Energies **2014**, *7*, 5151-5176; doi:10.3390/en7085151

OPEN ACCESS

energies

ISSN 1996-1073

www.mdpi.com/journal/energies

Article

Shadow Replication: An Energy-Aware, Fault-Tolerant Computational Model for Green Cloud Computing

Xiaolong Cui, Bryan Mills, Taieb Znati * and Rami Melhem

Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA;
E-Mails: mclarencui@cs.pitt.edu (X.C.); bmills@cs.pitt.edu (B.M.); melhem@cs.pitt.edu (R.M.)

* Author to whom correspondence should be addressed; E-Mail: znati@cs.pitt.edu;
Tel.: +1-412-624-8417; Fax: +1-412-624-8854.

Received: 4 June 2014; in revised form: 30 July 2014 / Accepted: 5 August 2014 /
Published: 12 August 2014

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 impact of energy consumption and CO_2 emission on the environment. This paper proposes Shadow Replication, a novel fault-tolerance model for cloud computing, which seamlessly addresses failure at scale, while minimizing energy consumption and reducing its impact on the environment. 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 that 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 due to reduced energy consumption.

Keywords: shadow computing; fault tolerance; scheduling; resilience; energy-aware

---

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 [1]. 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 [2].

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 [3]. 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 of the computing infrastructure, high-levels of energy consumption and propensity to failure. The benefits of green cloud computing are clear. As data centers are fast becoming a major source of global energy consumption, the potential savings related to energy use, CO_2 emissions and e-waste are undeniable. Achieving these savings, however, calls for new computation models designed to reduce energy and power consumption and promote environmentally friendly cloud computing execution environments.

Another challenge for cloud computing, at scale, stems from its propensity to failure. While the likelihood of a server failure is very small, the sheer number of computing, storage and communication components that can fail, however, is daunting. At such a large scale, failure becomes the norm rather than an exception [4].

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 that energy costs alone could account for 23%–50% of the expenses and this bill mounts up to $30 billion worldwide [5,6]. This raises the question of how fault tolerance might impact power consumption and ultimately impact the environment.

Current fault tolerance approaches rely upon either time or hardware redundancy in order to tolerate failures. The first approach, which uses time redundancy, requires the re-execution of the failed task after the failure is detected [7]. Although this can further be optimized by the use of checkpointing and roll-back 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 [8,9]. 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 uses Dynamic Voltage and Frequency Scaling (DVFS) to adjust the speed of the main instance so as to maximize profit and slow down the replicas in order to reduce energy-consumption, thereby enabling a parameterized trade-off between resiliency to failure, response time, and energy consumption. Furthermore, the model allows CSPs to maximize the expected profit, by accounting for income and potential penalties incurred when SLAs are not met, while reducing the impact of energy consumption on the environment.

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, green 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, while reducing the impact of energy consumption on the environment;
- 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.

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

Today's popular online services, such as web searches, video streaming, and social networks, are all powered by large datacenters. In addition to these public services, high-performance applications and

business activities are migrating onto cloud computing [10,11]. Table 1 lists several classes of cloud computing applications.

Table 1. Typical cloud computing applications.

Class	Example 1	Example 2
Data Analytics	Bioinformatics	Business Intelligence
Graph Analytics	Social Networks	Recommendation Systems
Web Search	Text Processing	Recommendation Systems

Two main characteristics of the above applications are large dataload and high parallelism. In 2008, Google processed 20 PB of data with MapReduce each day; in April 2009, a blog revealed the contents of eBay's two enormous data warehouses: one with 2 PB of data and the other with 6.5 PB of data; shortly thereafter, Facebook announced 2.5 Peta-bytes of user data, at a growth rate of at 15 Tera-bytes per day [12]. Without massive parallelism, handling such volumes of data is beyond the capability of current computing infrastructre.

Each job, that processes these data, typically requires multiple phases that execute in sequence. During each phase, workload is split into tasks and processed in parallel to speed up the whole process. Consequently, a cloud computing job can be modeled as a set of tasks, executing in parallel on different computing nodes. For the job to complete, all tasks must finish their exectution. If a task is delayed then all remaining tasks must wait, until the delayed task completes. This model is directly reflective of the *MapReduce* computational model, which is predominately used in Cloud Computing [11].

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 otherwise.

Each task is mapped to one compute core and executes at a speed, σ, in terms of CPU cycles per second. We assume that the job is evenly partitioned among tasks and each task processes the same workload, W, measured in the number of CPU cycles [13,14]. Consequently, baring failures, tasks are expected to complete at 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 resiliency in cloud computing.

3. Shadow Replication

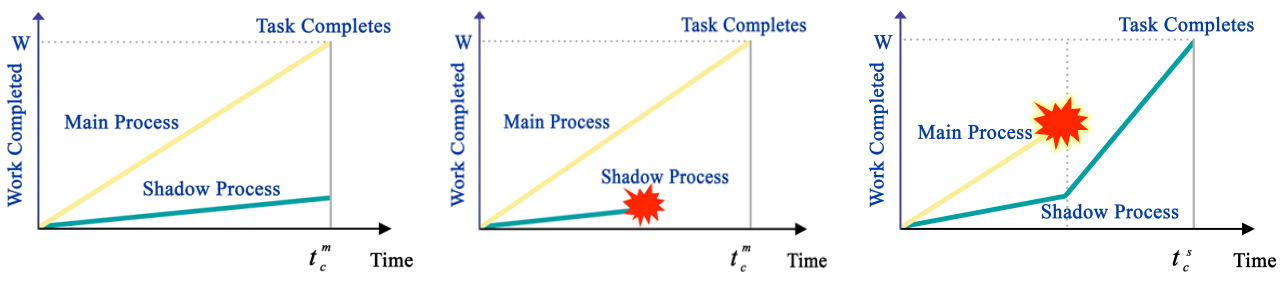
The basic tenet of Shadow Replication is to associate with each main process a suite of "shadows" whose size depends on the "criticality" of the application and its performance requirements, as defined by the SLA.

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

- A main process, $P_m(W,\ \sigma_m)$, whose responsibility is to execute a task of size W at a speed of σ_m on an exclusive computing core;
- 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 cores. Each shadow process is associated with two speeds. All shadows start execution simultaneously with the main process at speed σ_b^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 1, assuming at most one failure. Figure 1a 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 1b 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 1c 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. In the current implementations of MapReduce, such as Hadoop, there is usually a heartbeat protocol used to detect the failure of a task. The waiting time threshold of the heartbeat is tunable, so the gap between the failure occurrence of the main tasks and the change of the speed of the backup shadow tasks can be minimized [15]. In this paper, we assume that the time for failure detection is small compared to the task's execution time and thus is negligible. Given that the failure rate of an individual node is much lower than the aggregate system failure rate, 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.

Figure 1. Shadow Replication for a single task with single replica. (**a**) No Failure; (**b**) Shadow Process Failure; (**c**) Main Process Failure.



(**a**) (**b**) (**c**)

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.

As stated previously, the large amount of data being processed requires the use of thousands, if not millions, of tasks during each phase. As the number of tasks increases, the likelihood of failure during an individual phase will also increase. Fault tolerance is critical at the phase-level because the delay of one task results in a delay of the entire phase. Hence, techniques such as Shadow Replication would be applied to each phase independently. Due to the independence between phases, we focus on the fault-tolerance issues related to one computing phase of a cloud computing job.

The feasibility of Shadow Replication, as an energy-aware, fault-tolerant model for cloud computing, requires a methodology to determine the speeds, both before and after a failure occurs, at which the main process and its associated shadow must execute to meet the performance and resiliency requirements of the supported application. In the following section, we describe a profit-based optimization framework to compute the optimal speeds of the Shadow Replication processes. In this framework, it is assumed that failures can be detected. 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", a sub-optimal Shadow Replication scheme, whereby both the main process and its associated shadow execute independently at stretched execution speeds to meet the expected response time and achieve the expected level of failure tolerance. Finally, the expected energy consumption of "re-execution", a commonly used fault-tolerance model in cloud computing, is described. Both models are used in the fault-tolerance comparative analysis carried out in this paper.

4. Reward Based Optimal Shadow Replication

In this section, we describe a profit-based optimization framework for the cloud-computing execution model previously described. In this framework, we adopt the fail-stop fault model, where a processor stops execution once a fault occurs and failure can be detected by other processes [16,17]. Furthermore, given that the probability of two individual nodes executing the same instances of a task fail at the same time is low, we will focus on the study of Shadow Replication model with a single shadow. Consequently, the completion of the main or its shadow results in the successful execution of the task. If the main process fails, it is implied that shadow process would complete the task. It is clear, however, that the model can be extended to support multiple shadow processes, as required by the application's fault-tolerance requirement.

Using this framework we compute the profit-optimized execution speeds by optimizing the expected profit, $E[\text{profit}]$, as an objective function:

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

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, $E[\text{profit}]$, to be the expected income, $E[\text{income}]$, minus the expected expense, $E[\text{expense}]$:

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

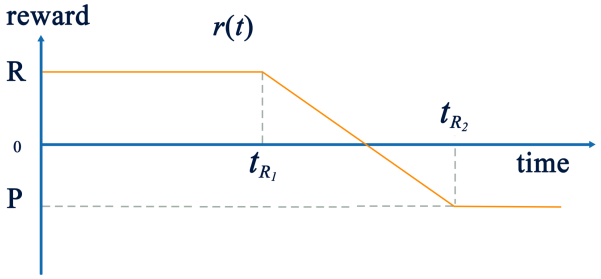
The expected income and expected expense will be explained in the following subsections.

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 2, customers expect that their job deployed on the cloud finishes by a mean response time t_{R_1}. In return, the provider earns a certain amount of reward, denoted by R, for satisfying the 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 a 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.

Figure 2. A reward function.



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 1a and Figure 1b. This results in the following completion time:

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

If the main process fails (shown in Figure 1c, 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 2. 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 [18].

4.2. Failure Model

Failure can occur at any point during the execution of the main or shadow process. We assume that processes belonging to the same job are mapped to different physical nodes and failures are independent. We also assume that among all the processes associated with the same task, at most one would fail, 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 processes 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 $\frac{1}{\lambda}$ [19,20].

4.3. Power and Energy Models

DVFS has been widely exploited as a technique to reduce CPU dynamic power [21,22]. It is well known that one can reduce the dynamic CPU power consumption at least quadratically by reducing the execution speed linearly. Moreover, advances in semiconductor technology have permitted flexible control at each core in modern many-core architecture through separate voltage and frequency islands [23–25]. The dynamic CPU power consumption of a computing core 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 $p(\sigma) = \rho \times \sigma_{max}^n + (1 - \rho) \times \sigma^n$. Throughout this paper we assume that dynamic power is cubic in relation to speed [26–28], and that

CPU can go into a sleep mode in which the power consumption is negligible. Therefore, the overall system power when executing at speed σ is defined as:

$$p(\sigma) = \begin{cases} \rho\sigma_{max}^3 + (1-\rho)\sigma^3 & \text{if not in sleep mode and } \sigma \geq 0 \\ 0 & \text{if in sleep mode} \end{cases} \quad (5)$$

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

$$E(\sigma, T) = p(\sigma) \times T \quad (6)$$

Corresponding to Figure 1, there are three cases considering the failure occurrence during the execution of a task with a pair of processes: neither the main or the shadow fails, only the shadow fails, and only the main fails. The expected energy consumption for a single task with a pair of processes is then the weighted average of the expected energy consumption in the three cases. In each case, the energy consumption is calculated for both the main process and the shadow process. In addition, the energy consumption in each case is calculated as an expected value because it depends on the failure occurrence time which we model using an exponential distribution. In the following, we first calculate the energy consumption in each of the three cases, denoted as E_1, E_2, and E_3 respectively. Then the total expected energy consumption is derived.

First consider the case where no failure occurs and the main process successfully completes the task at time t_c^m, corresponding to Figure 1a.

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

The probability of fault-free execution of both the main process and the shadow process is equal to the probability that the main does not fail times the probability that the shadow does not fail. We multiply this probability by the energy consumed by the main and the shadow process during this fault free execution, from start to completion at t_c^m.

Next, consider the case where the main process does not fail and the shadow process fails at some time, *t*, before the main process successfully completes the task, corresponding to Figure 1b.

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

Again, the probability of this case is equal to the probability that the main process does not fail times the probability that shadow fails. Energy consumption is included in the second factor 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 until its failure.

The last case is when the shadow does not fail and the main process fails at some time, *t*. In this case, the shadow process must continue executing until the task completes, corresponding to Figure 1c.

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

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 some task. If it is the case that processes must wait for all tasks to complete, then this energy overhead of synchronization 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.

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

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, since its shadow process has already failed only the main process has to wait if some other task is delayed. 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 2, 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 expense of running the cloud computing service is more than just energy, and must includes hardware, software, 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 that energy will become a dominate factor [5,6], we decided to focus on this challenge and leave other aspects to future work.

Based on the above formalization of the optimization problem, the MATLAB Optimization Toolbox [29] was used to solve the resulting nonlinear optimization problem. The parameters of this problem are listed in Table 2.

Table 2. Analytical model main parameters.

Parameter	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)

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:

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

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 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, where a checkpoint is implicitly taken at the end of each phase, and because the tasks are loosely coupled, they can restart independently.

Based on the single 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 before 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

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

7. Evaluation

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

- Static power ratio ρ—The portion of power that is independent of 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 the 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 h, the node MTBF is 5 years, the number of tasks is $100,000$, and the response time thresholds for maximal and minimal rewards are 1.3 h and 2.6 h respectively. Static power ratio is typically from 40% to 70% [30–32]; the task size and the number of tasks are inferred from current workloads at Google, Amazon and their processing speed [11,12]; there are no published MTBF statistics for the cloud environment, but Google assumes 30 years for super reliable servers [33], so 5 years is reasonable since today's cloud datacenters use inexpensive commodity computers [34]; and we determined the response time thresholds based on our belief that customers would expect some extra time (like 30%) for most cases and their endurance limit is the double of the expected completion time. 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.

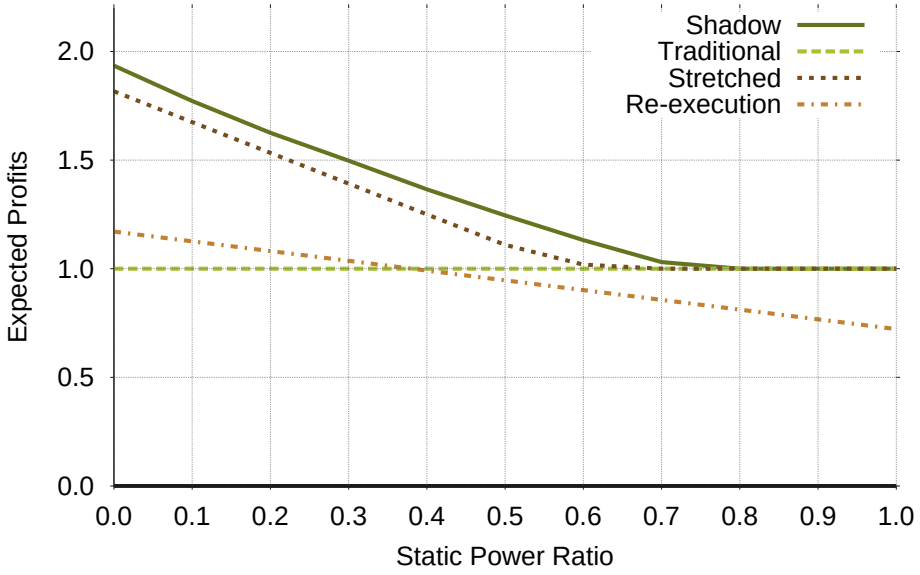
Table 3 shows how the profit-optimized execution speeds of Shadow Replication will change as static power increases. The execution speeds increase to reduce the execution time as static power ration increases. Observe that σ_a is always equal to σ_{max}, which means that after sensing the failure of 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.

Table 3. Speeds for different static power ratio. MTBF = 5 years, N = 100,000, W = 1 h, t_{R_1} = 1.3 h, t_{R_2} = 2.6 h.

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

The potential profit gains achievable by using profit-aware replication techniques decreases as static power increases, as is shown in Figure 3. 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% [30–32] 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 re-execution.

Figure 3. Profit for different static power ratio. MTBF = 5 years, N = 100,000, W = 1 h, t_{R_1} = 1.3 h, t_{R_2} = 2.6 h.



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.

In Table 4 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.

Table 4. Speeds for different response time threshold. $\rho = 0.5$, MTBF = 5 years, N = 100,000, W = 1 h.

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 4 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.

Figure 4. Profit for different response time threshold. $\rho = 0.5$, MTBF = 5 years, N = 100,000, W = 1 h.

7.3. Sensitivity to Number of Tasks

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 keep all the other processes waiting and thus hurt the total income significantly. Thus, Shadow Replication will adjust its execution speeds to reduce the waiting time.

Table 5 is similar to Table 4 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 100,000, the speeds stay the same when $N \geq 100,000$.

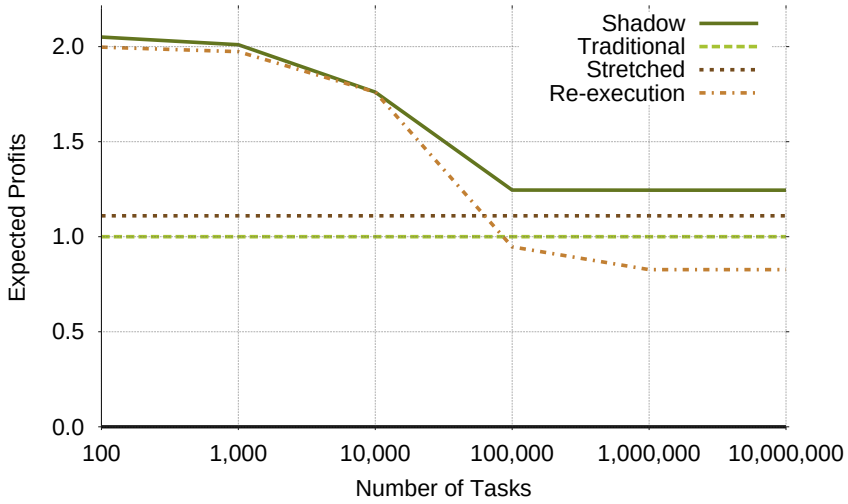
Table 5. Speeds for different number of tasks. $\rho = 0.5$, MTBF = 5 years, W = 1 h, $t_{R_1} = 1.3$ h, $t_{R_2} = 2.6$ h.

N	σ_m	σ_b	σ_a
100	0.80	0.00	1
1,000	0.84	0.00	1
10,000	1.00	0.00	1
100,000	0.86	0.72	1
1,000,000	0.86	0.72	1
10,000,000	0.86	0.72	1

Figure 5 confirms that for small number of tasks re-execution is more profitable than traditional replication. However, re-execution is not scalable as its profit decreases rapidly after N reaches 10,000. 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-execution, respectively.

Figure 5. Profit for different number of tasks. $\rho = 0.5$, MTBF = 5 years, W = 1 h, t_{R_1} = 1.3 h, t_{R_2} = 2.6 h.



7.4. Sensitivity to Failure

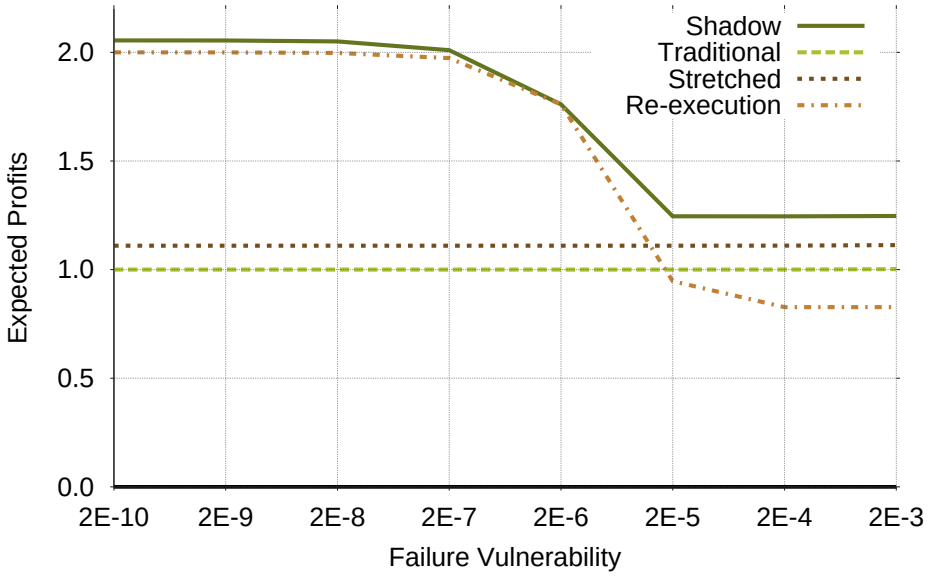
The ratio between task size and node MTBF represents the task's vulnerability to failure, specifically it is an approximation of the probability that failure occurs during the execution of the task. In our analysis we found that increasing task size will have the same effect as reducing node MTBF. Therefore,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

As seen in Table 6 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 5). 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 6.

Table 6. Speeds for different task size over MTBF. ρ = 0.5, N = 100,000, t_{R_1}=1.3 h, t_{R_2} = 2.6 h.

$W/MTBF$	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 6. Profit for different task size over MTBF. ρ = 0.5, N = 100,000, t_{R_1} = 1.3 h, t_{R_2} = 2.6 h.



7.5. Sensitivity to Application Characteristics

Cloud computing applications exhibit different behaviors, in term of computation requirements, data access patterns and communication dependencies. While some applications are compute-intensive, others involve the processing of increasingly large amounts of data. The comparative analysis in this subsection is carried out for three different benchmark applications, representing a wide range of applications: Business Intelligence, Bioinformatics and Recommendation System [11]. 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.

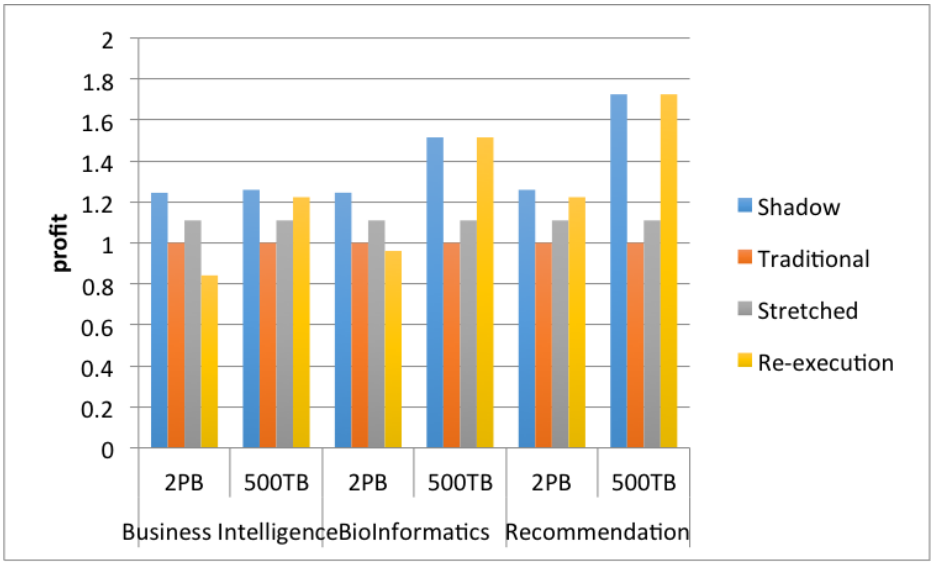
Using the results of the experiments reported in [11], we derived the time required to process data for each application type (Table 7). We assume that these processing rates per task will not change when scaling the applications to future cloud environments. This is a reasonable assumption given that map-reduce tasks are loosely coupled and data are widely distributed, therefore data and task workload will scale linearly.

Table 7. Cloud Applications [11].

Application	Class	Processing Rate
Business Intelligence	Data Analytics	3.3 (MB/s)
Bioinformatics	Data Analytics	6.6 (MB/s)
Recommendation System	Web Search	13.2 (MB/s)

In Figure 7, we compare the expected profit for each application using each of the four resilience techniques. We consider two data sizes expected in future cloud computing environments, 500 TB and 2 PB. 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 min 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.

Figure 7. Application comparison. ρ = 0.5, MTBF = 5 years, N = $500,000$, t_{R_1} = 1.3 t_{min}, t_{R_2} = 2.6 t_{min}.



7.6. Sensitivity to Communication Types

One of the key application factors is the amount of coordination between the tasks. For example, an application might require no inner-task communication, allowing a single task to fail and restart without effecting the completion time of other tasks. However, if inner-task communication is required, then the failing of a single task could potentially delay all tasks with which it would have normally communicated. The amount of communication can be thought of as the amount of dependencies one task has upon all other tasks and affects the application completion time and energy consumption. There are wide range of potential applications but they can be classified into one of the following types: No Dependency, Blocking Dependency or Full Dependency. Details of these communication types can be found in Table 8. When the tasks are not completely independent, a synchronization overhead will be incurred when a failed main process needs to be replaced with its corresponding shadow process. The way we model the synchronization overhead of barrier communication is that all the main processes except the failed one would have to wait idly (speed equals 0) and consume static power until the

shadow process catches up. The way we model the synchronization overhead of full communication is that all the main processes except the failed one would have to keep running and consume full power until the shadow process catches up. The reason for the difference is that barrier communication needs coordination only at the very end while full communication needs frequent coordination.

Table 8. Classification of application communication types and their descriptions.

Application Type	Case	Description
No Dependency	Map Tasks	There are no inner-task dependencies, all tasks must complete but could do so with no coordination.
Blocking Dependency	Independent Simulation	Tasks can execute independently but are required to communicate at the end of execution to complete the solution. This represents applications that divide work into non-overlapping parts but require collective operations at the end of execution or perform asynchronous communication during execution, the difference being how much work is necessary at the end of execution.
Full Dependency	Coordinated Simulation	Tasks are fully-dependent upon other tasks and require constant communication between tasks. This represents an application that frequently performs collective operations during execution.

In this subsection, we evaluate the profit gains of Shadow Replication over other resilience methods for applications with different communication types, *i.e.*, no communication, barrier communication, and full communication. Figure 8 compares Shadow Replication with Re-execution. Generally speaking, as static power ratio increases the profit gains of Shadow Replication decreases. This is because Shadow Replication uses DVFS to reduce dynamic power, whose portion becomes smaller when static power ratio increases by definition. However, the above conclusion has one exception, *i.e.*, the profit gains of barrier communication goes up after static power ratio reaches 0.7. This matches previous results in Section 7.1, and the reason is that Shadow Replication converges to Traditional Replication when static power ratio reaches 0.7 and then its profit becomes constant while the profit of Re-execution continues to decrease. Another conclusion from Figure 8 is that the higher the synchronization requirement is, the higher profit gains Shadow Replication can achieve over Re-execution. This implies that the failure of a single process in a highly synchronized application can incur a much larger cost for re-execution.

Figure 9 compares Shadow Replication with Re-execution. For the same reason, the profit gains of Shadow Replication decreases as static power ratio increases, which is similar to Figure 8. The major difference from Figure 8 is that when static power ratio is low, there is barely any difference among the three communication types in Figure 9. After looking into the raw data, we found the reason is communication types have no influence on Traditional Replication and very limited influence on Shadow Replication while having large influence on Re-execution when static power ratio is low.

Figure 8. Profit gains of Shadow Replication over Re-execution for different static power ratio. MTBF = 5 years, N = 100,000, W = 1 h, t_{R_1} = 1.3 h, t_{R_2} = 2.6 h.

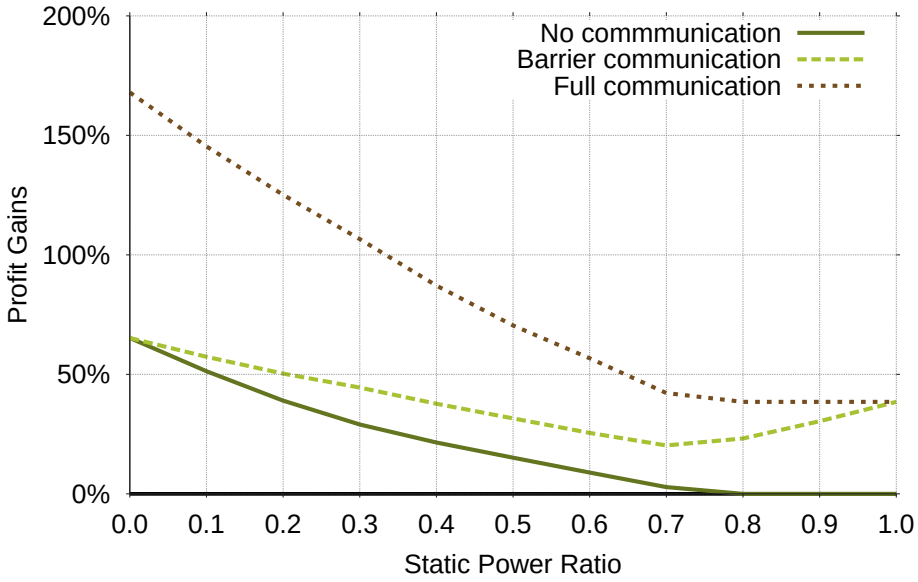
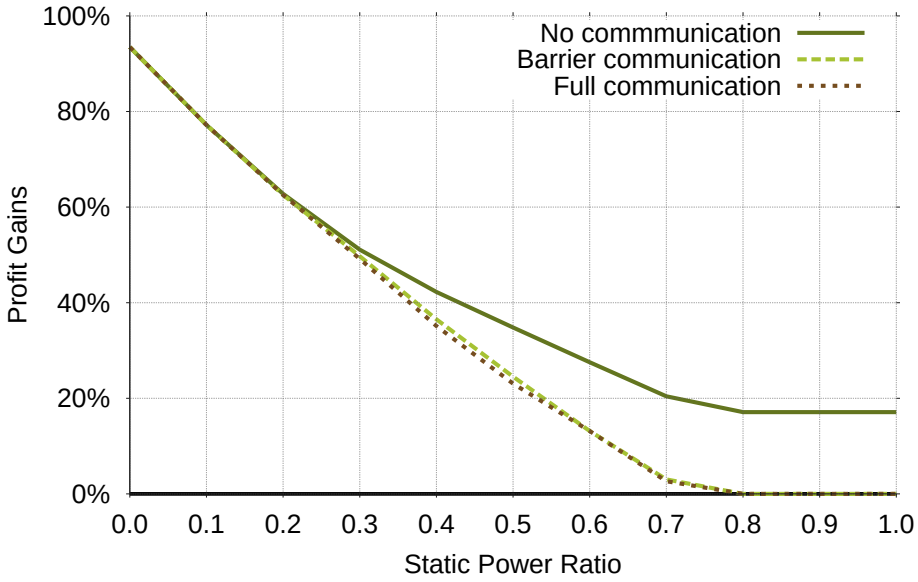


Figure 9. Profit gains of Shadow Replication over Traditional Replication for different static power ratio. MTBF = 5 years, N = 100,000, W = 1 h, t_{R_1} = 1.3 h, t_{R_2} = 2.6 h.



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) [35–38]. 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 [39]. 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 [40,41]. 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 [42] the authors propose a high-level scheme that allows users to specify the desired level of resilience, while hiding implementation details. Similarly, [43] 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 [1] propose a collaborative solution to enhance fault-tolerance efficiency. In [44], the authors leverage virtual disk image snapshots to minimize the storage space and checkpointing overhead. In [34] the author 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 [45,46]. 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. Conclusions

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.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grants Number CNS-1252306 and CNS-1253218. 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.

Author Contributions

Xiaolong Cui planned and drafted the paper. Bryan Mills revised the paper. Taieb Znati and Rami Melhem offered great suggestions and polished the paper.

Conflicts of Interest

The authors declare no potential conflict of interest.

References

1. Tchana, A.; Broto, L.; Hagimont, D. Approaches to cloud computing fault tolerance. In Proceedings of the International Conference on Computer, Information and Telecommunication Systems (CITS), Amman, Jordan, 14–16 May 2012; pp. 1–6.
2. Anderson, E.; Eschinger, C.; Wurster, L.F.; de Silva, F.; Contu, R.; Liu, V.K.; Biscotti, F.; Petri, G.; Zhang, J.; Yeates, M.; *et al. Forecast Overview: Public Cloud Services, Worldwide, 2011–2016, 4Q12 Update*; Gartner: Stamford, CT, USA, 2013.
3. Amazon. Amazon Elastic Compute Cloud. Available online: http://aws.amazon.com/ec2/ (accessed on 12 August 2014).
4. Schroeder, B.; Gibson, G. A large-scale study of failures in high-performance computing systems. *IEEE Trans. Dependable Secur. Comput.* **2010**, *7*, 337–350.
5. Elnozahy, M.; Kistler, M.; Rajamony, R. Energy conservation policies for web servers. In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, Berkeley, CA, USA, 26–28 March 2003.
6. Raghavendra, R.; Ranganathan, P.; Talwar, V.; Wang, Z.; Zhu, X. No "power" struggles: coordinated multi-level power management for the data center. *SIGARCH Comput. Archit. News* **2008**, *36*, 48–59.

7. Gelenbe, E.; Hernaǹdez, M. Optimum checkpoints with age dependent failures. *Acta Inform.* **1990**, *27*, 519 – 531.
8. Tsai, W.T.; Zhong, P.; Elston, J.; Bai, X.; Chen, Y. Service replication strategies with MapReduce in clouds. In Proceedings of the 10th International Symposium on Autonomous Decentralized Systems (ISADS), Hiroshima, Tokyo, 23–27 March 2011; pp. 381–388.
9. Ko, S.Y.; Hoque, I.; Cho, B.; Gupta, I. Making cloud intermediate data fault-tolerant. In Proceedings of the 1st ACM symposium on Cloud computing, Indianapolis, IN, USA, 10–11 June 2010; pp. 181–192.
10. Ferdman, M.; Adilehet, A.; Kocberber, O.; Volos, S.; Alisafaee, M.; Jevdjic, D.; Kaynak, C.; Popescu, A.D.; Ailamaki, A.; Falsafi, B.; *et al.* Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, London, UK, 3–7 March 2012; pp. 37–48.
11. Sangroya, A.; Serrano, D.; Bouchenak, S. Benchmarking dependability of MapReduce systems. In Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems, Washington, DC, USA, 8–11 October 2012; pp. 21–30.
12. Lin, J.; Dyer, C. Data-intensive text processing with MapReduce. *Synth. Lect. Hum. Lang. Technol.* **2010**, *3*, 1–177.
13. Choi, K.; Soma, R.; Pedram, M. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2005**, *24*, 18–28.
14. Choi, K.; Soma, R.; Pedram, M. Dynamic voltage and frequency scaling based on workload decomposition. In Proceedings of the 2004 International Symposium on Low Power Electronics and Design, Bangalore, India, 11–13 August 2004; pp. 174–179.
15. Costa, P.; Pasin, M.; Bessani, A.; Correia, M. Byzantine fault-tolerant MapReduce: Faults are not just crashes. In Proceedings of IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), Athens, Greece, 29 November–1 December 2011; pp. 32–39.
16. Gartner, F.C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.* **1999**, *31*, 1–26.
17. Cristian, F. Understanding fault-tolerant distributed systems. *Commun. ACM* **1991**, *34*, 56–78.
18. Daw, N.D.; Touretzky, D.S. Long-term reward prediction in TD models of the dopamine system. *Neural Comput.* **2002**, *14*, 2567–2583.
19. Wasserman, G. *Reliability Verification, Testing, and Analysis in Engineering Design*, 1st ed.; CRC Press: Boca Raton, FL, USA, 2002.
20. Ross, S.M. *Introduction to Probability and Statistics for Engineers and Scientists*, 4th ed.; Academic Press: San Diego, CA, USA, 2009.
21. Flautner, K.; Reinhardt, S.; Mudge, T. Automatic performance setting for dynamic voltage scaling. *Wirel. Netw.* **2002**, *8*, 507–520.

22. Pillai, P.; Shin, K.G. Real-time dynamic voltage scaling for low-power embedded operating systems. In Proceedings of the Eighteenth ACM Symposium on Operating systems Principles, Banff, AB, Canada, 21–24 October 2001; pp. 89–102.
23. Vajda, A. *Programming Many-Core Chips*, 1st ed.; Springer: New York, NY, USA, 2011.
24. Ogras, U.; Marculescu, R.; Marculescu, D.; Jung, E.G. Design and management of voltage-frequency island partitioned networks-on-chip. *IEEE Trans. Very Large Scale Integr. Syst.* **2009**, *17*, 330–341.
25. Guang, L.; Nigussie, E.; Koskinen, L.; Tenhunen, H. Autonomous DVFS on Supply Islands for Energy-Constrained NoC Communication. In Proceedings of the 22nd International Conference on Architecture of Computing Systems, Delft, The Netherlands, 10–13 March 2009; pp. 183–194.
26. Rusu, C.; Melhem, R.; Mossé, D. Maximizing rewards for real-time applications with energy constraints. *ACM Trans. Embed. Comput. Syst.* **2003**, *2*, 537–559.
27. Zhai, B.; Blaauw, D.; Sylvester, D.; Flautner, K. Theoretical and practical limits of dynamic voltage scaling. In Proceedings of the 41st Design Automation Conference, San Diego, CA, USA, 7–11 June 2004; pp. 868–873.
28. Park, J.; Shin, D.; Chang, N.; Pedram, M. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors. In Proceedings of the 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), Austin, TX, USA, 18–20 August 2010; pp. 419–424.
29. MathWorks. *Optimization Toolbox*; MathWorks: Natick, MA, USA, 2013.
30. Butts, J.; Sohi, G. A static power model for architects. In Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture, Monterey, CA, USA, 10–13 December 2000; pp. 191–201.
31. Kim, N.S.; Austin, T.; Blaauw, D.; Mudge, T.; Flautner, K.; Hu, J.S.; Irwin, M.J.; Kandemir, M.; Narayanan, V. Leakage current: Moore's law meets static power. *Computer* **2003**, *36*, 68–75.
32. Khouri, K.; Jha, N. Leakage power analysis and reduction during behavioral synthesis. *IEEE Trans.Very Large Scale Integr. Syst.* **2002**, *10*, 876–885.
33. Dean, J. Designs, Lessons and Advice from Building Large Distributed Systems. Available online: http://www.lamsade.dauphine.fr/litwin/cours98/CoursBD/doc/dean-keynote-ladis2009_scalable_distributed_google_system.pdf (accessed on 8 August 2014)
34. Zheng, Q. Improving MapReduce fault tolerance in the cloud. In Proceedings of 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, 19–23 April 2010; pp. 1–6.
35. Agarwal, S.; Garg, R.; Gupta, M.S. Adaptive incremental checkpointing for massively parallel systems. In Proceedings of the 18th Annual International Conference on Supercomputing, Saint-Malo, France, 26 June–1 July 2004; pp. 277–286.
36. Alvisi, L.; Elnozahy, E.; Rao, S.; Husain, S.; de Mel, A. An analysis of communication induced checkpointing. In Proceedings of Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, Madison, WI, USA, 15–18 June 1999; pp. 242–249.
37. Daly, J. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst.* **2006**, *22*, 303–312.

38. Helary, J.M.; Mostefaoui, A.; Netzer, R.; Raynal, M. Preventing useless checkpoints in distributed computations. In Proceedings of the Sixteenth Symposium on Reliable Distributed Systems, Durham, NC, USA, 22–24 October 1997; pp. 183–190.
39. El Mehdi Diouri, M.; Gluck, O.; Lefevre, L.; Cappello, F. Energy considerations in checkpointing and fault tolerance protocols. In Proceedings of the 2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W), Boston, MA, USA, 25–28 June 2012; pp. 1–6.
40. Ferreira, K.; Stearley, J.; Laros, J.H., III.; Oldfield, R.; Pedretti, K.T.; Brightwell, R.; Riesen, R.; Bridges, P.G.; Arnold, D. Evaluating the viability of process replication reliability for Exascale Systems. In Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, WA, USA, 14–17 November 2011; pp. 44:1–44:12.
41. Sousa, P.; Neves, N.; Verissimo, P. Resilient state machine replication. In Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing, Washington, DC, USA, 12–14 December 2005; p. 5.
42. Jhawar, R.; Piuri, V.; Santambrogio, M. Fault tolerance management in cloud computing: A system-level perspective. *IEEE Syst. J.* **2013**, *7*, 288–297.
43. Zhao, W.; Melliar-Smith, P.; Moser, L. Fault tolerance middleware for cloud computing. In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), Miami, FL, USA, 5–10 June 2010; pp. 67–74.
44. Nicolae, B.; Cappello, F. BlobCR: Efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots. In Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, WA, USA, 12–18 November 2011; pp. 34:1–34:12.
45. Chen, X.; Liu, X.; Wang, S.; Chang, X.W. TailCon: Power-minimizing tail percentile control of response time in server clusters. In Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems (SRDS), Irvine, CA, USA, 8–11 October 2012; pp. 61–70.
46. Lin, M.; Wierman, A.; Andrew, L.; Thereska, E. Dynamic right-sizing for power-proportional data centers. In Proceedings of IEEE INFOCOM, Shanghai, China, 10–15 April 2011; pp. 1098–1106.