Shadows on the Cloud: An energy-aware, profit maximizing resilience framework for cloud computing

Xiaolong Cui, Bryan Mills, Taieb Znati and Rami Melhem
Department of Computer Science, University of Pittsburgh, Pittsburgh, U.S.A [*]
{*mclarencui,bmills,znati,melhem*}*@cs.pitt.edu*

Keywords: shadow replication, fault tolerance, scheduling, resilience, energy-aware

Abstract: As the demand for cloud computing continues to increase, cloud service providers face the daunting challenge to meet the negotiated SLA agreement, in terms of reliability and timely performance, while achieving cost-effectiveness. This challenge is increasingly compounded by the increasing likelihood of failure in large-scale clouds and the rising cost of energy consumption. This paper proposes Shadow Replication, a novel profit-maximization resiliency model, which seamlessly addresses failure at scale, while minimizing energy consumption. The basic tenet of the model is to associate a suite of shadow processes to execute concurrently with the main process, but initially at a much reduced execution speed, to overcome failures as they occur. Two computationally-feasible schemes are proposed to achieve shadow replication. A performance evaluation framework is developed to analyze these schemes and compare their performance to traditional replication-based fault tolerance methods, focusing on the inherent tradeoff between fault tolerance, the specified SLA and profit maximization. The results show Shadow Replication leads to significant energy reduction, and is better suited for compute-intensive execution models, where up to 30% more profit increase can be achieved.

1 INTRODUCTION

Cloud Computing has emerged as an attractive platform for increasingly diverse compute- and data-intensive applications, as it allows for low-entry costs, on demand resource provisioning and allocation and reduced cost of maintaining internal IT infrastructure (Tchana et al., 2012). Cloud computing will continue to grow and attract attention from commercial and public market segments. Recent studies predict annual growth rate of 17.7 percent by 2016, making cloud computing the fastest growing segment in the software industry.

In its basic form, a cloud computing infrastructure is a large cluster of interconnected back-end servers hosted in a datacenter and provisioned to deliver on-demand, "pay-as-you-go" services and computing resources to customers through a front-end interface (Amazon, 2013). As the demand for cloud computing accelerates, cloud service providers (CSPs) will be faced with the need to expand their underlying infrastructure to ensure the expected levels of performance, reliability and cost-effectiveness, resulting in a multifold increase in the number of computing, storage and communication components in their datacenters. The direct implication of large datacenters is increased management complexity and propensity to failure. While the likelihood of a server failure is very small, the sheer number of computing, storage and communications components that can fail, however, is daunting. At such a large scale, failure becomes the norm rather than an exception (Schroeder and Gibson, 2010).

As the number of users delegating their computing tasks to CSPs increases, Service Level Agreements (SLAs) become a critical aspect for a sustainable cloud computing business model. In its basic form, an SLA is a contract between the CSPs and consumers that specifies the terms and conditions under which the service is to be provided, including expected response time and reliability. Failure to deliver the service as specified in the SLA subjects the CSP to pay a penalty, resulting in a loss of revenue.

In addition to penalties resulting from failure to meet the SLA requirement, CSPs face rising energy costs of their large-scale datacenters. It is reported

[*]This research is based in part upon work supported by the National Science Foundation under Grant Number CNS12-53218 and CNS12-52306. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

that energy costs alone could account for 23-50% of the expenses (Elnozahy et al., 2003) and this bill mounts up to $30 billion worldwide (Raghavendra et al., 2008). This raises the question of how fault tolerance might impact power consumption and ultimately the expected profit of the service providers.

Current fault tolerance approaches rely upon either time or hardware redundancy in order to tolerate failure. The first approach, which uses time redundancy, requires the re-execution of the failed task after the failure is detected. Although this can further be optimized by the use of checkpointing and rollback recovery, such an approach can result in a significant delay increase subjecting CSPs to penalties, when SLA terms are violated, and high energy costs due to re-execution of failing tasks.

The second approach exploits hardware redundancy and executes multiple instances of the same task in parallel to overcome failure and guarantee that at least one task reaches completion. This approach, which has been used extensively to deal with failure in critical applications, is currently used in cloud-computing to provide fault tolerance while hiding the delay of re-execution (Tsai et al., 2011; Ko et al., 2010). This solution, however, increases the energy consumption for a given service, which in turn might outweigh the profit gained by providing the service. The trade-off between profit and fault-tolerance calls for new frameworks to take both SLA requirements and energy awareness in dealing with failures.

In this paper, we address the above trade-off challenge and propose an energy-aware, SLA-based profit maximization framework, referred to as "Shadow Replication", for resilience in cloud computing. Similar to traditional replication, Shadow Replication ensures successful task completion by concurrently running multiple instances of the same task. Contrary to traditional replication, however, Shadow Replication executes the main instance of the task at the speed required to maximize profit and uses dynamic voltage and frequency scaling (DVFS) to slow down the execution of the replicas, thereby enabling a parameterized trade-off between response time, energy consumption and hardware redundancy. This allows CSPs to maximize the expected profit by accounting for income, potential penalties and energy cost.

The main challenge of Shadow Replication resides in determining jointly the execution speeds of all task instances, both before and after a failure occurs, with the objective to minimize energy and maximize profit. In this paper, we propose a reward-based analytical framework to achieve this objective. The main contributions of this paper are as follows:

- An energy-aware, SLA-based, profit maximization execution model, referred to as "Shadow Replication", for resilient cloud computing.
- A profit-based optimization model to explore the applicability of Shadow Replication to cloud computing, and to determine the optimal speeds of all task instances to maximize profit.
- In environments where either the specification or the detection of failure is hard to achieve, we propose a sub-optimal, yet practical resilience scheme, called profit-aware stretched replication.
- An evaluation framework to analyze profit and energy savings achievable by Shadow Replication, compared to existing resilience methods.

The analysis shows that in all cases, Shadow Replication outperforms existing fault tolerance methods. Furthermore, shadow replication would converge to traditional replication, when target response time is stringent, and to re-execution when target response time is relaxed or failure is unlikely, as expected.

The rest of the paper is organized as follows. We begin by describing a computing model typically used in cloud computing for compute- and data-intensive applications in Section 2. We then introduce the Shadow Replication framework in Section 3. Section 4, 5, and 6 present our analytical models and optimization problem formalization, followed by experiments and evaluation in section 7. Section 8 briefly surveys related work. Section 9 concludes this work.

2 CLOUD WORKLOAD CHARACTERIZATION

Cloud computing workload ranges from business applications and intelligence, to analytics and social networks mining and log analysis, to scientific applications in various fields of sciences and discovery. These applications exhibit different behaviors, in term of computation requirements and data access patterns. While some applications are compute-intensive, others involve the processing of increasingly large amounts of data. The scope and scale of these applications are such that an instance of a job running one of these applications requires the sequential execution of multiple computing phases; each phase consists of thousands, if not millions, of tasks scheduled to execute in parallel and involves the processing of a very large amount of data (Lin and Dyer, 2010; Ferdman and et. al., 2012). This model is directly reflective of the *MapReduce* computational model, which is predominately used in Cloud Computing (Sangroya et al., 2012). An instance of this model, is depicted in Figure 1.

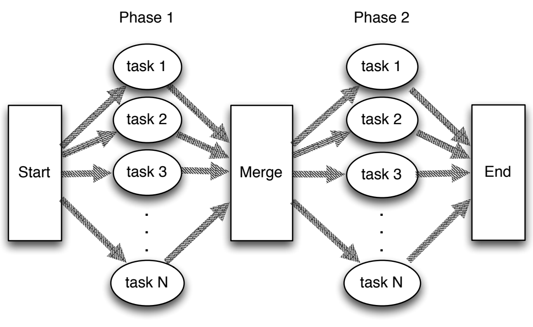


Figure 1: Cloud computing execution model with 2 phases.

Each job has a targeted response time defined by the terms of the SLA. Further, the SLA defines the amount to be paid for completing the job by the targeted response time as well as the penalty to be incurred if the targeted response time is not met.

Each task is mapped to one compute core and executes at a speed, σ. The partition of the job among tasks is such that each task processes a similar workload, W. Consequently, baring failures, tasks are expected to complete at about the same time. Therefore, the minimal response time of each task, when no failure occurs, is $t_{min} = \frac{W}{\sigma_{max}}$, where σ_{max} is the maximum speed. This is also the minimal response time of the entire phase.

As the number of tasks increases, however, the likelihood of a task failure during an execution of a given phase increases accordingly. This underscores the importance of an energy-efficient fault-tolerance model to mitigate the impact of a failing task on the overall delay of the execution phase. The following section describes Shadow Replication, a fault-tolerant, energy-aware computational model to achieve profit-maximizing, energy-efficient resiliency in cloud computing.

3 SHADOW REPLICATION

The basic tenet of Shadow Replication is to associate with each main process a suite of “shadows” whose size depends on the “criticality” of the application and its performance requirements, as defined by the SLA.

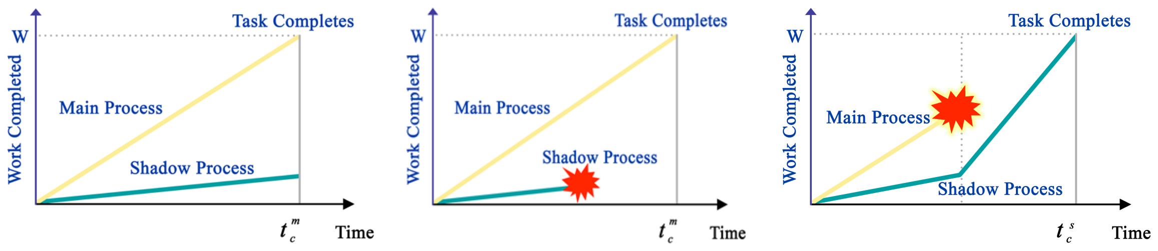
Formally, we define the Shadow Replication fault-tolerance model as follows:

- A main process, $P_m(W,\ \sigma_m)$, whose responsibility is to executes a task of size W at a speed of σ_m;
- A suite of shadow processes, $P_s(W,\ \sigma_b^s,\ \sigma_a^s)$ $(1 \leq s \leq \mathcal{S})$, where $\mathcal{S}$ is the size of the suite. The shadows execute on separate computing nodes. Each shadow process is associated with two execution speeds. All shadows start execution simultaneously with the main process at speed σ_b^s $(1 \leq s \leq \mathcal{S})$. Upon failure of the main process, all shadows switch their executions to σ_a^s, with one shadow being designated as the new main process. This process continues until completion of the task.

To illustrate the behavior of Shadow Replication, we limit the number of shadows to a single process and consider the scenarios depicted in Figure 2, assuming a single process failure. Figure 2(a) represents the case when neither the main nor the shadow fails. The main process, executing at a higher speed, completes the task at time t_c^m. At this time, the shadow process, progressing at a lower speed, stops execution immediately. Figure 2(b) represents the case when the shadow fails. This failure, however, has no impact on the progress of the main process, which still completes the task at t_c^m. Figure 2(c) depicts the case when the main process fails while the shadow is in progress. After detecting the failure of the main process, the shadow begins execution at a higher speed, completing the task at time t_c^s. When possible, the shadow execution speed upon failure must be set so that t_c^s does not exceed t_c^m. Given that the failure rate of an individual node is much lower than the aggregate system failure, it is very likely that the main process will always complete its execution successfully, thereby achieving fault tolerance at a significantly reduced cost of energy consumed by the shadow.

A closer look at the model reveals that shadow replication is a generalization of traditional fault tolerance techniques, namely re-execution and traditional replication. If the SLA specification allows for flexible completion time, shadow replication would take advantage of the delay laxity to trade time redundancy for energy savings. It is clear, therefore, that for a large response time, Shadow Replication converges to re-execution, as the shadow remains idle during the execution of the main process and only starts execution upon failure. If the target response time is stringent, however, Shadow Replication converges to pure replication, as the shadow must execute simultaneously with the main at the same speed. The flexibility of the Shadow Replication model provides the basis for the design of a fault tolerance strategy that strikes a balance between task completion time and energy saving, thereby maximizing profit.

Given that the probability of two individual nodes executing the same instances of a task fail at the same



(a) No Failure (b) Shadow Process Failure (c) Main Process Failure

Figure 2: Shadow replication for a single task and single replica

time is low, we will focus on the study of Shadow Replication model with a single shadow. It is clear, however, that the model can be extended to support multiple processes, as required by the application's fault-tolerance requirement. Furthermore, we adopt the fail-stop fault model, where a processor stops execution once a fault occurs and failure can be detected by other processes(Gärtner, 1999; Cristian, 1991).

4 REWARD BASED OPTIMAL SHADOW REPLICATION

In this section, we describe a profit-based optimization framework for the cloud-computing execution model previous described. Using this framework we compute profit-optimized execution speeds by optimizing the following objective function:

$$\begin{aligned} \max_{\sigma_m,\sigma_b,\sigma_a} & E[profit] \\ s.t. 0 \leq & \sigma_m \leq \sigma_{max} \\ 0 \leq & \sigma_b \leq \sigma_m \\ 0 \leq & \sigma_a \leq \sigma_{max} \end{aligned} \tag{1}$$

We assume that processor speeds are continuous and use nonlinear optimization techniques to solve the above optimization problem.

In order to earn profit, service providers must either increase income or decrease expenditure. We take both factors into consideration for the purpose of maximizing profit while meeting customer's requirements. In our model, we set the expected profit to be expected income minus expected expense.

$$E[\text{profit}] = E[\text{income}] - E[\text{expense}] \tag{2}$$

4.1 Reward Model

The cloud computing SLA can be diverse and complex. To focus on the profit and reliability aspects of the SLA, we define the reward model based on job completion time. Platform as a Service (PaaS) companies will continue to become more popular causing an increase in SLAs using job completion time as their performance metric. We are already seeing this appear in web-based remote procedure calls and data analytic requests.

As depicted in Figure 3, customers expect that their job deployed on cloud finishes by a mean response time t_{R_1}. As a return, the provider earns a certain amount of reward, denoted by R, for satisfying customer's requirements. However, if the job cannot be completed by the expected response time, the provider loses a fraction of R proportional to the delay incurred. For large delay, the profit loss may translate into a penalty that the CSP must pay to the customer. In this model, the maximum penalty P is paid if the delay reaches or exceeds t_{R_2}. The four parameters, R, P, t_{R_1} and t_{R_2}, completely define the reward model.

There are two facts that the service provider must take into account when negotiating the terms of the SLA. The first is the response time of the main process assuming no failure (Figure 2(a) and Figure 2(b)). This results in the following completion time:

$$t_c^m = W/\sigma_m \tag{3}$$

If the main process fails (shown in Figure 2(c)), the task completion time by shadow process is the time of the failure, t_f, plus the time necessary to complete the remaining work.

$$t_c^s = t_f + \frac{W - t_f \times \sigma_b}{\sigma_a} \tag{4}$$

This reward model is flexible and extensible; it is not restricted to the form shown in Figure 3. In particular, the decrease may be linear, concave, or convex and the penalty can extend to infinity. This model can further be extended to take into consideration both the short-term income and long-term reputation of the service provider (Daw and Touretzky, 2002).

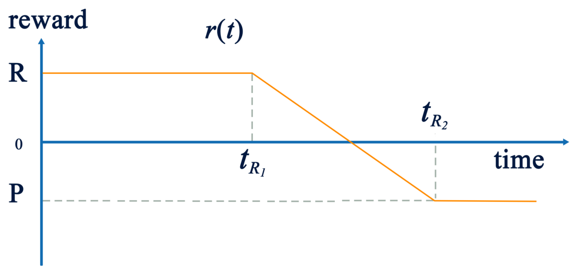


Figure 3: A reward function

4.2 Failure Model

Failure can occur at any point during the execution of the main or shadow process. Our assumption is that at most one failure occurs, therefore if the main process fails it is implied that the shadow will complete the task without failure. We can make this assumption because we know the failure of any one node is rare thus the failure of any two specific nodes is very unlikely.

We assume that two probability density functions, $f_m(t_f)$ and $f_s(t_f)$, exist which express the probabilities of the main and shadow process failing at time t_f separately. The model does not assume a specific distribution. However, in the remainder of this paper we use an exponential probability density function, $f_m(t_f) = f_s(t_f) = \lambda e^{-\lambda t_f}$, of which the mean time between failure (MTBF) is $\frac{1}{\lambda}$.

4.3 Power and Energy Models

Dynamic voltage and frequency scaling (DVFS) has been widely exploited as a technique to reduce CPU dynamic power (Flautner et al., 2002; Pillai and Shin, 2001). It is well known that one can reduce the dynamic CPU power consumption at least quadratically by reducing the execution speed linearly. The dynamic CPU power consumption of a computing node executing at speed σ is given by the function $p_d(\sigma) = \sigma^n$ where $n \geq 2$.

In addition to the dynamic power, CPU leakage and other components (memory, disk, network etc.) all contribute to static power consumption, which is independent of the CPU speed. In this paper we define static power as a fixed fraction of the node power consumed when executing at maximum speed, referred to as ρ. Hence node power consumption is expressed as $p(\sigma) = \rho \times \sigma_{max}^n + (1-\rho) \times \sigma^n$. When the execution speed is zero the machine is in a sleep state, powered off or not assigned as a resource; therefore it will not be consuming any power, static or dynamic. Throughout this paper we assume that dynamic power is cubic in relation to speed (Rusu et al., 2003; Zhai et al., 2004), therefore the overall system power when executing at speed σ is defined as:

$$p(\sigma) = \begin{cases} \rho\sigma_{max}^3 + (1-\rho)\sigma^3 & \text{if } \sigma > 0 \\ 0 & \text{if } \sigma = 0 \end{cases} \tag{5}$$

Using the power model given by Equation 5, the energy consumed by a process executing at speed σ during an interval T is given by

$$E(\sigma, T) = p(\sigma) \times T \tag{6}$$

Corresponding to Figure 2, there are three failure cases to consider: main and shadow both succeed, shadow fails and main fails. As described earlier, the case of both the main and shadow failing is very rare and will be ignored. The expected energy consumption for a single task is then the weighted average of the expected energy consumption in the three cases.

First consider the case where no failure occurs and the main process successfully completes the task at time t_c^m, corresponding to Figure 2(a).

$$\begin{aligned} E_1 =& (1 - \int_0^{t_c^m} f_m(t)dt) \times (1 - \int_0^{t_c^m} f_s(t)dt) \times \\ & (E(\sigma_m, t_c^m) + E(\sigma_b, t_c^m)) \end{aligned} \tag{7}$$

The first line is the probability of fault-free execution of the main process and shadow process. Then we multiple this probablity by the energy consumed by the main and the shadow process during this fault free execution, ending at t_c^m.

Next, consider the case where the shadow process fails at some point before the main process successfully completes the task, corresponding to Figure 2(b).

$$\begin{aligned} E_2 =& (1 - \int_0^{t_c^m} f_m(t)dt) \times \\ & \int_0^{t_c^m} (E(\sigma_m, t_c^m) + E(\sigma_b, t)) \times f_s(t)dt \end{aligned} \tag{8}$$

The first factor is the probability that the main process does not fail, and the probability of shadow fails is included in the second factor which also contains the energy consumption since it depends on the shadow failure time. Energy consumption comes from the main process until the completion of the task, and the shadow process before its failure.

The one remaining case to consider is when the main process fails and the shadow process must continue to process until the task completes, corresponding to Figure 2(c).

$$\begin{aligned} E_3 =& (1 - \int_0^{t_c^m} f_s(t)dt) \times \int_0^{t_c^m} (E(\sigma_m, t) + \\ & E(\sigma_b, t) + E(\sigma_a, t_c^s - t)) f_m(t)dt \end{aligned} \tag{9}$$

Similarly, the first factor expresses the probability that the shadow process does not fail. In this case, the shadow process executes from the beginning to t_c^s when it completes the task. However, under our "at most one failure" assumption, the period during which shadow process may fail ends at t_c^m, since the only reason why shadow process is still in execution after t_c^m is that main process has already failed. There are three parts of energy consumption, including that of main process before main's failure, that of shadow process before main's failure, and that of shadow process after main's failure, all of which depend on the failure occurrence time.

The three equations above describe the expected energy consumption by a pair of main and shadow processes for completing a task under different situations. However, under our system model it might be the case that those processes that finish early will wait idly and consume static power if failure delays one task. If it is the case that processes must wait for all tasks to complete, then this energy needs to be accounted for in our model. The probability of this is the probability that at least one main process fails, referred to as the system level failure probability.

$$P_f = 1 - (1 - \int_0^{t_c^m} f_m(t)dt)^N \tag{10}$$

Hence, we have the fourth equation corresponding to the energy consumed while waiting in idle.

$$\begin{aligned} E_4 =& (1 - \int_0^{t_c^m} f_m(t)dt) \times (1 - \int_0^{t_c^m} f_s(t)dt) \times \\ & 2P_f \times E(0, t_c^j - t_c^m) + \int_0^{t_c^m} f_s(t)dt \times \\ & (1 - \int_0^{t_c^m} f_m(t)dt) \times P_f \times E(0, t_c^j - t_c^m) \end{aligned} \tag{11}$$

Corresponding to the first case, neither main process nor shadow process fails, but both of them have to wait in idle from task completion time t_c^m to the last task's completion (by a shadow process) with probability P_f. Under the second case, only the main process has to wait if some other task is delayed since its shadow process has already failed. These two aspects are accounted in the first and last two lines in E_4 separately. We use the expected shadow completion time t_c^j as an approximation of the latest task completion time which is also the job completion time.

By summing these four parts and then multiplying it by N we will have the expected energy consumed by Shadow Replication for completing a job of N tasks.

$$E[\text{energy}] = N \times (E_1 + E_2 + E_3 + E_4) \tag{12}$$

4.4 Income and Expense Models

The income is the reward paid by customer for the cloud computing services that they utilize. It depends on the reward function $r(t)$, depicted in Figure 3, and the actual job completion time. Therefore, the income should be either $r(t_c^m)$, if all main processes can complete without failure, or $r^*(t_c^s)$ otherwise. It is worth noting that the reward in case of failure should be calculated based on the last completed task, which we approximate by calculating the expected time of completion allowing us to derive the expected reward, i.e. $r^*(t_c^s) = \frac{\int_0^{t_c^m} r(t_c^s) \times f_m(t)dt}{\int_0^{t_c^m} f_m(t)dt}$. Therefore the income is estimated by the following equation.

$$E[\text{income}] = (1 - P_f) \times r(t_c^m) + P_f \times r^*(t_c^s) \tag{13}$$

The first part is the reward earned by the main process times the probability that all main processes would complete tasks without failure. If at least one main process fails, that task would have to be completed by a shadow process. As a result, the second part is the reward earned by shadow process times the system level failure probability.

If C is the charge expressed as dollars per unit of energy consumption (e.g. kilowatt hour), then the expected expenditure would be C times the expected energy consumption for all N tasks:

$$E[\text{expense}] = C \times E[\text{energy}] \tag{14}$$

However, the expenditure of running the cloud computing service is more than just energy, and must includes hardware, maintenance, and human labor. These costs can be accounted for by amortizing these costs into the static power factor, ρ. Because previous studies have suggested (Elnozahy et al., 2003; Raghavendra et al., 2008) that energy will become a dominate factor we decided to focus on this challenge and leave other aspects to future work.

Table 1: Symbols used in our analytical model.

Symbols	Definition
W	Task size
N	Number of tasks
$r(t)$	Reward function
R, P	Maximum reward and penalty
t_{R_1}, t_{R_2}	Response time thresholds
C	Unit price of energy
ρ	Static power ratio
t_c^m, t_c^s, t_c^j	Completion time of main process, shadow process, and the whole job
$f_m()$, $f_s()$	Failure density function of main and shadow
λ	Failure rate
P_f	System level failure probability
σ_m, σ_b, σ_a	Speeds of main, shadow before and after failure (Optimization Outputs)

Based on the above formalization of the optimization problem, the MATLAB Optimization Toolbox (MathWorks, 2013) was used to solve the resulting nonlinear optimization problem. The parameters of this problem are listed in Table 1.

5 PROFIT-AWARE STRETCHED REPLICATION

We compare Shadow Replication to two other replication techniques, traditional replication and profit-aware stretched replication. Traditional replication requires that the two processes always execute at the same speed σ_{max}. Unlike traditional replication Shadow Replication is dependent upon failure detection, enabling the replica to increase its execution speed upon failure and maintain the targeted response time thus maximizing profit. While this is the case in many computing environments, there are cases where failure detection may not be possible. To address this limitation, we propose profit-aware stretched replication, whereby both the main process and the shadow execute independently at stretched speeds to meet the expected response time, without the need for the main processes failure detection. In profit-aware stretched replication both the main and shadow execute at speed σ_r, found by optimizing the profit model. For both traditional replication and stretched replication, the task completion time is independent of failure and can be directly calculated as:

$$t_c = \frac{W}{\sigma_{max}} \text{ or } t_c = \frac{W}{\sigma_r} \tag{15}$$

Since all tasks will have the same completion time, the job completion time would also be t_c. Further, the expected income, which depends on negotiated reward function and job completion time, is independent of failure:

$$E[income] = r(t_c) \tag{16}$$

Since both traditional replication and profit-aware stretched replication are special cases of our Shadow Replication paradigm where $\sigma_m = \sigma_b = \sigma_a = \sigma_{max}$ or $\sigma_m = \sigma_b = \sigma_a = \sigma_r$ respectively, we can easily derive the expected energy consumption using Equation 12 with E_4 fixed at 0 and then compute the expected expense using Equation 14.

6 RE-EXECUTION

Contrary to replication, re-execution initially assigns a single process for the execution of a task. If the original task fails, the process is re-executed. In the cloud computing execution framework this is equivalent to a checkpoint/restart, the checkpoint is implicitly taken at the end of each phase and because the tasks are loosely coupled they can restart independently.

Based on the one failure assumption, two cases must be considered to calculate the task completion time. If no failure occurs, the task completion time is:

$$t_c = \frac{W}{\sigma_{max}} \tag{17}$$

In case of failure, however, the completion time is equal to the sum of the time elapsed until failure and the time needed for re-execution. Again, we use the expected value $t_f^* = \frac{\int_0^{t_c} t \times f_m(t)dt}{\int_0^{t_c} f_m(t)dt}$ to approximate the time that successfully completed processes have to spend waiting for the last one.

Similar to Shadow Replication, the income for re-execution is the weighted average of the two cases:

$$E[\text{income}] = (1 - P_f) \times r(t_c) + P_f \times r(t_c + t_f^*) \tag{18}$$

For one task, if no failure occurs then the expected energy consumption can be calculated as

$$E_5 = (1 - \int_0^{t_c} f_m(t)dt) \times (E(\sigma_{max}, t_c) + P_f \times E(0, t_f^*)) \tag{19}$$

If failure occurs, however, the expected energy consumption can be calculated as

$$E_6 = \int_0^{t_c} (E(\sigma_{max}, t) + E(\sigma_{max}, t_c)) \times f_m(t)dt \tag{20}$$

Therefore, the expected energy consumption by re-execution for completing a job of N tasks is

$$E[energy] = N \times (E_5 + E_6) \tag{21}$$

7 EVALUATION

This section evaluates the expected profit of each of the fault tolerance methods discussed above under different system environment. We have identified 5 important parameters which affect the expected profit:

- Static power ratio ρ, which determines the portion of power that is unaffected by the execution speed.
- SLA - The amount of reward, penalty and the required response times.
- N - The total number of tasks.
- MTBF - The reliability of an individual node.
- Workload - The size, W, of each individual task.

Without loss of generality, we normalize σ_{max} to be 1, so that all the speeds can be expressed as a fraction of maximum speed. Accordingly, the task workload W is also adjusted such that it is equal to the amount of time (in hours) required for a single task, preserving the ratios expressed in Equation 3 and Equation 4. The price of energy is assumed to be 1 unit. We assume that R in our reward model is linearly proportional to the number of tasks N and the maximal reward for one task is 3 units, so the total reward for a job is $3 \times N$ units. However, for the analysis we look at the average of expenditure and income on each task by dividing the total expenditure and income by N. In our basic configuration we assume that the static power ratio is 0.5, the task size is 1 hour, the node MTBF 5 is years, the number of tasks is 100000, and the response time thresholds for maximal and minimal rewards are 1.3 hours and 2.6 hours respectively. Since the maximal power consumption is 1 unit, the energy needed for the task with one process at maximal speed is also 1 unit.

7.1 Sensitivity to static power

With various architectures and organizations, servers deployed at different data centers will have different characteristics in terms of power consumption. The static power ratio is used to abstract the amount of static power consumed versus dynamic power.

Table 2: Speeds for different static power ratio. MTBF=5 years, N=100000, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

ρ	σ_m	σ_b	σ_a
0.0	0.77	0.65	1.00
0.1	0.78	0.66	1.00
0.2	0.83	0.66	1.00
0.3	0.84	0.68	1.00
0.4	0.85	0.70	1.00
0.5	0.86	0.72	1.00
0.6	0.87	0.73	1.00
0.7	0.91	0.81	1.00
0.8	1.00	1.00	1.00
0.9	1.00	1.00	1.00
1.0	1.00	1.00	1.00

Table 2 shows how the profit-optimized execution speeds of Shadow Replication will change as static power increases. The execution speeds increase to reduce the execution time as static power ration increases. Observe that σ_a is always equal to σ_{max}, which means that after sensing the failure of the main process, the shadow process should always shift to maximum speed. This is expected because the optimization will reduce the amount of work done by the shadow process before failure resulting in the maximum execution speed after failure, thus minimizing the amount of repeated work.

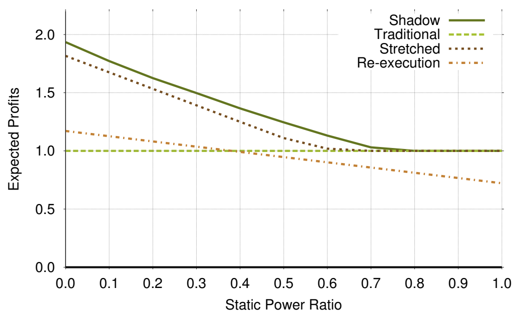


Figure 4: Profit for different static power ratio. MTBF=5 years, N=100000, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

The potential profit gains achievable by using profit-aware replication techniques decreases as static power increases, as is shown in Figure 4. The reason is that our profit-aware techniques rely upon the fact that one can reduce energy costs by adjusting the execution speeds. Modern systems have a static power between 40%-70% and it is reasonable to suspect that this will continue to be the case. Within this target range of static power, Shadow Replication can achieve, on average, 19.3% more profit than traditional replication, 8.9% more than profit-aware stretched replication, and 28.8% more than reexeuction.

7.2 Sensitivity to response time

Response time is critical in the negotiation of SLA as customers always expect their tasks to complete as soon as possible. In this section we show a sensitivity study with respect to task response time. We vary the first threshold t_{R_1} from the minimal response time t_{min} to $1.9t_{min}$, and set the second threshold t_{R_2} to be always $2t_{R_1}$. We do not show results for varying the reward and penalty values of the SLA. The reason is that changing these values have no effect on the choice of fault tolerance methods because they are all affected in a similar way.

Table 3: Speeds for different response time threshold. ρ=0.5, MTBF=5 years, N=100000, W=1 hour.

t_{R_1}	σ_m	σ_b	σ_a
1.0	1.00	1.00	1.00
1.1	0.94	0.88	1.00
1.2	0.89	0.79	1.00
1.3	0.86	0.72	1.00
1.4	1.00	0.00	1.00
1.5	1.00	0.00	1.00
1.6	0.84	0.00	1.00
1.7	0.74	0.00	1.00
1.8	0.64	0.00	1.00
1.9	0.64	0.00	1.00

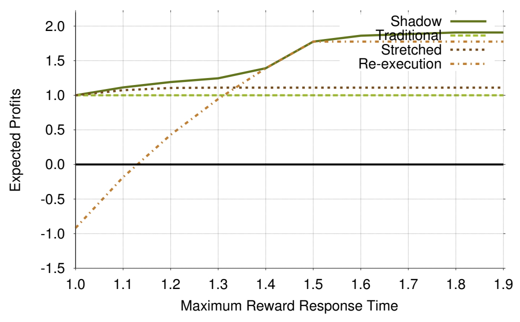


Figure 5: Profit for different response time threshold. ρ=0.5, MTBF=5 years, N=100000, W=1 hour.

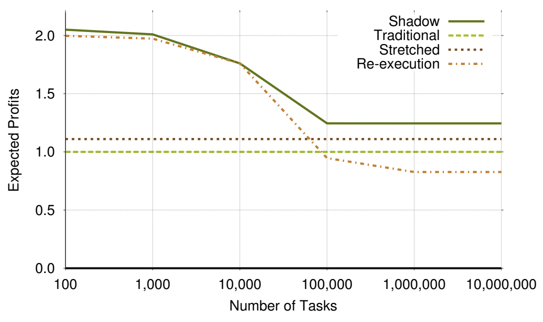


Figure 6: Profit for different number of tasks. ρ=0.5, MTBF=5 years, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

In Table 3 we see that Shadow Replication adapts the execution speeds to take advantage of the available laxity, reducing its speeds as laxity increases. It is clear that Shadow Replication has two different execution strategies separated by $t_{R_1} = 1.4$: when time is critical, it uses both a main and a shadow from the very beginning to guarantee that task can be completed on time; when time is not critical, it mimics re-execution and starts its shadow only after a failure. Also note that as t_{R_1} approaches t_{min}, the speeds of the main process and the shadow process converge, effectively causing Shadow Replication to mimic traditional replication when faced with time-critical jobs.

Figure 5 shows the effect that targeted response time has upon the profitability of each fault tolerance method. Compared to traditional replication, all the other methods increase their profit as the targeted response time increases, this is expected because each of the other techniques can make use of increased laxity in time to increase profit. Re-execution is the most sensitive to the target response time since it fully relies upon time redundancy, showing that it should only be used when the targeted response time is *not* stringent. Again, Shadow Replication always achieves more profit than traditional replication and profit-aware stretched replication, and the profit gains are 52.8% and 39.0% on average.

7.3 Sensitivity to number of tasks

Table 4: Speeds for different number of tasks. ρ=0.5, MTBF=5 years, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

N	σ_m	σ_b	σ_a
100	0.80	0.00	1
1000	0.84	0.00	1
10000	1.00	0.00	1
100000	0.86	0.72	1
1000000	0.86	0.72	1
10000000	0.86	0.72	1

The number of tasks has a direct influence upon the system level failure probability because as the number of tasks increase the probability that failure will occur to at least one task increases. Recall that even one failure can hurt the total income significantly, and keep the other processes waiting. Thus, shadow replication will adjust its execution speeds to reduce the waiting time.

Table 4 is similar to Table 3 in that there are also two execution strategies. When there are few parallel tasks, shadow replication chooses to execute the main processes at nearly full speed and keeps the shadow processes dormant. The reason is that it is very likely that all main processes can finish their tasks successfully, and the need for redundancy is thus less significant. The other case is when there is a huge number of tasks to execute, the shadow process would keep running at a slower speed than the main to protect the main as well as save energy. Since the system level failure probability is already 0.9 when N is 100000, the speeds stay the same when $N \geq 100000$.

Figure 6 confirms that for small number of tasks re-execution is more profitable than replication. However, re-execution is not scalable as its profit decreases rapidly after N reaches 10000. At the same time, traditional replication and profit-aware stretched replication are not affected by the number of tasks because neither are affected by the system level failure rate. On average, Shadow Replication achieves 43.5%, 59.3%, and 18.4% more profits than profit-aware stretched replication, traditional replication and re-exeuction, respectively.

7.4 Sensitivity to failure vulnerability

The ratio between task size and node MTBF represents the tasks vulnerability to failure, specifically it is an approximation of the probability that failure occurs during the execution of the task. In our analysis we found that increasing task size will have the same effect as reducing node MTBF. Therefore, we analyze these together using the vulnerability to failure, allowing us to analyze a wider range of system parameters.

Table 5: Speeds for different task size over MTBF. ρ=0.5, N=100000, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

$W/MTBF$	σ_m	σ_b	σ_a
2E-10	0.79	0.00	1.00
2E-09	0.79	0.00	1.00
2E-08	0.80	0.00	1.00
2E-07	0.84	0.00	1.00
2E-06	1.00	0.00	1.00
2E-05	0.86	0.72	1.00
2E-04	0.86	0.72	1.00
2E-03	0.86	0.72	1.00

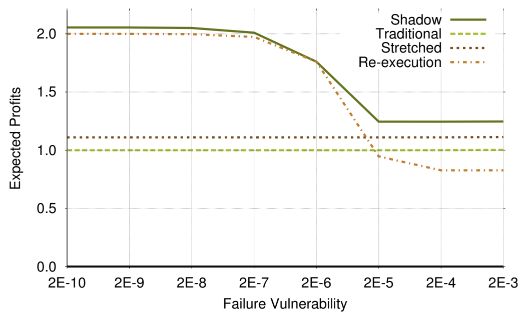


Figure 7: Profit for different task size over MTBF. ρ=0.5, N=100000, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

As seen in Table 5 when the vulnerability to failure is low the execution speeds for the shadow process is such that no work is done before failure. However, as the vulnerability increases, the shadow process performs more work before failure. This is analogous to what we observed as we increased the number of tasks (Table 4). As expected re-execution is desired when the vulnerability to failure is low. As always, Shadow Replication can adjust its execution strategy to maximize the profits, as shown in Figure 7.

7.5 Application comparison

To compare the potential benefit of "Shadow Replication" we evaluate the expected profit of each resilience technique using three different benchmark applications representing a wide range of application (Sangroya et al., 2012): Business Intelligence, Bioinformatics and Recommendation System. The business intelligence benchmark application is a decision support system for a wholesale supplier. It emphasizes executing business-oriented ad-hoc queries using Apache Hive. The bioinformatics application performs DNA sequencing, allowing genome analysis on a wide range of organisms. The recommendation system is similar to those typically found in e-commerce sites which, based upon browsing habits and history, recommends similar products.

Application	Processing Rate
Business Intelligence	3.3 (MB/s)
Bioinformatics	6.6 (MB/s)
Recommendation System	13.2 (MB/s)

Table 6: Cloud Applications (Sangroya et al., 2012)

Using the results of the experiments reported in (Sangroya et al., 2012), we derived the time required to process data for each application type (Table 6). We assume that these processing rates per task will not change when scaling the applications to future cloud environments. This is a reasonable assumption given that map-reduce tasks are loosely coupled and data are widely distributed, therefore data and task workload will scale linearly.

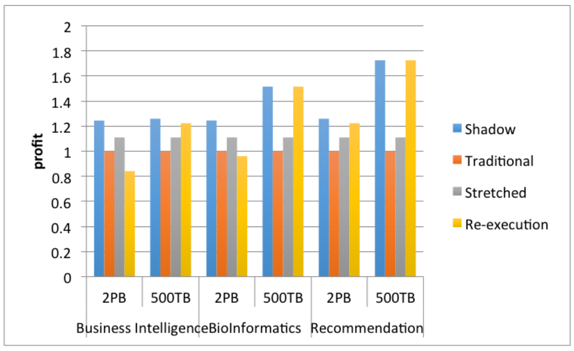


Figure 8: Application comparison. ρ=0.5, N=500000, $t_{R_1}=1.3t_{min}$, $t_{R_2}=2.6t_{min}$.

In Figure 8 we compare the expected profit for each application using each of the 4 resilience techniques. We consider two data sizes expected in future cloud computing environments, 500TB and 2PB. The figure shows that for business intelligence applications, Shadow Replication achieves significantly larger profits for both data sizes. This is because business intelligence applications tend to be IO intensive resulting in longer running tasks. Whereas recommendation systems tend to require little data IO resulting in shorter running tasks making re-execution as good as Shadow Replication. Bioinformatics tends to be in between these two applications resulting in shadow computing performing better when processing large datasets (2 PB) but not outstanding on smaller datasets (500 TB). The take away from this evaluation is that for the shown system parameters if phase execution is short, then re-execution performs as well as Shadow Replication. Alternatively, if a phase is long (20 minutes or greater), then Shadow

Replication can be as much as 47.9% more profitable than re-execution. The previous sensitivity analysis can be used to extrapolate expected profit for different system parameters.

8 RELATED WORK

The increase of failures in large-scale systems brought to the forefront the need for new fault-tolerance techniques. Coordinated checkpointing, with roll-back recovery, has been the dominant fault-tolerance method in high performance computing (HPC) (Agarwal et al., 2004; Alvisi et al., 1999; Daly, 2006; Helary et al., 1997). Based on this method, the execution state of each process is periodically saved to a stable storage. In case of failure, computation is rolled-back to the last error-free, consistent state recorded by all processes. As the scale of the system increases, the viability of coordinated checkpointing has become questionable. Despite numerous improvements of the basic coordinated checkpointing scheme, recent studies show that high failure rates, coupled with the checkpointing overhead, limit the feasibility of centralized, coordinated checkpointing (El Mehdi Diouri et al., 2012). Re-execution and state machine replication have emerged as viable schemes to deal with failures in large-scale systems.

Re-execution waits until a failure occurs and re-executes the failed process. The major shortcoming of this method stems from the large delay the completion of a job incurs. To avoid such a delay, state machine replication executes simultaneously one or more replicas of each process on different computing nodes. a (Ferreira and et. al., 2011; Sousa et al., 2005). Although it enhances fault tolerance without incurring excessive delay, state machine replication increases the computing resources needed to complete a job reliably.

Efforts have been devoted to increase the resiliency of cloud computing. Several of the proposed schemes aim at enhancing existing fault-tolerance approaches. In (Jhawar et al., 2013) the authors propose a high-level scheme that allows users to specify the desired level of resilience, while hiding implementation details. Similarly, (Zhao et al., 2010) take a middleware-based approach to fault-tolerance and propose a method that maintains strong replica consistency, while achieving transparency and low end-to-end latency. Authors in (Tchana et al., 2012) propose a collaborative solution to enhance fault-tolerance efficiency. In (Nicolae and Cappello, 2011), the authors leverage virtual disk image snapshots to minimize the storage space and checkpointing overhead. In (Zheng, 2010) investigates how redundant copies can be provisioned for tasks to improve MapReduce fault tolerance and reduce latency. Most of these schemes do not consider the impact of energy on the system.

As the significance of power and energy consumption in large datacenters increases, energy management becomes critical (Chen et al., 2012; Lin et al., 2011). Schemes are proposed to optimize power consumption by either shutting down servers, or using CPU DVFS. Our work takes a different approach to fault-tolerance, and proposes a new computational model, referred to as Shadow Replication, to achieve high-levels of resiliency, while minimizing energy consumption. In this work, we combine DVFS with traditional replication to achieve fault tolerance and maximize profit, while meeting users' SLA requirement.

9 CONCLUSION

The main motivation of this work stems from the observation that, as systems become larger and more complex, the rate of failures is highly-likely to increase significantly. Hence, understanding the interplay between fault-tolerance, energy consumption and profit maximization is critical for the viability of Cloud Computing to support future large-scale systems. To this end, we propose Shadow Replication as a novel energy-aware, reward-based computational model to achieve fault-tolerance and maximize the profit. What differentiates Shadow Replication from other methods is its ability to explore a parameterized tradeoff between hardware and time redundancy to achieve fault-tolerance, with minimum energy, while meeting SLA requirements.

To assess the performance of the proposed fault-tolerance computational model, an extensive performance evaluation study is carried out. In this study, system properties that affect the profitability of fault tolerance methods, namely failure rate, targeted response time and static power, are identified. The failure rate is affected by the number of tasks and vulnerability of the task to failure. The targeted response time represents the clients' desired job completion time, as expressed by the terms of the SLA. Our performance evaluation shows that in all cases, Shadow Replication outperforms existing fault tolerance methods. Furthermore, shadow replication will converge to traditional replication when target response time is stringent, and to re-execution when target response time is relaxed or failure is unlikely. Furthermore, the study reveals that the system static power plays a critical role in the tradeoff between the desired level of fault-tolerance, profit maximization and energy consumption. This stems from the reliance of Shadow Replication upon DVFS to reduce energy costs. If the static power is high, slowing down process execution does not lead to a significant reduction in the total energy needed to complete the task.

REFERENCES

Agarwal, S., Garg, R., and Gupta, M. S. (2004). Adaptive incremental checkpointing for massively parallel systems. In *ICS: Proc. of the* 18th *annual Int. Conf. on Supercomputing*, pages 277–286. ACM Press.

Alvisi, L., Elnozahy, E., Rao, S., Husain, S., and de Mel, A. (1999). An analysis of communication induced checkpointing. In *Fault-Tolerant Computing. 29th Annual Int. Symp. on*, pages 242–249.

Amazon (2013). Amazon elastic compute cloud. http://aws.amazon.com/ec2/.

Chen, X., Liu, X., Wang, S., and Chang, X.-W. (2012). Tailcon: Power-minimizing tail percentile control of response time in server clusters. In *Reliable Distributed Systems (SRDS), IEEE* 31st *Symp. on*, pages 61–70.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78.

Daly, J. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303 – 312.

Daw, N. D. and Touretzky, D. S. (2002). Long-term reward prediction in td models of the dopamine system. *Neural Comput.*, 14(11):2567–2583.

El Mehdi Diouri, M., Gluck, O., Lefevre, L., and Cappello, F. (2012). Energy considerations in checkpointing and fault tolerance protocols. In *Dependable Systems and Networks Workshops (DSN-W)*, pages 1–6.

Elnozahy, M., Kistler, M., and Rajamony, R. (2003). Energy conservation policies for web servers. In *Proc. of the* 4th *USENIX Symp. on Internet Tech. and Sys.*

Ferdman, M. and et. al. (2012). Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proc. of the* 17th *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, pages 37–48, New York, NY, USA. ACM.

Ferreira, K. and et. al. (2011). Evaluating the viability of process replication reliability for exascale systems. In *Proc. of Int. Conf. for HPC, Networking, Storage and Analysis*, SC, pages 44:1–44:12, New York.

Flautner, K., Reinhardt, S., and Mudge, T. (2002). Automatic performance setting for dynamic voltage scaling. *Wirel. Netw.*, 8(5):507–520.

Gärtner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26.

Helary, J.-M., Mostefaoui, A., Netzer, R., and Raynal, M. (1997). Preventing useless checkpoints in distributed computations. In *Reliable Distributed Systems. Proc., The* 16th *Symp. on*, pages 183–190.

Jhawar, R., Piuri, V., and Santambrogio, M. (2013). Fault tolerance management in cloud computing: A system-level perspective. *Systems Journal*, 7(2):288–297.

Ko, S. Y., Hoque, I., Cho, B., and Gupta, I. (2010). Making cloud intermediate data fault-tolerant. In *Proc. of the* 1st *ACM Symp. on CC*, pages 181–192. ACM.

Lin, J. and Dyer, C. (2010). Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177.

Lin, M., Wierman, A., Andrew, L., and Thereska, E. (2011). Dynamic right-sizing for power-proportional data centers. In *INFOCOM, Proc.*, pages 1098–1106.

MathWorks (2013). Matlab: Optimization toolbox.

Nicolae, B. and Cappello, F. (2011). Blobcr: efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots. In *Proc. of Int. Conf. for HPC, Networking, Storage and Analysis*, pages 34:1–34:12.

Pillai, P. and Shin, K. G. (2001). Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of the* 18th *ACM Symp. on Operating systems principles*, SOSP, pages 89–102.

Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., and Zhu, X. (2008). No "power" struggles: coordinated multi-level power management for the data center. *SIGARCH Comput. Archit. News*, 36(1):48–59.

Rusu, C., Melhem, R., and Mossé, D. (2003). Maximizing rewards for real-time applications with energy constraints. *ACM Trans. Emb. Comp. Syst.*, 2(4):537–559.

Sangroya, A., Serrano, D., and Bouchenak, S. (2012). Benchmarking dependability of mapreduce systems. In *Reliable Distributed Systems (SRDS), IEEE 31st Symp. on*, pages 21–30.

Schroeder, B. and Gibson, G. (2010). A large-scale study of failures in high-performance computing systems. *Depend. and Sec. Comp., IEEE Tran. on*, 7(4):337–350.

Sousa, P., Neves, N., and Verissimo, P. (2005). Resilient state machine replication. In *Dependable Computing. Proc.* 11th *Pacific Rim Int. Symp. on*, pages 305–309.

Tchana, A., Broto, L., and Hagimont, D. (2012). Approaches to cloud computing fault tolerance. In *Comp., Infor. & Tele. Sys., Int. Conf. on*, pages 1–6.

Tsai, W.-T., Zhong, P., Elston, J., Bai, X., and Chen, Y. (2011). Service replication strategies with mapreduce in clouds. In *Autonomous Decentralized Systems,* 10th *Int. Symp. on*, pages 381–388.

Zhai, B., Blaauw, D., Sylvester, D., and Flautner, K. (2004). Theoretical and practical limits of dynamic voltage scaling. In *Design Automation Conf. Proc.* 41st, pages 868–873.

Zhao, W., Melliar-Smith, P., and Moser, L. (2010). Fault tolerance middleware for cloud computing. In *Cloud Computing, IEEE* 3rd *Int. Conf. on*, pages 67–74.

Zheng, Q. (2010). Improving mapreduce fault tolerance in the cloud. In *Parallel Distributed Processing, Workshops and Phd Forum, IEEE Int. Symp. on*, pages 1–6.