Journal of Universal Computer Science, vol. 29, no. 8 (2023), 892-910
submitted: 5/9/2022, accepted: 5/4/2023, appeared: 28/8/2023 CC BY-ND 4.0

diffReplication − An Energy-Aware Fault Tolerance Model for Silent Error Detection and Mitigation in Heterogeneous Extreme-scale Computing Environment

Longhao Li
(University of Pittsburgh, Pittsburgh PA 15260, USA
lol16@pitt.edu)

Taieb Znati
(United Arab Emirates University, Al Ain, United Arab Emirates
znati@uaeu.ac.ae)

Rami Melhem
(University of Pittsburgh, Pittsburgh PA 15260, USA
https://orcid.org/0000-0001-6403-5446, melhem@cs.pitt.edu)

Abstract: At extreme scale, the frequency of silent errors – a class of errors that remain undetected by low-level error detection mechanisms – increases significantly with the computational complexity of the application and the scale of the computing infrastructure. As hardware and software advances are made to usher in the next scientific era of computing, developing new approaches to mitigate the impact of silent errors remains a challenging problem. In this work, we propose an energy-aware fault-tolerance model, referred to diffReplication to overcome silent errors. In the proposed model, the main process is associated with one replica that executes at the same rate as the main process, and one diffReplica that is executed at a fraction of the main process' execution rate. If the main and its replica reach consensus at the end of a computation phase, the state of the diffReplica is updated and computation is resumed. If the synchronization attempt results in a disagreement, however, the diffReplica increases its execution speed to complete the computation and quickly reach the synchronization barrier. Assuming a single error over any given synchronization interval, a majority voting is used to reach consensus and tolerate silent errors. To further enhance its performance, diffReplication is augmented with speculative execution, whereby the main or its fast replica is selected to continue execution without waiting for the diffReplica. The selection process is based on the previous behaviour of the main and its replica. A performance analysis study is carried out to assess the performance of diffReplication, in terms of the energy saving and time-to-completion reduction achieved by the diffReplication scheme. The experiment shows that speculative execution reduces the time to completion with additional energy, and dynamic decision-making balances the energy consumption and time to completion.

Keywords: silent error, fault tolerance, extreme-scale, heterogeneous environment
Categories: C.2.4, D.4.5
DOI: 10.3897/jucs.94462

1 Introduction

In both the scientific and commercial fields, the complexity and computational intensity required to solve problems in these fields have been steadily increasing. Future

applications to solve these problems will require massive scale parallelism to support the training of AI models and process extremely large data sets, in order to achieve significantly higher resolutions and fidelity than what is possible with existing computing infrastructure. The massive levels of parallelism will be achieved using hundreds of thousands of threads executed as an ordered sequence of local computation and global communication phases, separated by barrier synchronizations.

Propensity to failure and the need to operate within *power constraints* are two of the most daunting challenges faced by extreme scale systems. They are both direct results of the computational complexity of the underlining application and the siginificant amount of parallelism necessary to meet its QoS requirements, in terms of energy saving and reduced time-to-completion [Sachs, 2011]. This stems from the fact that the rate of system level failures increases dramatically as the number of computing nodes increases in spite of unprecedented and continued improvement of the software and hardware technologies. For example, a computing infrastructure with 200,000 nodes will experience a mean time between failures (MTBF) of less than one hour, even when the MTBF of an individual node is as large as 5 years [Mills et al., 2013, Riesen et al., 2010]. The continued pressure to physically shrink board- and chip-level control further compounds the propensity of computing systems to failures.

As the scale of the complexity of the computational infrastructure continues to increase, erroneous behaviour, caused by silent errors, is gradually being observed in computing nodes. This type of errors is known as silent data corruption, silent data errors or corrupt execution errors. These errors, which will be referred to as silent errors (SEs) in this paper, are neither bugs nor software errors. It is caused by bit flips in memory, disk, or a processor core register. They occur randomly and unpredictably and remain undetected by low-level error detection mechanisms. At extreme scale, the likelihood of SE occurrence increases significantly. Since a single SE may corrupt the entire execution, it is critical to apply SE-tolerance techniques to mitigate the impact on applications' QoS requirements in extreme-scale computing systems. To tolerate this type of errors, two tasks need to be performed: fault detection and fault correction. To detect SE, the computational results need to be verified. The most accurate method is to compare the results computed by other replicas and check if they match. Unmatched results is only an indication that SE has occurred, but what the correct answer is remains unknown. In order to tolerate f SEs, $(2f+1)$ replicas are required to determine the correct outcome, using majority voting [Veronese et al., 2011]. The resource usage required by this SE-tolerance scheme, however, is tripled, with direct implication on energy consumption and time-to-completion.

To address the shortcoming of the basic SE-tolerance scheme proposed above, the energy-ware diffReplication model is proposed to overcome silent errors in extreme scale computing infrastructure. In the proposed model, the main process is associated with one replica that executes at the same rate as the main process, and one diffReplica that executes at a fraction of the main process' execution rate. If the main and its replica reach consensus at the end of a synchronization interval, the state of the diffReplica is updated and computation is resumed. If consensus is not reached, the diffReplica increases its execution speed to complete the computation and quickly reach the synchronization barrier. Assuming a single error occurs over a given synchronization interval, a majority voting is used to tolerate silent errors. To further enhance its performance, diffReplication is augmented with speculative execution, whereby the main or its fast replica is selected to continue execution without waiting for the

diffReplica. The selection of which process continues execution without waiting for the diffReplica is based on the previous behaviour of each process.

Mindful of the fact that, in heterogeneous environment, silent errors occur unpredictably and at different rates depending on the underlying computational components, the diffReplication model takes heterogeneity into consideration to determine how replication is carried out and at what speed diffReplicas executes. A performance analysis study is carried out to assess the energy saving and time-to-completion reduction achieved by the diffReplication scheme. The experiment shows that speculative execution reduces the time to completion with additional energy, and dynamic decision-making balances the energy consumption and time to completion.

The rest of the paper is introduced as follows. Related work is discussed in Section 2. The diffReplication execution model is introduced in Section 3. Experimental evaluation is introduced in Section 4. Section 5 concludes the paper and proposes areas for future research.

2 Related Work

The field of fault tolerance in computing systems is well established, and significant advances on how to deal with faults have been achieved by different communities. Rollback and recovery are predominated mechanisms to achieve fault tolerance in current HPC environments [Elnozahy et al., 2002, Kalaiselvi and Rajaraman, 2000]. Upon the occurrence of a fault, recovery is achieved by restarting the computation from a safe checkpoint [Kalaiselvi and Rajaraman, 2000]. Coordinated checkpointing and rollback recovery schemes have been proposed [Elnozahy et al., 2002, Kalaiselvi and Rajaraman, 2000, Daly, 2003]. The drawback of coordinated checkpointing is a lack of scalability, as it requires global process coordination [Bougeret et al., 2012, Hargrove and Duell, 2006, Losada et al., 2019]. Uncoordinated checkpointing and rollback recovery schemes have been proposed as an alternative approach to achieve fault-tolerance. These schemes, however, have not been widely adopted in HPC environments, due to their dependency on the underlying application [Guermouche et al., 2011]. Multi-level checkpointing can benefit tolerance to failure but may increase failure rates of individual nodes and increase per-node cost [Hakkarinen and Chen, 2013]. Message logging-based approaches have been proposed to harness applications' temporal computation and communication patterns to reduce the cost of checkpointing in hybrid systems.

Process and state machine replication have long been used to provide fault tolerance in distributed [Schneider, 1990] and mission critical systems [Bartlett, 1981]. Based on this technique, processes' state and computation are replicated across independent computing nodes. Redundancy has been proposed, to augment existing checkpointing techniques [Stearley et al., 2012, Elliott et al., 2012, Casanova et al., 2012]. In its basic form, replication, when used to tolerate f fail-stop failures, requires the execution of $(f+1)$ replicas. Byzantine fault-tolerant systems require at least $(2f+1)$ replicas to overcome f failures. Consequently, tolerating a single failure in a system of N nodes, executing independently, requires doubling or tripling the number of nodes, depending on the type of failure to be tolerated.

In extreme scale environments, the energy cost of such a replicated infrastructure may quickly become prohibitive, especially in the system that has to operate under a

power cap. Furthermore, the cost of recovery remains proportional to the system size and not to the degree of failure. In our previous work, we took an adaptive approach to dynamically adjust the replica's execution rate based on the predicted failures to tolerate fail-stop errors. Replicas only execute at higher rates when their associated main processes are at risk of failure. Therefore, these replicas can be collocated to fewer numbers of processors than regular replicas. In this case, the energy consumption is reduced, without increasing the task completion time [Li et al., 2021, Cui et al., 2018, Mills et al., 2014].

The silent data corruption (SE) problem has been the subject of large number of studies, which can be classified as either hardware-, or software-based, redundancy-based, algorithm-based, and runtime-based [Di and Cappello, 2016, Lee et al., 2011, Peña et al., 2015, Bosilca et al., 2015, Mukherjee et al., 2005, Chen et al., 2018, Krluku et al., 2019, Huang, et al., 2022]. Error Correction Code (ECC) is an architecture solution that correct bit-wised errors [Mukherjee et al., 2005]. This method is typically well adopted to tolerate errors in memories. But may limited when it comes to correcting computational errors incurred by processors. Similarly, algorithm-based fault tolerant (ABFT) method can detect and recover errors using the properties of the giving algorithm such as protecting matrix data by checking rows' checksums [Bosilca et al., 2015]. ABFT methods are algorithm specific methods and may not be general enough to tolerate unknown errors.

Process replications can be used to detect and correct SEs during run time [Fiala et al., 2012]. The overhead, however, may be prohibitive in extreme-scale environment. To reduce the overhead, and detect the majority of SEs, Di et al. proposed an adaptive SE detection method based on local run time data [Di and Cappello, 2016]. Similarly, Huang et al. proposed a method to adaptively duplicate selected instructions to maximize the SE coverage and maintain a low performance overhead [Huang, et al., 2022]. These methods, however, does not capture all the SEs. In extreme-scale systems, a single undetected SE can corrupt the entire computation and cause unacceptable increase in completion time and energy consumption, due to re-execution. How to reduce the energy consumption and ensure all the SEs can be detected and corrected in a reasonable amount of time is paramount to the design of an effective scheme to tolerate SEs in extreme-scale systems.

FT-MPI has been proposed as a user-level framework to control failures in a dynamic, HPC environments [Fagg and Dongarra, 2000]. The focus of this paper is on node failure checking and corrective recovery actions in MPI HPC/MPI environment. The framework enhances the MPI checkpointing and roll-back recovery scheme with user-level capabilities to control failures. It is suitable to deal with fail-stop failures in HPC environments. Our scheme is designed for extreme scale computing, where the time between system failures is smaller than the time between successive checkpoints. Furthermore, our scheme is designed to deal with silent failures, under the dual constraint of energy minimizing and adherence to time-to-completion.

3 diffReplication Execution Model

The diffReplication fault-tolerance model assumes that the execution of a computational job follows a Bulk Synchronous Parallel (BSP) model, whereby a job is split into N equal-sized tasks executed in parallel. The computation is segregated into

consecutive global supersteps [Valiant, 1990]. In each superstep, the main processes execute independently of each other and communicate when they reach the synchronization barrier. A superstep ends when all computation in the superstep is completed and all messages have been sent.

The diffReplication model consists of one main process, a set of $(M-1)$ replicas and one diffReplica. The main process and its replicas execute the same code at the same processor execution rate. The diffReplica, however, executes the same code as the main process, but at a lower execution rate. Collectively the main process, its replicas and diffReplica constitute a diffReplication group. Processes within a group do not share messages, to prevent the propagation of silent errors.

In order to overcome f SEs, the diffReplication group size must be at least $2f+1$. When the processes reach a synchronization interval, which consists of a single superstep, a majority-based consensus algorithm is executed to validate the computation output. Based on this algorithm, the output of the group is that of the diffReplication process majority. Without loss of generality, we assume a single SE within a superstep. Consequently, the size of the diffReplication group is 3, namely the main process, one replica and one diffReplica.

During the execution within a superstep, the main and replica processes execute at the same rate μ to ensure no delay in SE detection. The diffReplica, however, initially execute at a slower rate, σ^b. When they reach the end of a superstep, the main process and its replica engage in a process to validate their output. If consensus is reached over the computation output, indicating that no process has incurred a silent error during its execution, the state of the diffReplica is "leaped" to be an exact replica of the main process's state. Leaping consists mainly in consolidating the address space of the main process with the address space of the diffReplica[1]. The case of no silent error occurring during execution is depicted in Figure 1.

Failure to reach consensus, however, indicates that at least one SE occurred during the superstep execution interval. In response, the diffReplica is signalled to increase its processor execution rate to σ^a, in order to reach the end of superstep execution and validated the group output. To this end, majority voting is used to identify the faulty process and determine the group output. The case of a silent error occurring during the execution is depicted in Figure 2.

3.1 Speculative Execution

While waiting for the diffReplica to reach the end of a superstep, the main and its replica remain idle, thereby consuming static energy. In addition to the wasted energy, the delay incurred by the main and its replica, while waiting for the diffReplica, can also significantly increase the overall time to completion of the computational job. To address both shortcomings, speculative execution is used. Based on this approach, the state of one process, either the main's or its replica's, is selected to be the correct state. The other process is then leaped and execution of the two processes resumes immediately, without waiting for the diffReplica to reach the end of a superstep. Both

[1] Techniques, based on duplication mechanisms and awareness of context, which are frequently used to migrate virtual machines, can be easily adopted to reduce leaping overhead.

processes execute at a rate, μ^s, lower than μ. This rate is determined to achieve a balance between energy consumption and increase in time-to-completion.

When the diffReplica reaches the synchronization barrier, majority voting is used to identify which process, the main or its replica, is faulty and determine the superstep outcome. Depending on the voting outcome, multiple scenarios are possible:

Case 1 – **Speculative execution with faulty process**: Upon reaching the end of the superstep, the diffReplica compares its output with the output of the selected process. The discrepancies between the two outputs indicates that the faulty computational state has been selected to proceed with the speculative execution. In response, the diffReplica interrupts the current execution of the main and its replica. Leaping then takes place to bring the state of the two processes to the correct state of the diffReplica. Execution of all processes at the original rate of execution resumes immediately after the completion of the leaping process. The basic steps of this case are depicted in Figure 3.

Case 2 – **Speculative execution with non-faulty process**: Upon reaching the end of a superstep, the diffReplica compares its output with the output produced by the selected process. If it is determined that the non-faulty process has been correctly selected to carry out the speculative execution, the diffReplica process does not take any action to inform the main or its associated replica. Instead, it continues its execution, but at an execution rate, σ^c, which is lower than the maximum speed, σ_{max}, but higher than the initial speed, σ^b. Adjusting the execution rate to σ^c enables the scheme to proactively deal with the potential occurrence of a second silent error during the current superstep. In the one hand, resuming execution at a rate σ^b results in energy saving, but increases the delay to break the tie, if a second silent error detected at the end of the current superstep. This case is depicted in Figure 4. On the other hand, resuming execution rate at a rate σ_{max} may increase energy consumption unnecessarily, if no silent error is detected at the end of the superstep. The selection of σ^c as the execution rate strikes a balance between reducing the job time-to-completion and energy consumption. Depending on the outcome of the output validation upon reaching the end of the superstep, two subcases are possible:

Subcase 2.1 – Output validation disagreement: When the output validation fails, the diffReplica, executing at speed, σ^c, increases its speed to σ_{max} to complete the remaining computation and reach the end of a superstep. The output of the diffReplica is used to determine the fault-process and trigger the appropriate leaping process to bring all processes to a consistent state. A new superstep is entered, with the main and its replica executing at speed, μ, and the diffReplica executing at initial speed, σ^b. The basic steps of this scenario are depicted in Figure 4.

Subcase 2.2 – Output validation agreement: If the main and its replica reach an agreement on the end of the subsequent superstep, diffReplica's state is leaped to mirror the main process's state. All processes resume execution with speed, μ, for the main and its associated replica and with the initial speed, σ^b, for the diffReplica. The basic steps of this scenario are depicted in Figure 5.

3.2 Dynamic Decision Making

Speculative execution may reduce the time to completion. It also, however, may induce additional energy consumption as an incorrect choice may consume more energy. To improve energy efficiency and ensure an acceptable completion time, we propose a real-time approach to determine whether to perform a speculative execution upon SE is detected based on the estimated cost incurred by time delay or the estimated cost of the energy consumed in making the decision. For the speculative execution approach, the dominating factor is the cost of wasted energy consumption, while the cost of the no-speculative approach is the time delay. If saving energy is more important than reducing time to completion, waiting for correction is the best option. Otherwise, speculative execution should be conducted. We use different cost functions for wasted energy and time delays to make them comparable, which is explained in the following subsection. Firstly, we need to calculate the estimated wasted energy and time delay. The estimated wasted energy for speculative execution is calculated using following equation.

$$\bar{E}_{waste} = p_f \times 2E^d(\mu_s, t_p) \tag{1}$$

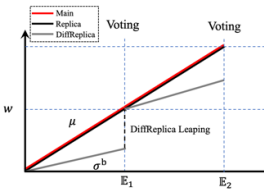

Figure 1: Case of No SE

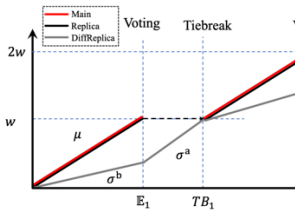

Figure 2: Case of SE and Wait for DiffReplica to Catch Up

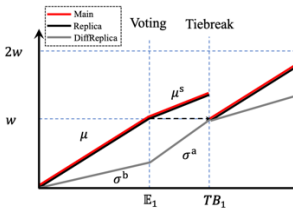

Figure 3: Case of SE and Speculative Execution of the Incorrect Replica

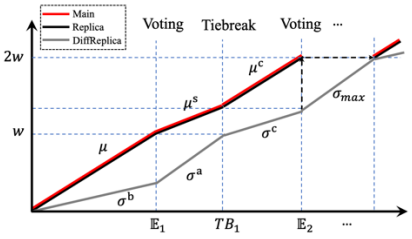

Figure 4: Case of SE and Speculative Execution of the Correct Replica but Disagreed at Second Voting Point

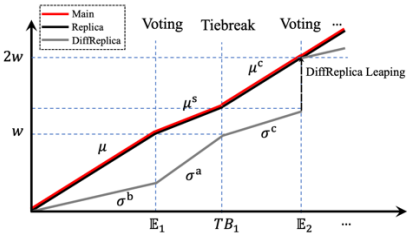

Figure 5: Case of SE and Speculative Execution of the Correct Replica and Agreed at Second Voting Point

p_f is the likelihood of SE for the process that the computational state is selected to use for speculative execution, and $2E^d(\mu_s, t_p)$ is the dynamic energy consumed by the two processes when performing the speculative execution. It can be expressed as $E^d(\mu_s, t_p) = \gamma \times \mu_s^\alpha \times t_p$, where γ is the unit of energy consumption per unit time depends on the processor's design, and α is the parameter that represents the relationship between power and the execution rate. Normally, power is proportional to the cube of execution rate. t_p is the time used for the speculative execution, which is same as the time used by the diffReplica to catch up with its main process. It is calculated as $t_p = \frac{w - \sigma^b \times \frac{w}{\mu}}{\sigma^a}$, where $w - \sigma^b \times \frac{w}{\mu}$ is the workload left for the diffReplica to catch up with its main process at execution rate σ^a. The estimated time delay is calculated as:

$$\bar{T}_{delay} = (1 - p_f) \times t_p \tag{2}$$

To calculate these penalties, we need to first obtain the likelihood of SE. In this work, we used an online failure prediction model that is based on exponential smoothing. This model can dynamically learn and predict the failure likelihood and robust enough to be not influenced by sudden changes in the data. The predicted likelihood of SE is obtained by the following equation:

$$\tilde{p}(\tau_i) = (1 - w^p(\tau_i))\tilde{p}(\tau_{i-1}) + w^p(\tau_i)p(\tau_i) \tag{3}$$

where $p(\tau_i)$ is the observed failure likelihood, and $\tilde{p}(\tau_{i-1})$ is the previous predicted failure likelihood. $w^p(\tau_i)$ is the learning weight, and it is defined as $w^p(\tau_i) = c^p \frac{ep^2(\tau_i)}{\sigma p(\tau_i)}$, where $0 < c_p < 1$ and $0 < w^p(\tau_i) < 1$. $ep(\tau_i)$ is the prediction error function, which is calculated as $ep(\tau_i) = p(\tau_i) - \tilde{p}(\tau_i)$. $\sigma p(\tau_i)$ is the square prediction error. It is learned by a second order exponential smoothing as $\sigma p(\tau_i) = c_p ep^2(\tau_i) + (1 - c_p)\sigma p(\tau_{i-1})$. Therefore, in the case of sudden changes in the data, the prediction results will not be significantly affected.

The workload for each interval, w, may be affected by the potential communication between processes. Therefore, the actual workload may vary among all the intervals. To have an estimate of the actual workload to calculate the time delay and wasted energy consumption, we used exponential smoothing to predict the future workload $\omega(\tau_i)$. Following is the formula of the model.

$$\tilde{\omega}(\tau_i) = (1 - w^l(\tau_i))\tilde{\omega}(\tau_{i-1}) + w^l(\tau_i)\omega(\tau_i) \tag{4}$$

$w^l(\tau_i)$ is defined as $c^l \frac{el^2(\tau_i)}{\sigma l(\tau_i)}$, where $el(\tau_i)$ is the prediction error function. Similar to the failure likelihood prediction, this model also used second order exponential smoothing (the square prediction error, $\sigma l(\tau_i)$) to ensure that it is not affected by sudden changes in the data, denoted as $\sigma l(\tau_i) = c_l el^2(\tau_i) + (1 - c_l)\sigma l(\tau_{i-1})$.

3.3 Cost Function

To fairly compare these two costs, we convert them to the same unit for comparison. The process of the decision making is depicted in Figure 6. Therefore, we need additional cost functions that convert the cost of $\bar{E}_{waste}$ and $\bar{T}_{delay}$ to the same unit. The cost of $\bar{E}_{waste}$ is the cost of money paid for additional energy consumption. The cost of $\bar{T}_{delay}$ is the revenue lost due to longer execution time than expected. These costs may be varied depend on the cost function. The cost of $\bar{E}_{waste}$ could be maxed out after exceeded a certain threshold of usage. This is also applied to revenue reduction. It also can be a linear increment function as each unit of $\bar{E}_{waste}$ is counted towards the cost with the same price. Also, another possibility is that the price of a unit of energy increases or decreases as the $\bar{E}_{waste}$ increases. We use an exponential or logarithmic increment function to represent this case. In this paper, we evaluate diffReplication with linear cost function, exponential cost function, logarithmic cost function, and cost function with a cap, where the cost is maxed out when exceeding the cap.

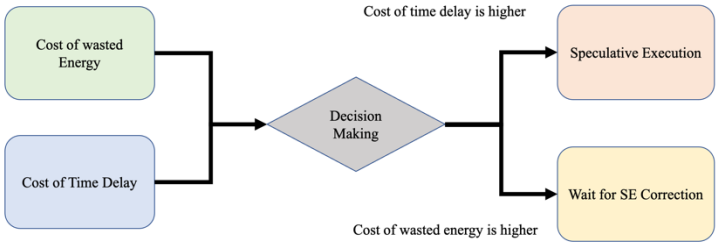

Figure 6: Decision Making Process Given Costs of Energy Wasted and Time Delay

As the cost may be directly proportional to the energy consumption or time to completion, we considered linear cost function as $Cost^{L}(x,\gamma) = \gamma\, x$, where γ is the price and x is either the time delay or wasted energy incurred by the scheme.

The cost of energy wastage or time delay may also be exponential or logarithmic based on the nature of the application and the required quality of services. To this end, we also investigate the performance of the proposed model, assuming an exponential cost function, $Cost^{E}(x,\gamma) = e^{\gamma\, x}$, where γ is the coefficient in the exponent, and a logarithmic cost function, $Cost^{Ln}(x,\gamma) = \ln \gamma\, x$.

In addition, the cost function under cost cap is expressed in (5), where γ is the coefficient, β is the maximum cost, assuming a budget cap, cap.

$$Cost^{Ecap}\,(x,\gamma,\beta,cap) = \begin{cases} e^{\gamma(x)}, & x < cap \\ \beta, & otherwise \end{cases} \tag{5}$$

The above is a representation of the scenario where the energy cost is capped, and the cost can be reduced if the energy is saved.

4 Experimental Evaluation

To evaluate the proposed model, we conducted an experimental evaluation in a simulated HPC environment, where the nodes have different failure rates. It should be noted that a single SE may propagate from one process to other processes via message passing. Therefore, if SE has occurred, all the processes are required to be corrected. To tolerate SE, messages among mains are not shared with replicas and diffReplicas to maintain isolation; a condition that is also applied to messages among replicas and diffReplicas. In addition, as processes may be blocked due to waiting for messages from slow processes, same-type processes are required to execute at the same rate. To this end, it is reasonable to model all mains as one, including the replicas and diffReplicas.

We use CSIM19 (C++ version) to conduct the simulation [Schwetman, 2001]. To simulate the SE of a process, a random number is drawn from an exponential distribution with the mean set to the processor's MTBF. This random number represents the time of SE occurrence. It occurs only if the time is earlier than the voting point. The simulation has the same assumption of the expected value calculation, as stated in the previous section.

In this experiment, we simulate a system with one million available processors, which is the minimum scale towards extreme-scale system based on current technology. As each main requires two more processes for SE detection and correction, only a third of the processors could be used to host main processes. Therefore, there are 333333 mains in our simulation. It is the best practice to use the reliable processors to host the main processes to minimize the failure likelihood. Therefore, all the main processes are executed on reliable processors with an average of 10 years MTBF. We varied the MTBF of the replicas from 5 to 10 years to simulate different reliability scenarios. All the diffReplicas execute on processors with an average of 5 years MTBF. The work for each epoch, w, is set to the workload that could be completed with the maximum execution rate of 0.1 hours on average. The actual work that was completed within an interval may be varied due to factors such as communication delay and computation in different scale. We randomly varied the actual workload from 0.05 to 0.15 hours to simulate different scenarios. There are 1000 (n) epochs of work to simulate long run tasks. The main and replica's execution rates, μ, μ_s, and μ_c, are set to 100%, 30%, and 100% of the maximum execution rate, respectively. The diffReplica's execution rates, σ^b and σ^a are set as 10% and 100% of the maximum execution rate, respectively. The energy consumption and time for leaping and SE correction are all set to 0.01 as a small fraction of the average workload within each interval. c_p, c_l in the failure likelihood predictor and workload predictor are set to 0.25. The initial failure likelihood $\tilde{p}(\tau_i)$ is set as 0.5. The initial prediction of the workload is set as 0.05. The experiment compared speculative execution, diffReplication, and no speculative execution approaches.

To demonstrate the advantage of diffReplication, we conducted sensitivity analyses on different failure rates, speculative execution rates, and parameters of cost functions. For each analysis, we simulated the execution 1000 times, and reported the average values in different scenarios and standard deviations as error bars in the figures.

We show energy consumption of one task's execution in the experiment, as each main process has a similar execution rate as other mains. It is the same story for the replicas, and diffReplicas.

4.1 Sensitivity Analysis of Different Failure Rates on Replicas

In this analysis, we focus on how different failure likelihoods impact the performance of the proposed model. In this experiment, we use linear cost function for energy consumption and time to completion, with the coefficients as 0.1 and 4.0, respectively. Figure 7 and Figure 8 depict the energy consumption and completion time changes with the changes of the replica's MTBF, respectively. Considering that there are 1000 epochs of work in each execution, the number of SEs are similar with each run. Therefore, the standard deviations are marginal in Figures 7 and 8. If the replica's failure rate decreases (increment on MTBF), the energy consumption reduces as less SEs may occur during the execution in all the approaches. diffReplication has the energy consumption between no speculative execution and speculative execution. It is also the same case for completion time. With the reduction of the replica's failure rate, the completion time of all the approaches decrease. It is worth noting that diffReplication has less energy consumption compared to speculative execution, and it has less time to completion than no speculative execution. With the same setting, speculative execution reduces the time to completion, but it consumes more energy as the corrupted process may be picked for speculative execution and has to revert back to the previous voting point.

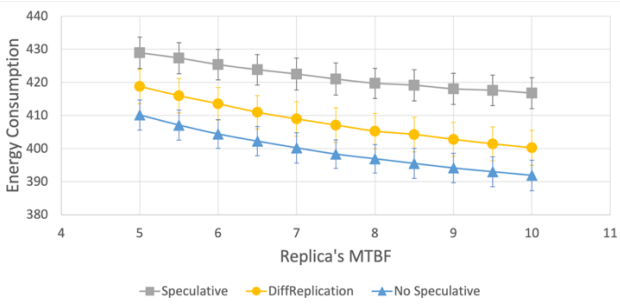

Figure 7: Sensitivity Analysis on Different Failure Rates for Replicas (Energy Consumption)

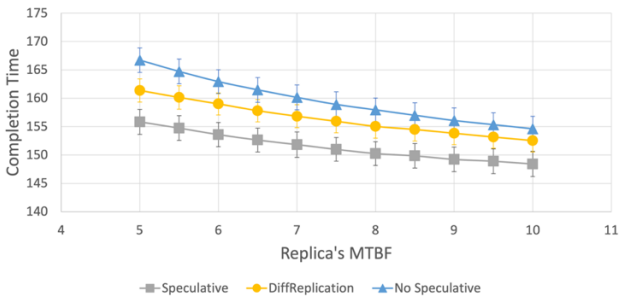

Figure 8: Sensitivity Analysis on Different Failure Rates for Replicas (Time)

4.2 Sensitivity Analysis of Different Speculative Execution Rates

In this analysis, we show how different speculative execution rates impact energy consumption and time to completion. As the proposed model always chooses the most reliable process for speculative execution, the success rate of choosing the correct process is higher than random. Therefore, with the increment of the speculative execution rate, the time to completion decreases, as depicted in Figure 10. When the speculative execution rate is at maximum, the speculative execution approach has 13% time to completion reduction compared to no speculative execution approach. The energy consumption also decreases when the speculative execution rate increases, as depicted in Figure 9. It is because that lower time to completion implies that more work has been completed successfully through speculative execution. The energy consumption reaches the minimum when the speculative execution rate reaches half of the maximum. The energy consumption starts to increase after the half point. The reason for this is that when the execution rate increases, the energy consumption increases cubically. In this case, the energy wastage due to the incorrect choice for speculative execution has a greater impact on energy consumption. Therefore, the diffReplication dynamically determines that it may be more costly to perform speculative execution. Consequently, when the speculative rate increases, it is more likely to choose to not perform speculative execution.

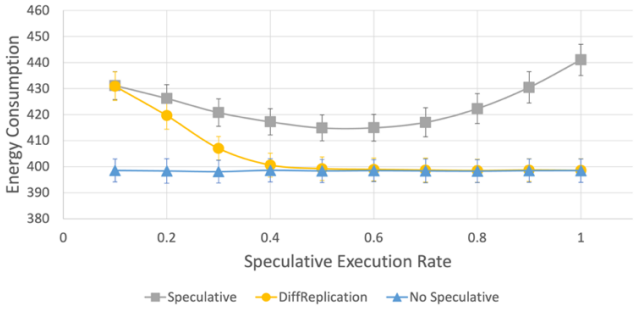

Figure 9: Sensitivity Analysis on Different Speculative Execution Rates (Energy Consumption)

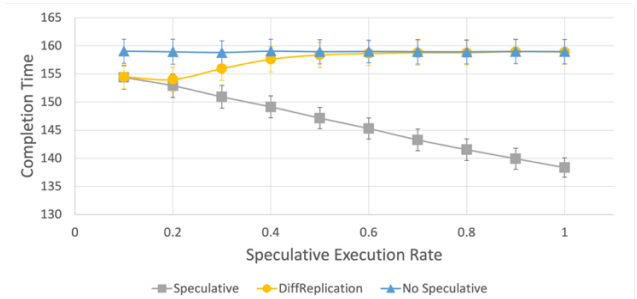

Figure 10: Sensitivity Analysis on Speculative Execution Rates (Time)

4.3 Sensitivity Analysis of Different Cost Function Setups

Other than the execution rate and hardware reliability, the cost function difference also has an impact on the performance of diffReplication. This analysis shows how diffReplication performs with different cost function setups. We compared different linear functions, exponential functions, and logarithmic functions. We also compared exponential function to exponential function with budgets to evaluate how the budget cap can influence the performance of diffReplication.

4.3.1 Linear to Linear Cost Function

In this analysis, we use the linear cost function for completion time and energy consumption. We fixed the coefficient of cost function for time delay as 0.1 and varied the coefficient for energy wastage. Figures 11 and 12 show the energy consumption and completion time with different linear coefficients for diffReplication. When the linear coefficient increases for the cost of energy wastage, the time to completion increases and the energy consumption linearly decreases.

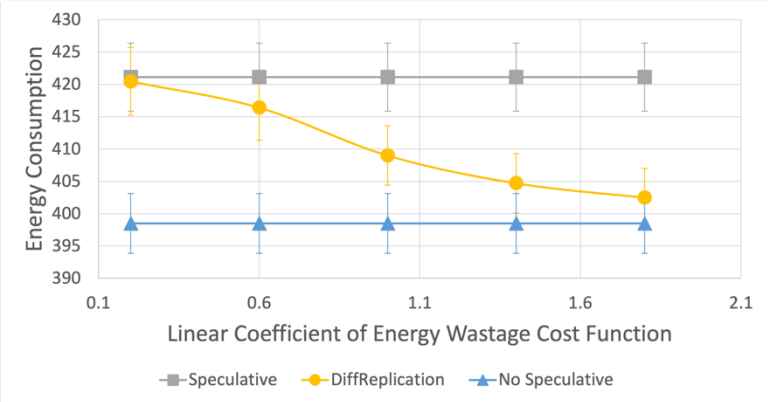

Figure 11: Sensitivity Analysis on Different Linear Rate for Cost Function of Time Delay (Energy Consumption)

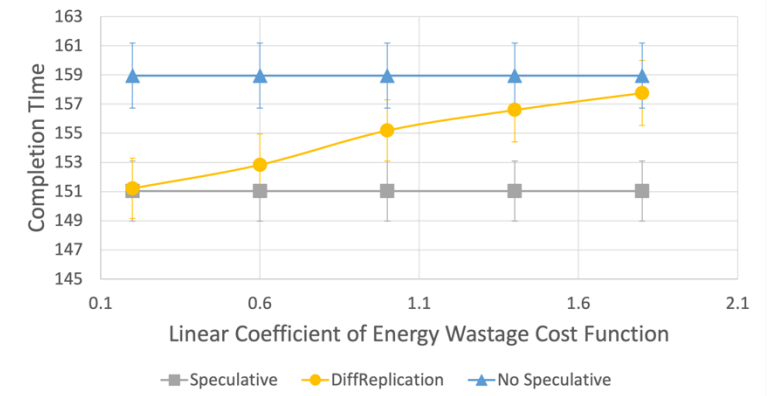

Figure 12: Sensitivity Analysis on Different Linear Rate for Cost Function of Time Delay (Time)

4.3.2 Exponential to Exponential Cost Function

In this analysis, we evaluated the diffReplication model with different exponential cost functions. We fixed the coefficient of the cost function for time delay as 5 and varied the coefficient in the cost function for wasted energy consumption. Figures 13 and 14 depict the energy consumption and time to completion changes. The energy consumption decreases, and the completion time increases when the coefficient increases, which means energy consumption is more costly when the coefficient increases. We also conducted the analysis that using logarithmic function as well as other combination of different functions. The results are similar with what is depicted in Figure 13 and 14.

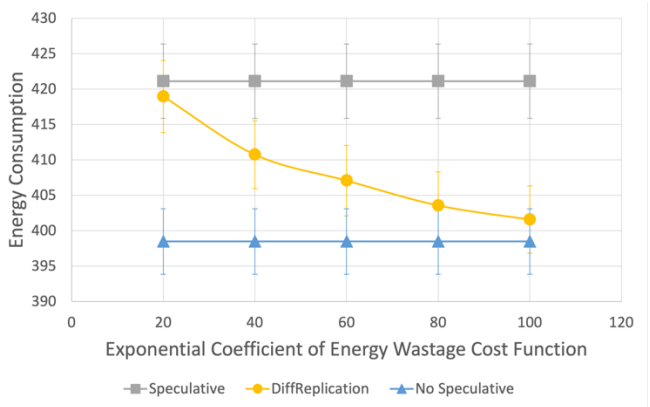

Figure 13: Sensitivity Analysis on Exponential Coefficient for Cost Function of Energy Wastage (Energy Consumption)

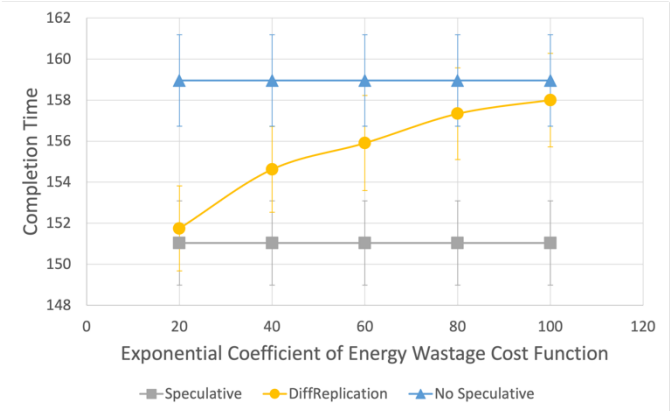

Figure 14: Sensitivity Analysis on Exponential Coefficient for Cost Function of Energy Wastage (Time)

4.3.3 Exponential to Capped Exponential Cost Function

In this analysis, we used exponential cost function for completion time and energy consumption. We also added a cap to the energy wastage so that if the energy wastage is higher than the defined cap, the cost of energy wastage will max out. We fixed the coefficient of the cost function for energy wastage and time delay as 150 and 5, respectively, and varied the cap for energy wastage. Figures 15 and 16 show the energy consumption and completion times with different caps for diffReplication. When the cap for the cost of energy wastage increases, the energy consumption starts to decrease and the time to completion starts to increase. If the cap is too small, diffReplication will prioritize improving completion time as energy wastage didn't relatively increases. These analyses clearly show that diffReplication can smartly choose if it is necessary to do speculative execution dynamically based on current cost estimations.

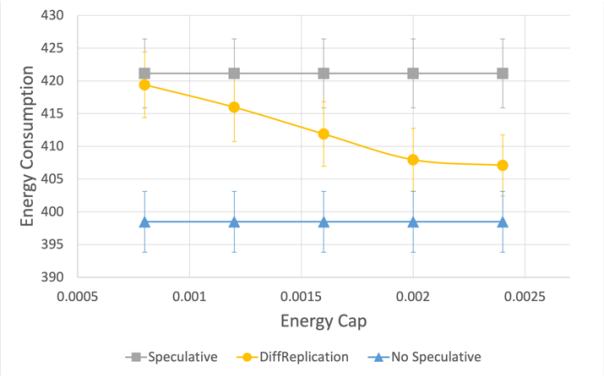

Figure 15: Sensitivity Analysis on Different Cap for Cost Function of Energy wastage (Energy Consumption)

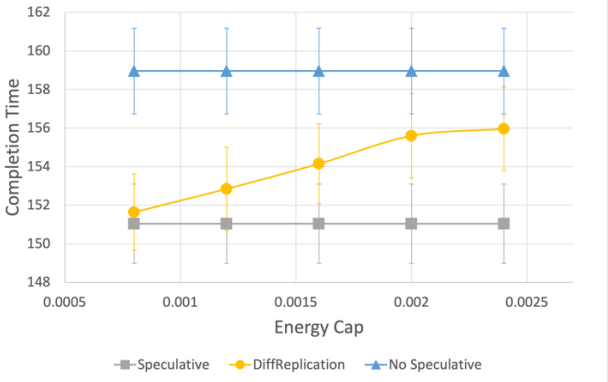

Figure 16: Sensitivity Analysis on Different Cap for Cost Function of Energy Wastage (Time)

5 Conclusion and Future Work

In future distributed systems, applications that require massive-scale parallel computing may be highly likely to encounter silent errors and produce incorrect results. In this work, we propose the diffReplication model that facilitates at least three processes for one task, where one process executes slowly and only speeds up when SE is detected to catch up and correct it. The key contribution of this work is the design of an energy-aware model, diffReplication, to tolerate silent failures, without incurring significant increase in time to completion of the underlying application. In order to reduce the waiting time required to reach consensus and determine correct output, the model uses an "intelligent" speculative execution method to determine which process should continue execution, without waiting the diffReplica to reach the synchronization barrier. Majority voting is achieved asynchronously and roll-back is only used when speculative execution selects the wrong process. The experiment results show that speculative execution could reduce the time to completion by a maximum of 13%, but with additional energy consumption. The experiment also shows that diffReplication can balance the energy consumption and time to completion to choose speculative execution when advantageous. The proposed model has the best performance in the case that energy saving is important but also maintaining an acceptable completion time is necessary, and vice versa.

The changing of execution rates for each process has the impact on the time to completion and energy consumption. How to find the optimal speculative execution rate that achieves the shortest completion time under power consumption remains a challenge. It is also to be noticed that the proposed approach assumes processors' MTBF does not change during the execution. If the failure rates can be dynamically determined by predictors, diffReplica's execution rates could be optimized based on the predicted failure rates to further reduce energy consumption and the time to completion.

Finally, this work assumes that each diffReplica is allocated a full processor core to achieve execution at maximum rate when consensus between the main and its replica is not achieved. In future work, we will explore virtualization-based approaches to execute multiple diffReplicas on a single core, in order to improve resource utilization. We also plan to explore the impact of dynamic, time-of-day dependent energy pricing schemes on the performance of the diffReplication model.

Acknowledgements

This research is based in part upon work supported by the Department of Energy under contract DE-SC0014376, and 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.

References

[Bartlett, 1981] Bartlett, J. F. (1981). A nonstop kernel. In ACM SIGOPS Operating Systems Review, pages 22–29, New York, NY, USA. ACM.

[Bosilca et al., 2015] Bosilca, G., Bouteiller, A., Herault, T., Robert, Y., and Dongarra, J. (2015). Composing resilience techniques: Abft, periodic and incremental checkpointing. International Journal of Networking and Computing, 5(1):2–25.

[Bougeret et al., 2012] Bougeret, M., Casanova, H., Robert, Y., Vivien, F., and Zaidouni, D. (2012). Using group replication for resilience on exascale systems. Rapport de recherche RR-7876, INRIA.

[Casanova et al., 2012] Casanova, H., Robert, Y., Vivien, F., and Zaidouni, D. (2012). Combining Process Replication and Checkpointing for Resilience on Exascale Systems. Rapport de recherche RR-7951, INRIA.

[Chen et al., 2018] Chen, C., Eisenhauer, G., Wolf, M., and Pande, S. (2018). Ladr: Low-cost application-level detector for reducing silent output corruptions. In Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, pages 156–167. ACM.

[Cui et al., 2018] Cui, X., Hussain, Z., Znati, T. and Melhem, R. (2018). A systematic fault-tolerant computational model for both crash failures and silent data corruption. In 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), pages 1-8. IEEE.

[Daly, 2003] Daly, J. (2003). A model for predicting the optimum checkpoint interval for restart dumps. In Int. Conf. on Computation Science, pages 3–12, Berlin, Heidelberg. Springer.

[Di and Cappello, 2016] Di, S. and Cappello, F. (2016). Adaptive impact-driven detection of silent data corruption for hpc applications. IEEE Transactions on Parallel and Distributed Systems, 27(10):2809–2823.

[Elliott et al., 2012] Elliott, J., Kharbas, K., Fiala, D., Mueller, F., Ferreira, K., and Engelmann, C. (2012). Combining partial redundancy and checkpointing for hpc. In 2012 IEEE 32nd International Conference on Distributed Computing Systems, pages 615–626, New York, NY, USA. IEEE.

[Elnozahy et al., 2002] Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv., 34(3):375–408.

[Fagg and Dongarra, 2000] Fagg, G.E. and Dongarra, J.J. (2000). FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In Recent Advances in Parallel Virtual Machine and Message Passing Interface: 7th European PVM/MPI Users' Group Meeting Balatonfüred, Hungary, September 10–13, 2000 Proceedings 7, pages 346-353. Springer Berlin Heidelberg.

[Fiala et al., 2012] Fiala, D., Mueller, F., Engelmann, C., Riesen, R., Ferreira, K., and Brightwell, R. (2012). Detection and correction of silent data corruption for large-scale high-performance computing. In SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pages 1–12. IEEE.

[Guermouche et al., 2011] Guermouche, A., Ropars, T., Brunet, E., Snir, M., and Cappello, F. (2011). Uncoordinated checkpointing without domino effect for send-deterministic mpi applications. In IPDPS, pages 989–1000, New York, NY, USA. IEEE.

[Hakkarinen and Chen, 2013] Hakkarinen, D. and Chen, Z. (2013). Multilevel diskless checkpointing. Computers, IEEE Transactions on, 62(4):772–783.

[Hargrove and Duell, 2006] Hargrove, P. and Duell, J. (2006). Berkeley lab checkpoint/restart (blcr) for linux clusters. J. Phys. Conf. Ser, 46(1):494.

[Huang et al., 2022] Huang, Y., Guo, S., Di, S., Li, G. and Cappello, F. (2022). Mitigating silent data corruptions in HPC applications across multiple program inputs. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pages 1-14. ACM.

[Kalaiselvi and Rajaraman, 2000] Kalaiselvi, S. and Rajaraman, V. (2000). A survey of checkpointing algorithms for parallel and distributed computers. Sadhana, 25(5):489–510.

[Krluku et al., 2019] Krluku, E. A., Gusev, M., and Zdraveski, V. (2019). Bi-source verification against silent data corruption in high performance computing. In Proceedings of the 9th Balkan Conference on Informatics, page 30. ACM.

[Lee et al., 2011] Lee, I., Basoglu, M., Sullivan, M., Yoon, D. H., Kaplan, L., and Erez, M. (2011). Survey of error and fault detection mechanisms. University of Texas at Austin, Tech. Rep, 11:12.

[Li et al., 2021] Li, L., Znati, T., Melhem, R., Differential Shadowing: A Resilience Framework for Extreme-scale, Heterogeneous Environments with Non-Uniform Node Failure Distribution. In 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC), pp. 1-8. IEEE, 2021.

[Losada et al., 2019] Losada, N., Bosilca, G., Bouteiller, A., González, P., and Martín, M. J. (2019). Local rollback for resilient mpi applications with application-level checkpointing and message logging. Future Generation Computer Systems, 91:450–464.

[Mills et al., 2013] Mills, B., Grant, R. E., Ferreira, K. B., and Riesen, R. (2013). Evaluating energy saving for checkpoint/restart. In First International Workshop on Energy Efficient Supercomputing (E2SC) in conjunction with SC13: The International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–8, New York, NY, USA. ACM.

[Mills et al., 2014] Mills, B., Znati, T. and Melhem, R. (2014). Shadow computing: An energy-aware fault tolerant computing model. In 2014 International Conference on Computing, Networking and Communications (ICNC), pages 73-77. IEEE.

[Mukherjee et al., 2005] Mukherjee, S. S., Emer, J., and Reinhardt, S. K. (2005). The soft error problem: An architectural perspective. In 11th International Symposium on High-Performance Computer Architecture, pages 243–247. IEEE.

[Peña et al., 2015] Peña, A. J., Bland, W., and Balaji, P. (2015). Vocl-ft: introducing techniques for efficient soft error coprocessor recovery. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, page 71. ACM.

[Riesen et al., 2010] Riesen, R., Ferreira, K., Stearley, J. R., Oldfield, R., III, J. L., Pedretti, K., and Brightwell, R. (2010). Redundant computing for exascale systems. No. SAND2010-8709. Sandia National Laboratories (SNL).

[Sachs, 2011] Sachs, S. (2011). Tools for exascale computing: challenges and strategies. In ASCR Exascale Tools Workshop.

[Schneider, 1990] Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys (CSUR), 22(4):299–319.

[Schwetman, 2001] Schwetman, H. (2001). Csim19: a powerful tool for building system models. In Proceeding of the 2001 Winter Simulation Conference (Cat. No. 01CH37304), volume 1, pages 250–255. IEEE.

[Stearley et al., 2012] Stearley, J., Ferreira, K., Robinson, D., Laros, J., Pedretti, K., Arnold, D., Bridges, P., and Riesen, R. (2012). Does partial replication pay off? In IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012), pages 1–6, New York, NY, USA. IEEE.

[Valiant, 1990] Valiant, L. G. (1990). A bridging model for parallel computation. Communications of the ACM, 33(8):103–111.

[Veronese et al., 2011] Veronese, G. S., Correia, M., Bessani, A. N., Lung, L. C., and Verissimo, P. (2011). Efficient byzantine fault-tolerance. IEEE Transactions on Computers, 62(1):16–30.