 16 GB for memory. The target reliability for the job was set to 0.999 or 0.9999, depending on the experiment.

The results presented below are average values obtained from 10 separate runs of each experiment.

V. Illustration and evaluation

We have already presented an analytical proof of the validity of our approach for computing the number of required extra tasks. This section therefore illustrates the impact of replica count on job reliability, for a typical use case. We also study the impact of different placement strategies on job reliability and the required number of extra tasks.

A. Impact of the number of replicas on reliability

As shown in Equation (7), it is not necessary to provide full redundancy for a job to meet the required reliability level. By exploiting this property of the reliability model, it is possible to reduce the complexity of the computations and facilitate the estimation of the required number of extra tasks. To illustrate this point, we studied the impact of increasing the number of extra tasks on reliability. As shown in Table I, as n increases the improvement in job reliability becomes progressively smaller and ultimately negligible. The results also show that there is an optimal number of replicas and further increasing the number of replicas does not significantly

increase job reliability. The optimal number increases in a stepwise fashion and is related to the number of failures that can be tolerated.

TABLE I: Impact of number of extra tasks on job reliability, K=1000

Extra replicas (n)	Reliability	Average improvement/replica
0	0.9619771697	
16	0.9884874963	1.66×10^{-3}
33	0.9997473744	6.62×10^{-4}
51	0.9999373772	1.05×10^{-5}
70	0.9999376013	1.17×10^{-8}
89	0.9999376019	3.15×10^{-11}

B. Impact of the scheduling strategy

Given a total number of tasks $N = K + n$, there are a number of distributions that can satisfy the reliability constraint $S(\mathbf{x}) \geq S_{min}$. Any distribution is acceptable as long as it guarantees a total of K tasks running on all possible combinations of available failure domains. We compare the proposed *Diehard (DH)* algorithm to two other intuitive placement strategies. We observe that, for high reliability targets, some quite intuitive placement strategies do not necessarily satisfy the reliability constraints with the approximated number of replicas.

The three placement strategies are:

- **DH**: The algorithm initially assigns an equal number of tasks over different failure domains. It then iteratively removes tasks from the domains with the highest probabilities of failure provided that the reliability constraints hold.
- **Proportional placement**: The algorithm distributes the tasks among failure domains in proportion to their probabilities of failure.
- **Highest reliability first (HRF)**: The *HRF* algorithm ranks the domains based on their failure probability. It then places the tasks in the domains with the greatest reliability as long as that they have available capacity.

To compare the three algorithms, we use the *affinity score* [2] as a metric to measure the likelihood of correlated failures. Let $\mathbf{x} = [x_1, ..., x_l]$ be the distribution of tasks over different failure domains, where $x_1 \leq x_2 \leq ... \leq x_l$. Let the impact of the correlated failure be the number of tasks sharing a common failure source. The affinity score is:

$$\sum_{i=1}^{l} \frac{x_i(x_i - 1)}{2} \tag{11}$$

The affinity score is maximized when all the tasks (task failures) are in the same domain, and minimized when tasks are spread over different domains. A low affinity score represents a low concentration and a low risk of correlated failure.

Table II compares the required numbers of extra tasks, reliabilities, and affinity scores of the three placement algorithms. For high reliability targets, the *DH* algorithm clearly requires the fewest extra replicas to guarantee reliability. The *DH* algorithm gives the lowest affinity score, showing that it reduces the risk of correlated failures by having the lowest task concentration. Reducing task concentration increases reliability with respect to correlated failures while minimizing the number of extra tasks required to guarantee the desired reliability.

It can also be seen that, for the same reliability target (0.9999), the *HRF* algorithm has the lowest reliability and the highest affinity score. Although it may seem intuitive to place as many tasks as possible on the domain with the highest reliability, this placement strategy leads to the highest affinity score and the lowest reliability value when there is a risk of correlated failure. It can also be seen that, for high reliability targets, the *HRF* algorithm cannot even satisfy the reliability constraint with the same upper bound value n as the other two placement strategies. Thus, placing tasks using the *HRF* algorithm would substantially increase the number of extra tasks required for each job to achieve a given level of reliability. This is because the *HRF* placements yield a high level of correlation among potential failures, reducing the benefits of increasing redundancy. Replication alone is thus not sufficient to protect against correlated failures. In other words, the risk of correlated failure is not mitigated and reliability is not improved by increasing the redundancy within the failure domain.

However, as shown in Table III, if a job's desired reliability is lower than the reliability of a failure domain (in this case 0.999), deploying all the tasks on any failure domain with higher reliability that the job's target reliability satisfies the $S(\mathbf{x}) \geq S_{min}$ constraint with no extra tasks. It should also be noted that the probability of failure of a component during a job runtime is a function of the job's duration: longer jobs are more likely to have failures during their runtime. Therefore, for long running jobs with high reliability targets, distributing the tasks over different failure domains (using *DH* or *proportional* placement), is necessary to satisfy the job's reliability. However, this may not be the case for a short-running job, as it is more probable to satisfy the reliability constraint by deploying the tasks using *HRF*. The impact of job duration on required number of extra tasks and job reliability is shown in Table IV.

It can also be seen that the *proportional* placement algorithm requires more extra tasks than the *DH* algorithm. This is because although it distributes tasks proportionally over different domains, the distribution is still biased by the domains' reliability. This bias in distribution increases the concentration of the placement scheme and thus the affinity score, thereby reducing reliability. The decrease in reliability forces the system to deploy more tasks to achieve the target reliability. In other words, a minimum level of distribution is required to achieve a high reliability.

VI. Related work

Below, we review two categories of related work.

A. Failure analyses in data centers

Several studies [2], [5], [13] have focused on characterizing failure sources and analyzing their impact on system availabil-

TABLE II: Impact of placement on required number of extra tasks, reliability and affinity score, $S_{min} = 0.9999$, job duration = 80 hours.

K	Algorithm	Extra tasks (n)	Affinity score	Reliability
500	DH	18	2089	0.9999
	Proportional	23	2274	0.9999
	HRF	25	137026	**0.9995**
700	DH	32	4015	0.9999
	Proportional	33	4476	0.9999
	HRF	36	269745	**0.9996**
1000	DH	36	8280	0.9999
	Proportional	45	9228.6	0.9999
	HRF	51	550725	**0.9996**

TABLE III: Impact of placement on required number of extra tasks, reliability and affinity score, $S_{min} = 0.999$, K = 500, job duration = 80 hours.

Algorithm	Extra tasks (n)	Affinity score	Reliability
DH	9	2022	0.9995
Proportional	12	2170	0.9995
HRF	0	124251	0.9995

ity and reliability in cloud data centers. Ford et al. [2] studied the impact of correlated failures on availability for Google's cloud storage system. They argued that scheduling strategies should be aware of failure bursts caused by correlated failures. Assuming that machines fail independently results in over-estimation of the system's availability at least by two orders of magnitude. They also developed an availability model using Markov chains and introduced multi-cell replication schemes to cope with correlated failures.

Similarly, Gill et al. [13] presented an analysis of possible failures in a Microsoft cloud data center. They also studied the effectiveness of redundancy at maintaining reliability. Their observations indicated that the effectiveness of network redundancy at masking network failure is only 40%. This was attributed to the propagation of configuration errors, which can lead to concurrent failures of many tasks. Their results highlight the necessity of spreading tasks over different domains of control to achieve high reliability.

B. Failure-aware scheduling and allocation

Cirne et al. [14] discussed a task backup strategy to provide reliability guarantees for a job. The goal is to determine the probability of losing a certain number of backups and use this probability for the admission control decisions. They took into account the possibility of correlated failures of tasks caused by rack and machine faults, but only if the failure domains nicely nest into a single tree, which does not fit the network or power domain models. They also did not fully investigate the possibility of multiple rack failures during the job runtime and its impact on job reliability.

TABLE IV: Impact of job duration and placement on required number of extra tasks, reliability and affinity score, $S_{min} = 0.999$, K = 500.

Duration	Algorithm	Extra tasks	Affinity score	Reliability
80 h	DH	9	2022	0.9995
	Proportional	12	2170.6	0.9995
	HRF	0	124251	0.9995
168 h	DH	18	2058	0.9995
	Proportional	21	2247	0.9992
	HRF	25	137026	**0.998**

Bakkaloglu et al. [15] studied correlated failures in storage systems. They modeled availability using a beta-binomial distribution, which was computed by randomizing the failure probabilities according to a binomial distribution, and used a correlation factor to quantify the intensity of correlations. However, different studies [2], [4] have shown that beta-binomial distributions do not provide a good fit to real-world data from data centers. Moreover, it is still challenging to accurately estimate correlation coefficients.

Tang et al. [3] analyzed the impact of correlated failures on reliability for DEC VAX clusters, and found that such failures can reduce reliability by several orders of magnitude. They proposed a correlation coefficient-based model to quantify the relationship between failures and reliability. However, the proposed model is only applicable to two-way correlations and is not straightforwardly generalized to higher levels of correlation.

Bodik et al. [8] presented an optimization framework for achieving high fault tolerance while reducing the bandwidth consumption in the network. They improved fault tolerance by spreading applications across different failure domains. However, their framework is not designed to cope with the problem of correlated failures and does not take probabilities of failure into account during scheduling.

Rabbani et al. [16] proposed a management framework for maintaining high reliability. Their method considers the heterogeneity of components' failure rates when planning the number and allocation of redundant virtual nodes in a virtual infrastructure. However, their main focus is on independent failures and not the correlated ones. Moreover, their main objective is to minimize the number of machines required to deploy the job, at the expense of increasing the number of backups. Their procedure iteratively increases the number of required backups, until the reliability constraint is satisfied. We believe that estimating the number of extra backups without considering allocations and the probability of machine failures leads to over provisioning and is not a reasonable way to maintain reliability.

Venice [17] is a framework for achieving high reliability for a 3-tier application with VM dependencies. It has a reliability-aware scheduler that deploys VMs on the machines with the lowest reliability capable of meeting the service reliability requirement. Then, over a number of trials, it removes the machines with the lowest reliability and deploys the VMs on the remaining set of machines. Finally, the scheduler selects the allocation scheme with the lowest cost as its final solution. Sampaio [18] also considered the Mean Time Between Failures (MTBF) of the nodes when planning allocations. Both of these works ignored the impact of correlated failures on service reliability and also did not consider the benefits of using redundant replicas to attain the required service reliability.

Mills et al. [19] addressed the replica scheduling problem

using a greedy heuristic in a tree structure. The tree structure represents the dependencies among system components. The aim is to minimize the number of concurrent component failures due to a single failure event. However, structuring the component dependencies as a tree eliminates the possibility of supporting overlapping failure domains.

VII. Conclusion

In this paper, we address the problem of efficiently scheduling resources in a cloud data center to achieve reliability even in the face of correlated failures. The goal is to achieve each job's reliability while minimizing the number of extra tasks required during the job's runtime. The reliability is achieved through task replication and diversified job placement over different failure domains.

We present a reliability model that accounts for failure probabilities and the topologies of power and network components in the data center. We also provide a method for obtaining approximate reliability estimates that does not require expensive computations. We use our model to approximate the minimum number of extra tasks required to achieve a desired reliability. This is done by using a decision tree to map the target reliability to a specific redundancy level. Moreover, we introduce a scheduling algorithm to schedule tasks on resources in a way that accounts for their capacity constraints.

The results show that if a job's desired reliability is lower than the reliability of a failure domain, deploying all the tasks on failure domains with high reliability would be sufficient. However, this is usually not the case. The job's desired reliability is often higher than the reliability of the failure domains. Moreover, the actual failure rates of the devices can be unknown and thus having an estimation the reliabilities. In these scenarios, it is recommended to distribute the tasks over multiple failure domains to reduce the risks of correlated failures and reduce the number of required replicas. This is because any concentration in placement yields a high level of correlation among potential failures and reduces the benefits of increasing the redundancy. In other words, replicating a task within a failure domain cannot prevent it from being affected by correlated failures.

In future work, we plan to extend our work to support non-terminal failures, where failure durations are finite, tasks can be restarted, and failures can be recovered. We also plan to extend the model to support multiple running tasks per one single machine.

Acknowledgement

Financial support for this work was provided in part by the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control, the Swedish Government's strategic research project eSSENCE, and the European Union's Seventh Framework Programme under grant agreement 610711 (CACTOS). Special thanks to Alessandro Papadopoulos for inspirations while choosing the title.

References

[1] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM Computer Communication Review*, vol. 39, pp. 51–62, ACM, 2009.

[2] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Operating Systems Design and Implementation (OSDI)*, pp. 61–74, 2010.

[3] D. Tang and R. K. Iyer, "Analysis and modeling of correlated failures in multicomputer systems," *IEEE Transactions on Computers*, vol. 41, no. 5, pp. 567–577, 1992.

[4] P. Yalagandula, S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems," in *Proceedings of the Workshop on Real, Large Distributed Systems (WORLDS '04)*, 2004.

[5] B. Chun and A. Vahdat, "Workload and failure characterization on a large-scale federated testbed," *Intel Research Berkeley Technical Report IRB-TR-03-040*, 2003.

[6] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, pp. 193–204, ACM, 2009.

[7] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in *OSDI*, vol. 99, pp. 45–58, 1999.

[8] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 431–442, ACM, 2012.

[9] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *European Conference on Computer Systems (EuroSys)*, pp. 18–35, ACM, 2015.

[10] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 13–23, ACM, 2007.

[11] H. Geng, *Data Center Handbook, 2014*. Wiley.

[12] W. A. Trybulec, "Pigeon hole principle," *Journal of Formalized Mathematics*, vol. 2, no. 199, 1990.

[13] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 350–361, ACM, 2011.

[14] W. Cirne and E. Frachtenberg, "Web-scale job scheduling," in *Job Scheduling Strategies for Parallel Processing*, pp. 1–15, Springer, 2013.

[15] M. Bakkaloglu, J. Wylie, C. Wang, and G. Ganger, "On correlated failures in survivable storage systems." Technical Report Carnegie Mellon University, (2002), School of Computer Science Carnegie Mellon University.

[16] M. G. Rabbani, M. F. Zhani, and R. Boutaba, "On achieving high survivability in virtualized data centers," *IEICE Transactions on Communications*, vol. 97, no. 1, pp. 10–18, 2014.

[17] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proceedings IEEE INFOCOM 2014*, pp. 289–297, IEEE, 2014.

[18] A. M. Sampaio and J. G. Barbosa, "Towards high-available and energy-efficient virtual computing environments in the cloud," *Future Generation Computer Systems*, vol. 40, pp. 30–43, 2014.

[19] K. A. Mills, R. Chandrasekaran, and N. Mittal, "Algorithms for replica placement in high-availability storage," *arXiv preprint arXiv:1503.02654*, 2015.