Department of Computer Science, UTSA

Technical Report: CS-TR-2007-001

Energy Efficient Redundant Configurations for Reliable Servers in Distributed Systems*

Dakai Zhu
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
dzhu@cs.utsa.edu

Rami Melhem and Daniel Mossé
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
{melhem, mosse}@cs.pitt.edu

Abstract

Modular redundancy and *temporal* redundancy are traditional techniques to increase system reliability. In addition to being used as temporal redundancy, with technology advancements, slack time can also be used by energy management schemes to save energy. In this paper, we consider the combination of modular and temporal redundancy to achieve energy efficient reliable service provided by multiple servers. We first propose an efficient *adaptive parallel recovery* scheme that appropriately processes service requests in parallel to increase the number of faults that can be tolerated and thus system reliability. Then we explore schemes to determine the *optimal redundant configurations* of the parallel servers to minimize system energy consumption for a given reliability goal or to maximize system reliability for a given energy budget. Our analysis shows that small requests, optimistic approaches, and parallel recovery favor lower levels of modular redundancy, while restricted large requests, pessimistic approaches and serial recovery favor higher levels of modular redundancy.

*A preliminary version of the paper appeared in EDCC 2005.

1 Introduction

The performance of modern computing systems has increased at the expense of drastically increased power consumption. In addition to embedded systems that are generally battery powered and have limited energy budget (e.g., complex satellite and surveillance systems), the increased power consumption has recently caught people's attention for large systems that consist of multiple processing units (e.g., data centers or web server farms) due to excessive heat dissipation and the associated complex packaging and cooling cost. Moreover, if the generated heat cannot be properly removed, it will also increase the system temperature and thus decrease system reliability.

Fault tolerance is an important requirement for reliable systems, and, in general, is achieved by exploring redundancy techniques. While modular redundancy can be used to detect and/or mask *permanent* faults by executing an application on several processing units in parallel, temporal redundancy can be explored to re-execute a faulty application due to *transient* faults and increase system reliability [21]. For extremely high reliability, we may combine modular and temporal redundancy. That is, when no majority result is obtained on modular redundant processing units during the processing of an application, we can further explore temporal redundancy to re-execute it and increase the reliability instead of declaring a failure [26].

In addition to temporal redundancy, slack time can also be used by *dynamic voltage scaling (DVS)* (a popular and widely used energy management technique) to scale down system processing speeds and supply voltages for saving energy [31, 33]. When more slack time is reserved as temporal redundancy for enhancing system reliability, less slack will be available for energy management. Therefore, there is an interesting trade-off between energy consumption and system reliability. Although fault tolerance through redundancy and energy management through DVS have been well studied separately, there is relatively less work addressing the combination problem of fault tolerance and energy management.

For systems where both lower levels of energy consumption and higher levels of reliability are important, managing the system reliability and energy consumption together is necessary.

A number of studies have reported that *transient faults* occur much more frequently than *permanent faults* [4, 12]. With the scaled technology size and reduced design margin for high performance, transient faults will become more prominent [1]. In this paper, focusing on tolerating transient faults, we consider event-driven reliable services (e.g., transactions for financial institutions and signal processing on satellites) that are processed redundantly on multiple servers in distributed system. For systems with fixed number of servers, higher levels of modular redundancy lead to larger and thus fewer *redundant server groups*, consequently needing more time to process a given workload and resulting in less available slack time. However, *redundant server groups* with higher levels of modular redundancy can mask and tolerate more faults, which may decrease the

need for temporal redundancy for the recovery of a given number of faults to be tolerated and leave more slack for energy management.

Another aspect is the static energy consumed by the system. For a given workload (i.e., the number of service requests in a fixed interval), considering the static leakage power consumed by each server, lower levels of modular redundancy may be more energy efficient and not all available servers will be used [5, 32].

In this paper, focusing on reliable services provided by parallel servers, we explore the *optimal redundant configuration* for the servers to either minimize system energy consumption for a given reliability goal (e.g., to tolerate a certain number of faults within the interval considered) or to maximize system reliability with a limit energy budget. To this end, we first propose an efficient *adaptive* parallel recovery scheme that appropriately processes service requests in parallel to increase the number of faults that can be tolerated and thus system reliability. Then, we extend our research results to combine the recovery schemes and redundancy techniques. In this context, an optimal redundant configuration specifies the level of modular redundancy employed, the number of active servers used (the unused servers are turned off for energy efficiency), the frequency at which the active servers run and the number of backup slots needed.

The remainder of this paper is organized as follows: below we first briefly describe related work. The system models and problem description are presented in Section 2. The recovery schemes are discussed in Section 3. The combination of modular redundancy and parallel recovery is addressed in Section 4. Section 5 presents two schemes to determine the optimal redundant configurations for the servers to minimize energy and maximize reliability, respectively. The analysis results are presented and discussed in Section 6 and Section 7 concludes the paper.

1.1 Related Work

The idea of trading processing speed for energy savings was first proposed by Weiser *et al.* in [31], where processor frequency (and corresponding supply voltage) is adjusted using utilization based predictions. Although the energy management for single-processor systems has been studied extensively based on *dynamic voltage scaling (DVS)* techniques, the energy management for parallel and distributed systems, where heat generated and cooling costs are big problems, has caught people's attention only recently.

For a given static schedule for real-time tasks running on distributed systems, Luo et al. proposed a static optimization algorithm by shifting the static schedule to re-distribute the static slack according to the average slack ratio on each processor element [16]. They improved the static optimization by using critical path analysis and task execution order refinement to get the maximum static slow down factor for each task [17]. In [20], Mishra et al. studied the slack distribution problem for a set of aperiodic real-time tasks in distributed

systems by considering the different levels of parallelism in the schedule, and the result is further extended by Hua and Qu [8]. In [18], Mahapatra and Zhao studied one slack distribution technique that was applied to each node and considers energy efficiency for the whole distributed system.

In [2], Bohrer *et al.* presented a case of managing power consumption in web servers. Elnozahy *et al.* evaluated policies that combine DVS and on/off techniques for cluster-wide power management in server farms [5, 15]. Considering the effects of static power and discrete processing frequencies, Xu *et al.* proposed schemes that adjust the number of active servers based on the system load [32], and the schemes are extended by considering QoS requirements in heterogeneous clusters [23]. Sharma *et al.* investigated adaptive algorithms for voltage scaling in QoS-enabled web servers to minimize energy consumption subject to service delay constraints [25].

When combining fault tolerance and energy management, for independent periodic tasks, using the primary/backup model, Unsal *et al.* proposed an energy-aware software-based fault tolerance scheme. The scheme postpones as much as possible the execution of backup tasks to minimize the overlap of primary and backup executions and thus to minimize energy consumption [30]. For Duplex systems (where two concurrent hardware platforms are used to run the same software for fault detection), Melhem *et al*. explored the optimal number of checkpoints, *uniformly* or *non-uniformly* distributed, to achieve minimum energy consumption [19]. Elnozahy *et al*. proposed an *Optimistic-TMR* (OTMR) scheme to reduce the energy consumption for traditional TMR (Triple Modular Redundancy, in which three hardware platforms are used to run the same software simultaneously to detect and mask faults) systems by allowing one processing unit to slow down provided that it can catch up and finish the computation before the deadline if there is a fault [6]. In [39], Zhu *et al.* further explored the optimal frequency setting for OTMR and presented detailed comparisons among Duplex, TMR and OTMR on reliability and energy consumption. Combined with voltage scaling techniques, Zhang *et al.* have proposed an adaptive checkpointing scheme to tolerate a fixed number of transient faults and save energy for serial applications [34]. The work was further extended to periodic real-time tasks in [35]. Considering the effects of energy management on fault rates [38], Zhu *et al.* explored the interplay between energy and reliability [36, 37].

However, for the servers in distributed systems, where both energy efficiency and reliability are important, no previous work addresses these issues simultaneously. To the best of our knowledge, this paper is the first work on such topic.

2 System Models and Problem Description

2.1 Power Model

The power in a server is mainly consumed by its processor, memory and the underlying circuits. For CMOS-based variable-frequency processors, the power consumption is dominated by dynamic power dissipation, which is cubicly related to the supply voltage and the processing speed [3]. As for memory, it can be put into different power-saving sleep states with different response times [14]. Therefore, for servers that employ variable speed processors [9, 29] and low power memory [22], the power consumption can be adjusted to satisfy different performance requirements. Although dynamic power dominates in most components, the static leakage power increases much faster than dynamic power with technology advancements and thus cannot be ignored [27, 28].

As mentioned before [31, 33], voltage is decreased when the CPU can be slowed down. In what follows, we use *frequency scaling* to stand for changing both processing frequency and supply voltage. To incorporate all power consuming components in a server, we adopt a simple system-level power model [38, 39] and assume that a server has three different states: *active*, *sleep* and *off*.

The system is in the *active state* when it is serving a request. The most static power is consumed in the active state; however, a request may be processed at different frequencies (and corresponding supply voltages) and consumes different dynamic power. The *sleep state* is a power saving state that removes all dynamic power and most of the static power. Servers in the sleep state can react quickly (e.g., in a few cycles) to new requests and the time to transit from sleep state to active state is assumed to be negligible.

A server consumes no power in the *off* state. Therefore, for a server running at frequency f, its power consumption can be modeled as [38, 39]:

$$P(f) = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{ef} f^m) \tag{1}$$

where P_s is the *sleep power*; P_{ind} and P_d are *frequency-independent* and *frequency-dependent* active powers, respectively. P_s is a constant and includes (but not limited to) the power to maintain basic circuits, keep the clock running and the memory in power saving sleep modes [14]. P_{ind} consists of part of memory and processor power as well as any power that can be efficiently removed by putting systems into sleep state(s) and is independent of system supply voltages and processing frequencies [9, 14], which is also a constant. P_d includes processors dynamic power and any power that depends on system processing frequencies and supply voltages [3].

From the above, $\hbar$ equals 1 if the server is *active* and 0 otherwise. C_{ef} and m are system dependent con-

stants. The maximum frequency-dependent active power corresponds to the maximum processing frequency f_{max} (and the highest supply voltage v_{max}) and is given by $P_d^{max} = C_{ef} f_{max}^m$. For convenience, the values of P_s and P_{ind} are normalized to P_d^{max}; that is, $P_s = \alpha \cdot P_d^{max}$ and $P_{ind} = \beta \cdot P_d^{max}$. Moreover, we assume that continuous frequency is used. For systems that have discrete frequencies, two adjacent frequencies can be used to emulate any frequency as discussed in [11].

Notice that less frequency-dependent active energy is consumed at lower frequencies; however, it takes more time to process a request and thus more sleep and frequency-independent active energy will be consumed. Therefore, considering the sleep power and frequency-independent active power, there is an energy efficient processing frequency at which the energy consumption to process a request is minimized [6, 7, 10, 13, 38, 39]. Considering the large timing overhead of turning on/off a server [2], we assume in this paper that the deployed servers are never turned off and the sleep power P_s is not manageable (i.e., always consumed). Thus, the *energy efficient frequency* can be easily found as [38, 39]:

$$f_{ee} = \sqrt[m]{\frac{\beta}{m-1}} \cdot f_{max} \tag{2}$$

Notice that f_{ee} is solely determined by the system's power characteristics and is independent of requests to be processed. If f_{low} is the lowest supported processing frequency, the minimum energy efficient frequency is defined as $f_{min} = max\{f_{low}, f_{ee}\}$. That is, the CPU may be set to a frequency higher than f_{ee} to meet an application's deadline or to comply with the lowest frequency limitation. However, for energy efficiency, the CPU should never be set to a frequency below f_{ee}. Moreover, for the case of $\beta > m - 1$ (i.e., the ratio of frequency-independent active power P_{ind} over the maximum frequency-dependent active power P_d^{max} power is more than $m - 1$), we will have $f_{ee} > f_{max}$, which means that no DVS is necessary and all requests should be processed at the maximum frequency f_{max} to minimize the energy consumption. For simplicity, in what follows, we assume that $f_{low} \leq f_{ee} \leq f_{max}$ (i.e., $\beta \leq m - 1$).

2.2 Application Model

In general, the workload of an event-driven application can be specified by the average service request arrival rate, which is the average number of requests over the length of the interval considered. Although the length of each individual request may vary, we use a mean service time for all requests at a given frequency, which can be justified in the case of high performance servers where the number of requests is large and each individual request has relatively short length [25]. That is, we assume that requests have the *same* (average) size and need C *cycles* to be processed. For the case of large variations in request size, checkpointing techniques may be employed to break requests into smaller sections of the same size [19]. However, exploring checkpoints is

beyond the scope of this paper and we leave it as our future work.

Given that we are considering variable frequency processors, the number of cycles needed to process a request may also depend on the processing frequency [24]. However, with a reasonable size cache, C has been shown to have small variations with different frequencies [19]. For simplicity, we assume that C is a constant and is the mean number of cycles needed to process a request. Notice that, this is a conservative model. With fixed access time for memory and other I/O devices, the number of CPU cycles needed to process a request will actually decrease at lower frequencies.

For simplicity, the time needed to process one request at f_{max} is assumed to be $c = \frac{C}{f_{max}} = 1$ time unit. Moreover, to ensure responsiveness, we consider time intervals with length of D time units. All requests arriving in one interval should be processed during the next interval. That is, the response time for each request is no more than $2D$.

2.3 Fault Model

During the processing of a request, a fault may occur due to various reasons, such as hardware failures, electromagnetic interferences or software errors. Considering that *transient* faults occur much more frequently than *permanent* faults [4, 12], in this paper, we focus on transient faults and explore redundancy techniques to tolerate them. The requests are duplicated and executed concurrently on multiple servers (which form a *redundant server group*; e.g., Duplex or TMR) and results from these servers are compared to detect faults. If no majority result is obtained (e.g., more than one fault in a TMR, assuming that faults on different servers produce different results), we say a request is *faulty* and it needs to be re-executed.

2.4 Problem Definition

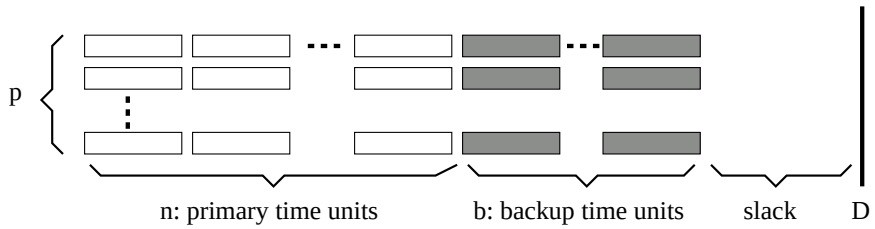


Figure 1: To achieve a k-fault tolerant system, p Duplexes are used to process w requests within a time interval of D. Here, b time units are reserved as backup slots.

For a distributed system that consists of M *identical* servers, suppose that p Duplex groups, where $2p \leq M$, are used to implement a *k-fault tolerant system*, which is defined as a system that can tolerate k faults within any interval D under *all* circumstances. Let w be the number of requests arriving within an interval D. Recall

that the processing of one request needs one time unit. Hence, $n = \lceil \frac{w}{p} \rceil$ time units are needed to process all the requests, without considering faults. For simplicity, we assume that the communication cost for the cooperation between servers is incorporated in the service time for the requests.

The schedule for processing all requests within the interval of D is shown in Figure 1. In the figure, each white rectangle represents a *section* that is used to process one request on a Duplex and the shadowed rectangles represent the *recovery sections* reserved for re-processing the faulty requests. To tolerate k faults in the worst case, a number of time units, b, have to be reserved as *backup slots*, where each backup slot has p recovery sections in parallel. For ease of presentation, the first n time units are referred to as *primary time units* and each white rectangle is called a *primary execution*. When a faulty request is being re-executed during a recovery section, another fault may happen. If all the recovery sections that process a given faulty request fail, then we say that there is a *recovery failure*, which could be further recovered using the remaining recovery sections.

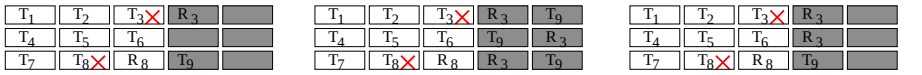
After scheduling the primary time units and backup slots, the amount of slack left is $D - (n + b)$, which can be used to scale down the processing frequency of servers and save energy.

For a given request arrival rate and a fixed time interval in an event-driven system that consists of M servers, where the performance and energy consumption of the servers are manageable, we focus on exploring the *optimal redundant configurations* of the servers (that is, the redundancy level of server groups, the number of server groups needed, the number of time units reserved for recovery, etc) to either (a) minimize energy consumption while achieving a k-fault tolerant system or (b) maximize the number of faults that can be tolerated with a limited energy budget.

3 Recovery with Parallel Backup Slots

In this section, we first present recovery schemes that work with parallel backup slots. For ease of discussion, we assume that servers are configured as duplexes. That is, the consequence of any single fault would be a faulty request, which will need a recovery section for re-processing. The case of servers being configured with higher levels of modular redundancy (e.g., TMR) will be addressed in Section 4.

In what follows, we calculate the worst case maximum number of faults that can be tolerated during the processing of w requests by p duplexes with the help of b backup slots. The addition of one more fault can cause an additional faulty request that can not be recovered and thus leads to a system failure. As a first step, we assume that the number of requests w is a multiple of p (i.e., $w = n \cdot p$, $n \geq 1$). The case of w being not a multiple of p will be discussed in Section 3.4. For different strategies of using backup slots, we consider three recovery schemes: *restricted serial recovery*, *parallel recovery* and *adaptive parallel recovery*.



a. Restricted serial recovery b. Parallel recovery c. Adaptive parallel recovery

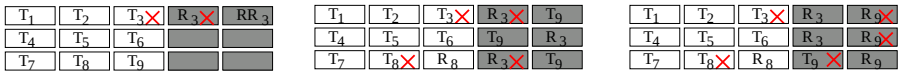
Figure 2: Different recovery schemes. '$\times$' represents faulty requests, and shaded rectangles are the recovery sections.

Figures 2 and 3 will be used below to illustrate the difference between the recovery schemes. Assume there are nine requests to be processed on three duplexes. The requests are labeled T_1 to T_9 and there are two backup slots (i.e., six recovery sections). Suppose that requests T_3 and T_8 become faulty on the top duplex during the third time unit and the bottom duplex during the second time unit, respectively. Note that request T_8 is recovered *immediately* during the third time unit (R_8) and the processing of request T_9 is postponed.

Therefore, at the end of the third time unit, there are two requests to be processed/re-processed: the original request T_9 and the recovery request R_3.

3.1 Restricted Serial Recovery

The restricted serial recovery scheme limits the re-processing of a faulty request to the *same* duplex. For example, Figure 2a shows that, R_3, the recovery of T_3, is performed on the top duplex while T_8 is recovered by R_8 on the bottom duplex.



a. Restricted serial recovery b. Parallel recovery c. Adaptive parallel recovery

Figure 3: The maximum number of faults that can be tolerated by different recovery schemes in the worst case scenario.

It is easy to see that, with b backup slots, the restricted serial recovery scheme can only recover from b faults (during primary and backup execution) in the worst case scenario. For example, as shown in Figure 3a, if there is a fault that causes the request T_3 to be faulty during primary execution, we can only tolerate one more fault in the worst case when the fault causes T_3's recovery, R_3, to be faulty. One additional fault could cause the second recovery RR_3 to be faulty and lead to a system failure since there is no additional backup slot and the recovery of the faulty requests is restricted to the same duplex.

3.2 Parallel Recovery

If faulty requests can be re-processed on multiple duplexes in parallel, we can allocate multiple recovery sections of the same backup slot to recover one faulty request *concurrently*. After finishing processing requests during the primary time units, the *parallel recovery* scheme considers all recovery sections within the backup slots and *equally* allocates them to the remaining requests to be recovered/executed. For the above example, there are six recovery sections in total and each of the remaining requests R_3 and T_9 gets three recovery sections. The schedule is shown in Figure 2b. As mentioned early, the recovery of a faulty request (e.g., T_8) during the primary time units is performed immediately after the faulty request and only on the same duplex.

Suppose that there are i faults during primary execution and i requests remain to be processed at the beginning of the backup slots. With $b \cdot p$ recovery sections in total, each remaining request will get at least $\lfloor \frac{b \cdot p}{i} \rfloor$ recovery sections. That is, at most $\lfloor \frac{b \cdot p}{i} \rfloor - 1$ additional faults can be tolerated. Therefore, when there are i faults during primary execution, the number of additional faults during the backup execution that can be tolerated by parallel recovery is:

$$PR(b, p, i) = \left\lfloor \frac{b \cdot p}{i} \right\rfloor - 1 \tag{3}$$

Notice that, w ($= n \cdot p$) is the maximum number of faults that could occur during the n primary time units. That is, $i \leq n \cdot p$. Furthermore, we have $i \leq b \cdot p$ because it is not feasible for $b \cdot p$ recovery sections to recover more than $b \cdot p$ faulty requests. Therefore, $i \leq \min\{n \cdot p, b \cdot p\}$. Let $PR_{b,p}$ represent the maximum number of faults that can be tolerated by p duplexes with b backup slots in the worst case scenario. Hence:

$$PR_{b,p} = \min_{1 \leq i \leq \min\{b \cdot p, n \cdot p\}} \{i + PR(b, p, i)\} \tag{4}$$

Differentiating Equation 4 and considering the feasible range of i, simple algebraic manipulations show that the value of $PR_{b,p}$ can be obtained when $i = \min\left\{\lfloor \sqrt{b \cdot p} \rfloor, n \cdot p\right\}$ and/or $i = \min\left\{\lfloor \sqrt{b \cdot p} \rfloor + 1, n \cdot p\right\}$ depending on the floor operation in Equation 3. For the example in Figure 2, we have $PR_{2,3} = 4$ when $i = 2$ (illustrated in Figure 3b) or $i = 3$. That is, for the case shown in Figure 3b, two more faults can be tolerated in the worst case scenario and we can achieve a 4-fault tolerant system. One additional fault could cause the third recovery section for R_3 to be faulty and lead to a system failure. Notice that, although T_9 is processed successfully during the first backup slot, the other two recovery sections in the second backup slot that are allocated to T_9 can not be used by R_3 due to the *fixed* recovery schedule.

3.3 Adaptive Parallel Recovery

Instead of considering all recovery sections together and fixing the recovery schedule, we can use *one* backup slot *at a time* and *adaptively* allocate the recovery sections to improve the performance and tolerate more faults. For example, as shown in Figure 2c, we first use the three recovery sections in the first backup slot to process the remaining two requests. Here, the recovery R_3 is processed on two duplexes and request T_9 on one duplex. If the duplex that processes T_9 happens to encounter a fault while R_3 completes successfully, the recovery R_9 can be processed using *all* recovery sections in the second backup slot on all three duplexes, thus allowing two additional faults as shown in Figure 3c. Therefore, a 5-fault tolerant system is achieved. Compared to the simple parallel recovery scheme, one more fault could be tolerated.

In general, since there are p recovery sections in each backup slot, we can use one backup slot to process/recover up to p requests. Suppose that there are i requests remaining to be processed/recovered before using the backup slots. If $i > p$, the first p requests will be processed/recovered during the first backup slot, one on each duplex, and the rest of the requests and any *newly* faulty recovery during the first backup slot will be processed/recovered on the following $b-1$ backup slots. If $i \leq p$, requests are processed redundantly using a *round-robin* scheduler. In other words, for the first $p - i\lfloor\frac{p}{i}\rfloor$ requests, each of them is processed on $\lfloor\frac{p}{i}\rfloor + 1$ duplexes redundantly; for other requests, each of them is processed on $\lfloor\frac{p}{i}\rfloor$ duplexes concurrently.

Assuming that z requests need to be processed after the first backup slot, then the same recovery algorithm that is used in the first backup slot to process i requests is used in the second backup slot to process z requests; and the process is repeated for all b backup slots.

With the adaptive parallel recovery scheme, let $APR_{b,p}$ be the worst case maximum number of faults that can be tolerated using b backup slots on p duplexes. $APR_{b,p}$ can be calculated by considering different number of faults, i, occurred in the primary execution and estimating the corresponding number of additional faults allowed in backup slots in the worst case, $APR(b,p,i)$, and then taking the minimum over all values of i. Recall that $i \leq \min\{n \cdot p, b \cdot p\}$. We have:

$$APR_{b,p} = \min_{1 \leq i \leq \min\{b \cdot p, n \cdot p\}} \{i + APR(b,p,i)\} \tag{5}$$

where i is the number of faults during the primary execution; $APR(b,p,i)$ is the maximum number of additional faults that can be tolerated during b backup slots given the worst case distribution of the faults, which can be found iteratively as shown below:

$$APR(1,p,i) \quad = \quad \left\lfloor \frac{p}{i} \right\rfloor - 1 \tag{6}$$

$$APR(b,p,i) \quad = \quad \min_{x(i)\le J\le y(i)} \{J + APR(b-1,p,z(i,J))\} \tag{7}$$

When there is a single backup slot (i.e., $b = 1$), Equation 6 says that the maximum number of additional faults that can be tolerated in the worst case is $\lfloor \frac{p}{i} \rfloor - 1$. That is, one more fault could cause a recovery failure that leads to a system failure since at least one request is processed on $\lfloor \frac{p}{i} \rfloor$ duplexes.

For the case of multiple backup slots (i.e., $b > 1$), in Equation 7, J is the number of faults occurred during the first backup slot and $z(i, J)$ is the number of requests that still need to be processed during the remaining $b-1$ backup slots. We search all possible values of J and the minimum value of $J + APR(b-1, p, z(i, J))$ is the worst case maximum number of additional faults that can be tolerated during b backup slots. The bounds on J, $x(i)$ and $y(i)$, depend on i, the number of requests that need to be processed during b backup slots. When deriving the $x(i)$ and $y(i)$ bounds, we consider two cases, namely $i > p$ and $i \le p$.

When $i > p$, we have enough requests to be processed and the recovery sections in the first backup slot are used to process p requests (each on one duplex). When J ($0 \le J \le p$) faults happen during the first backup slot, the total number of requests that remain to be processed during the remaining $b - 1$ backup slots is $z(i, J) = i - p + J$. Since we should have $z(i, J) \le (b - 1)p$, then J should not be larger than $b \cdot p - i$. That is, when $i > p$, we have $x(i) = 0$, $y(i) = \min\{p, b \cdot p - i\}$ and $z(i, J) = i - p + J$.

When $i \le p$, all requests are processed during the first backup slot and each of the requests is processed on at least $\lfloor \frac{p}{i} \rfloor$ duplexes. To get the maximum number of faults that can be tolerated, at least one recovery failure is needed during the first backup slot such that the remaining $b - 1$ backup slots can be utilized. Thus, the lower bound for J, the number of faults during the first backup slot, is $x(i) = \lfloor \frac{p}{i} \rfloor$. Therefore, $\lfloor \frac{p}{i} \rfloor = x(i) \le J \le y(i) = p$. When there are J faults during the first backup slot, the maximum number of recovery failures in the worst case is $z(i, J)$, which is also the number of requests that need to be processed during the remaining $b - 1$ backup slots. From the adaptive parallel recovery scheme and considering the integer property of the number of fauts to be recovered, we get:

$$z(i,J) = \begin{cases} \left\lfloor \frac{J}{\lfloor p/i \rfloor} \right\rfloor & \text{if } \lfloor \frac{p}{i} \rfloor \le J \le (i - p + i\lfloor \frac{p}{i} \rfloor)\lfloor \frac{p}{i} \rfloor; \\ (i - p + i\lfloor \frac{p}{i} \rfloor) + \left\lfloor \frac{J - (i-p+i\lfloor \frac{p}{i} \rfloor)\lfloor \frac{p}{i} \rfloor}{\lfloor p/i \rfloor + 1} \right\rfloor & \text{if } (i - p + i\lfloor \frac{p}{i} \rfloor)\lfloor \frac{p}{i} \rfloor < J \le p. \end{cases} \tag{8}$$

For the example in Figure 2c, applying the above equations, we can get $APR(2, 3, 1) = 5$. That is, if there is only one fault during the primary execution, it can tolerate up to 5 faults since all 6 recovery sections will be redundant. Similarly, $APR(2, 3, 2) = 3$ (illustrated in Figure 3c), $APR(2, 3, 3) = 2$, $APR(2, 3, 4) = 1$, $APR(2, 3, 5) = 0$ and $APR(2, 3, 6) = 0$. Thus, from Equation 5, $APR_{2,3} = \min_{i=1}^{6}\{i + APR(2, 3, i)\} = 5$.

3.4 Arbitrary Number of Requests

We have discussed the case where the number of requests, w, in an interval is a multiple of p, the number of duplexes employed. In this section we extend the results to the case where w is *not* a multiple of p. Without loss of generality, suppose that $w = n \cdot p + d$, where $n \geq 0$ and $0 < d < p$. Thus, processing all requests will need $(n + 1)$ primary time units.

However, the last primary time unit is not fully scheduled with requests. If we consider the last primary time unit as a backup slot, there will be $b + 1$ backup slots and *at least* d requests need to be processed after finishing the execution in the first n time units. That is, we *pretend* to have $b + 1$ backup slots and treat the last d requests that cannot be scheduled within the primary time units as faulty requests. Therefore, the minimum number of faulty requests to be processed is d and the maximum number of faulty requests is $\min\{w, (b + 1) \cdot p\}$. Thus, similar to Equations 3 and 5, for the parallel recover and the adaptive parallel recovery schemes, the worst case maximum number of faults that can be tolerated with b backup slots can be obtained as:

$$PR_{b+1,p} = \min_{d \leq i \leq \min\{w,(b+1)\cdot p\}} \{i + PR(b + 1, p, i)\} \quad (9)$$

$$APR_{b+1,p} = \min_{d \leq i \leq \min\{w,(b+1)\cdot p\}} \{i + APR(b + 1, p, i)\} \quad (10)$$

where i is the number of requests to be processed on $b+1$ backup slots. $PR(b+1, p, i)$ and $APR(b+1, p, i)$ are defined as in Equations 3 and 6-7, respectively.

3.5 Worst Case Maximum Number of Tolerated Faults

To illustrate the performance of different recovery schemes, we calculate the worst case maximum number of faults that can be recovered by p duplexes with b backup slots under different recovery schemes. Assuming that the number of requests w is a multiple of p and is more than the number of available recovery sections, Table 1 gives the worst case maximum number of faults that can be tolerated by a given number of duplexes with different numbers of backup slots under the *parallel* and *adaptive* parallel recovery schemes. Recall that, for the *restricted serial recovery* scheme, the number of faults that can be tolerated in the worst case is the number of available backup slots b (that is, the first row in the table) and is independent of the number of duplexes that work in parallel.

Intuitively, for given p and b, the parallel recovery scheme *should* be able to recover from more faults than the restricted serial recovery scheme in the worst case scenario. However, from the table, we can identify that the number of faults that can be tolerated by the parallel recovery scheme may be less than what can be

Table 1: The worst case maximum number of faults that can be tolerated by p duplexes with b backup slots.

b		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p = 4$	parallel	3	4	6	7	8	8	9	10	11	11	12	12	13	14	**14**
	adaptive	3	6	10	14	18	22	26	30	34	38	42	46	50	54	58
$p = 8$	parallel	4	7	8	10	11	12	14	15	16	16	17	18	19	20	20
	adaptive	4	10	17	24	31	**39**	**47**	**55**	**63**	**71**	**79**	**87**	**95**	**103**	**111**

tolerated by the restricted serial recovery scheme. For example, with $p = 4$, the restricted serial recovery scheme can tolerated 15 faults when $b = 15$. However, the parallel recovery can only tolerate 14 faults. The reason comes from the unwise decision of fixing allocation of all recovery slots under the parallel recovery scheme, especially for larger number of backup slots.

When the number of backup slots equals 1, the two parallel recovery schemes have the same behavior and can tolerate the same number of faults. From the table, we can also see that the adaptive parallel recovery scheme is much more fault tolerant than the restricted serial recovery and the simple parallel recovery schemes; this advantage is more pronounced for the case of more duplexes working in parallel and larger numbers of backup slots. Interestingly, for the adaptive parallel recovery scheme, the number of faults that can be tolerated by p duplexes increases linearly with the number of backup slots b when b is greater than a certain value that depends on p. For example, with $p = 8$, if $b > 5$, the number of faults that can be tolerated using adaptive parallel recovery scheme increases by 8 when b is incremented. However, for $p = 4$, when $b > 2$, the number of faults increases by 4 when b is incremented.

4 Modular Redundancy and Parallel Recovery

Although we introduced recovery schemes with Duplex, the servers can be configured with any level of modular redundancy (e.g., NMR, $2 \leq N \leq M$; recall that M is the number of servers), which may consume different amounts of energy and tolerated different numbers of faults within the interval considered. In this section, we show how recovery schemes work in general with NMR.

For a general *a-out-of-N* NMR model, at least a servers in a NMR group need to get correct results to avoid a *NMR group failure*. If there is a NMR group failure, a request becomes faulty and needs to be re-executed by further exploring temporal redundancy. For servers being configured with NMR, each rectangle in Figures 1, 2 and 3 would correspond to the processing of a request on a NMR group (that is, each rectangle represents (N servers) and the results about the worst case maximum number of faults that can be tolerated in Section 3 will correspond to the number of NMR group failures.

Notice that, when higher levels of modular redundancy are used to detect and tolerate faults, the number of faults to be tolerated may not be the same as the number of faulty requests. For the *a-out-of-N* NMR

model, the processing of one request succeeds if no less than a servers get correct results. That is, in the worst case, $N - a + 1$ faults cause a NMR group failure and thus lead to a faulty request. Therefore, to obtain a k-fault tolerant system, we only need to recover $q_k = \left\lfloor \frac{k}{N-a+1} \right\rfloor$ faulty requests. Inversely, if q is the number of faulty requests that can be recovered, the number of faults that can be tolerated will be at least $k_q = q \cdot (N - a + 1) + N - a$. For example, if each rectangle in Figure 3c represents the execution of a task on a TMR and the *2-out-of-3* TMR model is used, the maximum number of faults that can be tolerated in the worst case will be $APR(2, 3) \cdot (N - a + 1) + N - a = 5(3 - 2 + 1) + 3 - 2 = 11$.

5 Optimal Redundant Configurations

In what follows, we consider two problems: (a) minimizing system energy consumption for a given reliability goal; and (b) maximizing system reliability for a given energy budget. In this context, an *optimal redundant configuration* specifies the level of modular redundancy employed, the number of servers used (the unused servers are turned off for energy efficiency), the frequency at which the active servers run and the number of backup slots needed.

5.1 Minimize Energy with Fixed Reliability Goal

For a system consisting of M servers, to tolerate k faults during the processing of all requests within the interval considered, we may use different redundant configurations which in turn consume different amounts of energy. When the system is configured with NMR, there are at most $\lfloor \frac{M}{N} \rfloor$ NMR groups available. Due to energy considerations [5, 32, 23], it may be more energy efficient to use fewer NMR groups than what is available and turn the unused servers off.

In Sections 3 and 4, we have shown how to compute the maximum number of faults, k, that can be tolerated by p NMR groups ($pN \leq M$) with b backup slots in the worst case. In this section, we use the same type of analysis for the inverse problem. That is, finding the *least* number of backup slots, b, needed by p NMR groups to tolerate k faults, which will maximize the amount of remaining slack for power management and thus minimize energy consumption.

For a given recovery scheme, let b be the minimum number of backup slots needed by p NMR groups to guarantee that any k faults can be tolerated. If we need more backup slots than the available slack units (i.e., $b > D - \left\lceil \frac{w}{p} \right\rceil$), it is not *feasible* for p NMR groups to tolerate k faults during the processing of all requests within the interval considered. Suppose that $b \leq D - \left\lceil \frac{w}{p} \right\rceil$, the amount of remaining slack time on each server is $slack = D - \left\lceil \frac{w}{p} \right\rceil - b$. Expecting that no faults will occur (i.e., being *optimistic*), the slack can be used to scale down the primary execution of requests while the recoveries are executed at the maximum frequency

f_{max} if needed. Alternatively, expecting that k_e ($\leq k$) faults will occur (i.e., k_e-*pessimism*) and assuming that b_e ($\leq b$) is the least number of backup slots needed to tolerate k_e faults, the slack time is used to slow down both primary and recovery executions during the first b_e backup slots. Here, $k_e = 0$ corresponds to optimistic approach while $k_e = k$ corresponds to pessimistic approach. The recovery execution during the remaining backup slots is executed at the maximum frequency, f_{max}, if more than k_e faults occur. Thus, the k_e-*pessimism expected energy consumption* is:

$$E(k_e) \quad = \quad p \cdot \left[P_s D + (P_{ind} + C_{ef} f^m(k_e)) \frac{\lceil w/p \rceil + b_e}{f(k_e)} \right] \tag{11}$$

where

$$f(k_e) \quad = \quad \min \left\{ \frac{\lceil w/p \rceil + b_e}{D - (b - b_e)}, f_{ee} \right\} \tag{12}$$

is the frequency to process all original requests and the recovery requests during the first b_e backup slots. Recall that f_{ee} is the minimum energy efficient frequency (see Section 2). Algorithm 1 summarizes the procedure to get the optimal redundant configuration for minimizing the expected energy consumption while tolerating k faults within the interval D.

Algorithm 1 The optimal redundant configuration for energy minimization

```
1: INPUT: w, D, M, α, β, m, k, k_e
2: E_min = M(α + β + 1)P_d^max D;
3: N^opt = −1; p^opt = −1; b^opt = −1;
4: for (N from 2 to M ) do
5:    for (p from ⌊M/N⌋ to 1) do
6:       b = 0; b_e = 0;
7:       while (Recovery(scheme, N, p, b) < k) do
8:          b = b + 1; /*see Equations 13, 14 and 15*/
9:       end while
10:      if (b ≤ D − ⌈w/p⌉) then
11:         while (Recovery(scheme, N, p, b_e) < k_e) do
12:            b_e = b_e + 1;
13:         end while
14:         Calculate E(k_e) from Equation 11;
15:         if (E(k_e) < E_min) then
16:            E_min = E(k_e); N^opt = N; p^opt = p; b^opt = b;
17:         end if
18:      end if
19:   end for
20: end for
21: return (N^opt, p^opt, b^opt).
```

First, the function $Recovery(scheme, N, p, b)$ (where *scheme* can be restricted **serial** recovery, **parallel** recovery and **adaptive** parallel recovery, respectively) is defined as follows to find the number of faults that

can be tolerated by p NMR groups using b backup slots with the assumption of a general *a-out-of-N* NMR model:

$$Recovery(\mathbf{serial}, N, p, b) = (N - a + 1) \cdot b + (N - a) \quad (13)$$

$$Recovery(\mathbf{parallel}, N, p, b) = (N - a + 1) \cdot PR_{b,p} + (N - a) \quad (14)$$

$$Recovery(\mathbf{adaptive}, N, p, b) = (N - a + 1) \cdot APR_{b,p} + (N - a) \quad (15)$$

By invoking the function $Recovery(\cdot)$, the algorithm finds the least number of backup slots b for p NMR groups to tolerate k faults using a given recovery scheme (lines 7 and 8). If b is larger than the available slack $D - \lceil \frac{w}{p} \rceil$ (line 10), p NMR groups are not feasible. Otherwise (from line 11 to 17), the least number of backup slots b_e for p NMR groups to tolerate k_e faults is obtained (lines 11 to 13) and the expected energy consumption is computed (line 14). Searching through all feasible numbers of NMR groups (line 5) and all possible values of N (line 4), we get the optimal redundant configuration to minimize the expected energy consumption (line 21). Notice that, finding the least number of backup slots to tolerate k faults has a complexity of $\mathbf{O}(k)$ (lines 7 and 11). Thus, the complexity of this algorithm is $\mathbf{O}(M^2k)$.

5.2 Maximize Reliability with Fixed Energy Budget

For the interval considered, when the energy budget is limited, we may not be able to power up all M servers at the maximum frequency during the entire interval. The more servers are employed, the lower the frequency at which the servers can run. For a given number of servers that run at a certain frequency, different levels of modular redundancy will result in different numbers of modular redundant groups and further lead to different maximum number of faults that can be tolerated within the interval D. In this section, we determine the optimal redundant configuration that maximizes the number of faults that can be tolerated with a fixed energy budget.

Notice that, from the power model discussed in Section 2, the most energy efficient solution is to scale down all the employed servers *uniformly* within the interval considered. With the length of the interval being D and with limited energy budget, E_{budget}, the average power level that a system can consume is:

$$P_{budget} = \frac{E_{budget}}{D} \quad (16)$$

When p NMR groups ($p \cdot N \leq M$) are deployed, the minimum power level is consumed when every server runs at the minimum energy efficient frequency f_{ee}. Thus, the minimum power level for p NMR groups is:

$$P_{min}(p, N) = p \cdot N(P_s + P_{ind} + C_{ef} f_{ee}^m) = p \cdot N \left(\alpha + \beta + \frac{f_{ee}^m}{f_{max}^m} \right) P_d^{max} \quad (17)$$

If $P_{min}(p,N) > P_{budget}$, p NMR groups are not feasible in terms of power consumption. Suppose that $P_{min}(p,N) \leq P_{budget}$, which means that the servers may run at a higher frequency than f_{ee}. Assuming that the frequency of the active servers that leads to P_{budget} is $f_{budget}(p,N)$, we have:

$$f_{budget}(p,N) = \sqrt[m]{\frac{P_{budget}}{p \cdot N \cdot P_d^{max}} - \alpha - \beta} \cdot f_{max} \tag{18}$$

Assuming that there will be no faults, the total time needed for processing all requests at frequency $f_{budget}(p,N)$ is:

$$t_{primary} = \frac{\lceil w/p \rceil}{f_{budget}(p,N)} \tag{19}$$

If $t_{primary} > D$, p NMR groups cannot finish processing all requests within the interval considered under the energy budget. Suppose that $t_{primary} \leq D$. We have $D - t_{primary}$ units of slack time and the number of backup slots that can be scheduled at frequency $f_{budget}(p,N)$ is:

$$b_{budget}(p,N) = (D - t_{primary}) f_{budget}(p,N) = D \cdot f_{budget}(p,N) - \left\lceil \frac{w}{p} \right\rceil \tag{20}$$

Notice that (see Sections 3 and 4), the worst case maximum number of NMR group failures that can be recovered by p NMR groups using the restricted serial recovery scheme is $b_{budget}(p,N)$. From Equations 9 and 10, the maximum number of NMR group failures that can be recovered within the interval considered is either $PR_{p,b_{budget}(p,N)}$ (for the parallel recovery scheme) or $APR_{p,b_{budget}(p,N)}$ (for the adaptive parallel recovery scheme). The corresponding maximum numbers of faults that can be tolerated in the worst case are:

$$k_{serial}(p,N) = b_{budget}(p,N)(N-a+1) + (N-a) \tag{21}$$

$$k_{parallel}(p,N) = PR(p, b_{budget}(p,N))(N-a+1) + (N-a) \tag{22}$$

$$k_{adaptive}(p,N) = APR(p, b_{budget}(p,N))(N-a+1) + (N-a) \tag{23}$$

where an *a-out-of-N* NMR model is assumed.

For a given recovery scheme, by searching all feasible combinations of p and N ($pN \leq M$), we can find the optimal redundant configuration that maximizes the worst case maximum number of faults to be tolerated within the interval D. The algorithm is similar to Algorithm 1 and is omitted for brevity.

6 Analytical Results and Discussion

In what follows, we provide some analytical results to show that the optimal redundant configuration that minimizes expected energy consumption for different number of faults to be tolerated or maximizes system

reliability with different energy budgets may deploy different levels of modular redundancy. Assuming that any two faults produce different errors, the *2-out-of-N* NMR model is used.

Notice that the power characteristics of servers also affect the optimal configurations. As shown by Elnozahy *et al.* in [6], lower static and leakage power prefers TMR instead of Duplex. That is, the energy consumed by the additional servers in TMR may be less than that consumed during the recovery time needed by Duplex for the same number of faults to be tolerated. For different values of α and β (which represent sleep power and frequency independent power, respectively), we have obtained the similar results. In what follows, we assume that $m = 3$, $f_{max} = 1$ and the maximum frequency-dependent power is $P_d^{max} = C_{ef} f_{max}^m = 1$. The values of α and β are assumed to be 0.1 and 0.3 respectively [39].

In our analysis, we vary the size of requests, request arrival rate (i.e., system load), the number of faults to be tolerated (k) and the recovery schemes to see how they affect the optimal redundant configuration. The interval considered is 1 second (i.e., worst case response time is 2 seconds) and three different request sizes at f_{max} are considered: $1ms$, $10ms$ and $50ms$.

For illustration purposes, we consider a system that consists of 12 servers. Faults are detected through duplicated processing. Thus the least level of modular redundancy is Duplex and there are at most 6 duplex groups. Let the *system load* be the ratio of the total number of requests arrived in one interval over the number of requests that can be handled by *one* server within one interval. For example, with 6 duplex groups, the maximum system load that can be handled is 6. To get enough slack for illustrating the variation of the optimal redundant configurations, unless specified otherwise, a system load of 2.6 is used. Recall that the interval considered is 1 second, different request arrival rates are used for different request sizes to obtain the system load of 2.6.

6.1 Optimal Configuration for Energy Minimization

Table 2 shows the level of modular redundancy employed (N) and the number of NMR groups used (p) for the optimal redundant configuration that tolerates a given number of faults k using different recovery schemes (the remaining $M - pN$ severs are turned off for energy efficiency). Here, $k_e = \frac{k}{2}$.

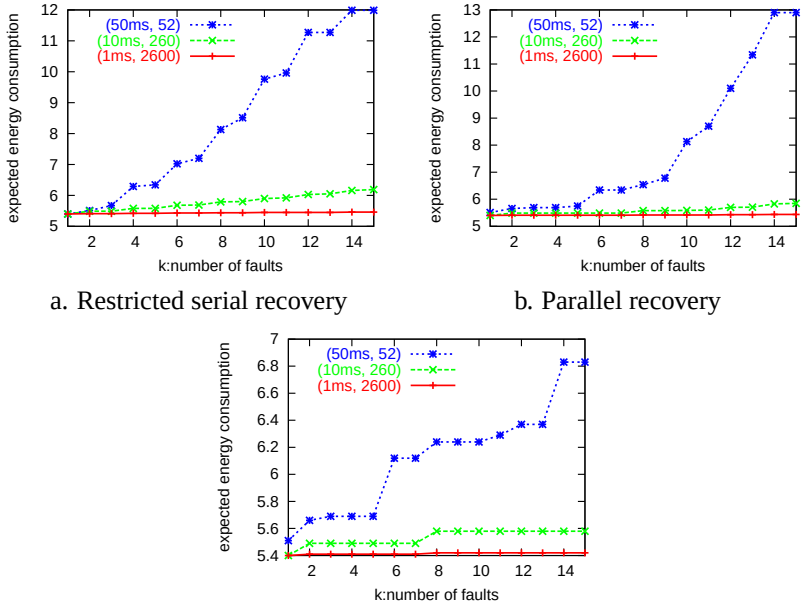
From the table, we can see that Duplex is the most energy efficient configuration in most cases. Moreover, smaller requests and the adaptive parallel recovery favor lower levels of modular redundancy while larger requests and the restricted serial recovery favor higher levels of modular redundancy, especially for larger number of faults to be tolerated (more than 11 in the table). Due to the effects of sleep power, the number of duplex groups needed does not increase monotonically when the number of faults increases, especially for the case of large request size (e.g., $50ms$) where more slack time is needed as temporal redundancy for the

Table 2: The optimal redundant configuration for different request sizes.

	k	2	4	6	8	10	12	14
	size,number	N, p	N, p	N, p	N, p	N, p	N, p	N, p
serial	$1ms$,2600	2, 4	2, 5	2, 5	2, 5	2, 5	2, 5	2, 5
	$10ms$,260	2, 5	2, 5	2, 5	2, 5	2, 5	2, 5	2, 5
	$50ms$,52	2, 4	2, 6	2, 6	2, 6	2, 6	**3, 4**	**3, 4**
parallel	$1ms$,2600	2, 4	2, 4	2, 4	2, 5	2, 5	2, 5	2, 5
	$10ms$,260	2, 5	2, 5	2, 5	2, 5	2, 5	2, 5	2, 5
	$50ms$,52	**2, 5**	**2, 5**	**2, 6**	**2, 5**	**2, 5**	**2, 5**	**3, 4**
adaptive	$1ms$,2600	2, 4	2, 4	2, 4	2, 4	2, 4	2, 5	2, 5
	$10ms$,260	2, 5	2, 5	2, 5	2, 5	2, 5	2, 5	2, 5
	$50ms$,52	**2, 5**	**2, 5**	**2, 4**	**2, 5**	**2, 5**	**2, 5**	**2, 6**

same number of backup slots

Figure 4 shows the corresponding expected energy consumption. Recall that the normalized power is used. For each server, the maximum frequency-dependent power is $P_d^{max} = 1$, sleep power is $P_s = 0.1$ and frequency-independent power is $P_{ind} = 0.3$. The two numbers in the legends stand for request size and request arrival rate (in terms of number of requests per second), respectively.



a. Restricted serial recovery

b. Parallel recovery

c. Adaptive parallel recovery

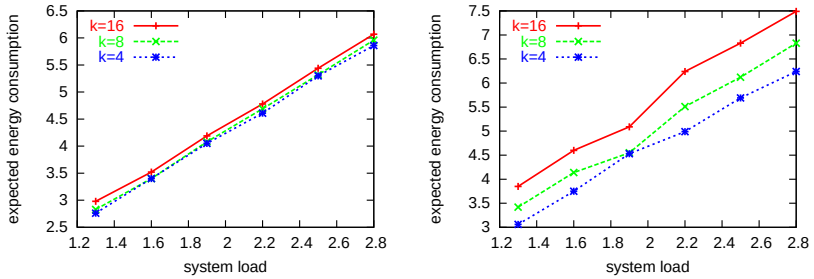
Figure 4: The expected energy consumption for different recovery schemes.

From the figure, we can see that, when the request size is $1ms$, the expected energy consumption is almost the same for different numbers of faults to be tolerated. The reason is that, to tolerate up to 15 faults, the

amount of slack time used by the backup slots is almost negligible and the amount of slack time used for energy management is more or less the same when each backup slot is only $1ms$. However, when the request size is $50ms$, the size of each backup slot is also $50ms$ and the minimum expected energy consumption increases significantly when the number of faults to be tolerated increases, because less slack is left for energy management.

Note that, to tolerate different numbers of faults, it may require the same number of backup slots, especially when higher levels of modular redundancy and parallel recovery schemes are deployed, which in turn leads to the same expected energy consumption as shown in the plateaus in Figure 4. Furthermore, to tolerate the same number of faults, the adaptive parallel recovery scheme is the most energy efficient for a give request size and corresponding request arrival rate (note the different scales of Y-axis in the figures).

For different system loads, Figure 5 further shows the expected energy consumption to tolerate given number of faults (e.g., 4, 8 or 16) under the adaptive parallel recovery scheme. For a given request size, different request arrival rates are used to obtain different system loads.



a. $10ms$ and 260 requests/second b. $50ms$ and 52 requests/second

Figure 5: The expected energy consumption under different system loads to tolerate given numbers of faults for the **adaptive** parallel recovery scheme ($k_e = \frac{k}{2}$).

From the figure, we can see that, for a given request size, as the system load increases, more requests need to be processed within the interval considered and the expected energy consumption to tolerate a given number of faults increases. As before, for a given number of faults, the difference in the expected energy consumption increases for larger requests due to larger backup slots and less slack for energy management. Moreover, the expected energy consumption for small request sizes ($1ms$ or $10ms$) is almost the same to tolerate 4, 8 or 16 faults within the interval considered.

To see the effects of different levels of pessimism, Table 3 shows the optimal redundant configuration that minimizes the expected energy consumption when tolerating a given number of faults. The request size used is $50ms$ and the request arrival rate is 52 requests per second. For adaptive parallel recovery, Duplex is always the best and is not shown in the table.

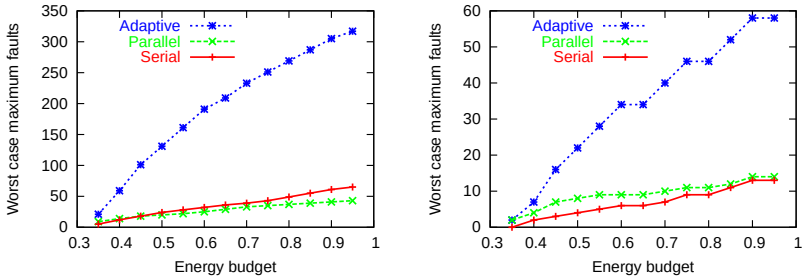
Table 3: The effects of pessimism levels on optimal redundant configuration.

k		2	4	6	8	10	12	14
recovery	k_e	N, p	N, p	N, p	N, p	N, p	N, p	N, p
restricted serial	0	2, 2	2, 3	2, 3	2, 5	2, 5	2, 5	2, 5
	$k/2$	2, 2	2, 3	2, 3	2, 3	**3, 3**	**3, 3**	**3, 3**
	k	2, 2	2, 3	2, 3	**3, 3**	**3, 3**	**3, 3**	**4, 3**
parallel	0	2, 2	2, 3	2, 3	2, 3	2, 5	2, 5	2, 5
	$k/2$	2, 2	2, 3	2, 3	2, 3	2, 5	2, 5	**3, 4**
	k	2, 2	2, 3	2, 3	2, 5	**3, 3**	**3, 3**	**4, 3**

From the table, we can see that the optimistic approach ($k_e = 0$) favors lower levels of modular redundancy and the pessimistic approach favors higher levels of modular redundancy. This comes from the fact that higher levels of modular redundancy needs less backup slots for tolerating a given number of faults, which results in more slack for scaling down all the processing and thus consumes less energy when the pessimistic approach is used. For the optimistic approach, only the processing during primary time slots counts in the expected energy consumption, which favors lower levels of modular redundancy that may have more backup slots.

6.2 Optimal Configuration for Reliability Maximization

Assume that the maximum power, P_{max}, corresponds to running all servers with the maximum processing frequency f_{max}. When the energy budget for each interval is limited, we can only consume a fraction of P_{max} when processing requests during any given interval. For different energy budgets (i.e., different fraction of P_{max}), Figure 6 shows the worst case maximum number of faults that can be tolerated when the optimal redundant configuration is employed. Again, different arrival rates are considered for different request sizes to get a fixed system load of 2.6 and Y-axes have different scales.



a. $10ms$ and 260 requests/second b. $50ms$ and 52 requests/second

Figure 6: The worst case maximum number of faults that can be tolerated for different request sizes when the energy budget is limited.

From the figure, we can see that the number of faults that can be tolerated increases with increased energy budget. When the request size increases, there are less available backup slots due to the large slot size and fewer faults can be tolerated. When the number of backup slots is very large (e.g., for the case of $10ms$ with 260 requests/second), the same as shown in Section 3.5, the parallel recovery performs worse than the restricted serial recovery (due to fixed allocation of all recovery sections). The adaptive parallel recovery performs the best and can tolerate many more faults than the other two recovery schemes at the expense of more complex management of backup slots.

As observed in the last section, for optimal redundant configurations, parallel recovery, small requests and optimistic approaches favor lower levels of modular redundancy, while restricted serial recovery, large requests and pessimistic approaches favor higher levels of modular redundancy.

7 Conclusions

In this work, we consider an event-driven application that is served by a distributed system that consists of a fixed number of servers. To efficiently use slack time as temporal redundancy for providing reliable service, we propose an adaptive parallel recovery scheme that appropriately recovers requests from faults in parallel. Furthermore, we show that this scheme leads to higher reliability than serial or non-adaptive parallel recovery schemes.

Combining modular redundancy with parallel recovery, we propose schemes to determine the optimal redundant configuration to minimize energy consumption for a given reliability goal or to maximize system reliability for a given energy budget. In this context, a redundant configuration is specified by the level of modular redundancy employed, the number of servers used (the remaining servers are turned off for energy efficiency), the number of backup slots needed and the processing frequency of the working servers.

Our analysis shows that the restricted serial recovery scheme, which does not consider parallelism of available slack time, favors higher levels of modular redundancy. The adaptive parallel recovery favors lower levels of modular redundancy since there are more available NMR groups. In most cases, Duplex is the best. In general, larger requests and pessimistic approaches favor higher levels of modular redundancy while smaller requests and optimistic approaches favor lower levels of modular redundancy.

References

[1] T. Austin, D. Blaauw, T. Mudge, and K. Flautner. Making typical silicon matter with razor. *IEEE Computer*, Mar. 2004.

[2] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. *The case for power management in web servers*, chapter 1. Power Aware Computing. Plenum/Kluwer Publishers, 2002.

[3] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of The HICSS Conference*, Jan. 1995.

[4] X. Castillo, S. McConnel, and D. Siewiorek. Derivation and calibration of a transient error reliability model. *IEEE Trans. on computers*, 31(7):658–671, 1982.

[5] E. (Mootaz) Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. of Power Aware Computing Systems*, 2002.

[6] E. (Mootaz) Elnozahy, R. Melhem, and D. Mossé. Energy-efficient duplex and tmr real-time systems. In *Proc. of The IEEE Real-Time Systems Symposium*, 2002.

[7] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage. In *Proc. of the Workshop on Power-Aware Computing Systems*, 2003.

[8] S. Hua and G. Qu. Power minimization techniques on distributed real-time systems by global and local slack management. In *Proc. of the 2005 conference on Asia South Pacific design automation*, pages 830–835, New York, NY, USA, Jan. 2005. ACM Press.

[9] Intel. Intel xscale processors. http://developer.intel.com/design/intelxscale/, 2006.

[10] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proc. of The* 14^{th} *Symposium on Discrete Algorithms*, 2003.

[11] T. Ishihara and H. Yauura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of The 1998 International Symposium on Low Power Electronics and Design*, Aug. 1998.

[12] R.K. Iyer, D. J. Rossetti, and M.C. Hsueh. Measurement and modeling of computer reliability as affected by system activity. *ACM Trans. on Computer Systems*, 4(3):214–237, Aug. 1986.

[13] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proc. of the* 41^{st} *annual Design automation conference (DAC)*, 2004.

[14] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Proc. of the* 9^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[15] C. Lefurgy, K. Rajamani, Freeman Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *IEEE Computer*, 36(12):39–48, 2003.

[16] J. Luo and N. K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *Proc. of International Conference on Computer Aided Design*, Nov. 2000.

[17] J. Luo and N. K. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *Proc. of* 15_{th} *International Conference on VLSI Design*, Jan. 2002.

[18] R. N. Mahapatra and W. Zhao. An energy-efficient slack distribution technique for multimode distributed real-time embedded systems. *IEEE Trans. on Parallel and Distributed Systems*, 16(7):650–662, Mar. 2005.

[19] R. Melhem, D. Mossé, and E. (Mootaz) Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Trans. on Computers*, 53(2):217–231, 2004.

[20] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem. Energy aware scheduling for distributed real-time systems. In *Proc. of International Parallel and Distributed Processing Symposium (IPDPS)*, pages 21–29, Piscataway, NJ, USA, Apr. 2003. IEEE CS Press.

[21] D. K. Pradhan. *Fault Tolerance Computing: Theory and Techniques*. Prentice Hall, 1986.

[22] Rambus. Rdram. http://www.rambus.com/, 1999.

[23] C. Rusu, A. Ferreira, C. Scordino, A. Watson, R. Melhem, and D. Mossé. Energy-efficient real-time heterogenerous sever clusters. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2006.

[24] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. In *Proc. of the IEEE Real-Time System Symposium*, 2003.

[25] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware qos management in web servers. In *Proc. of the* 24^{th} *IEEE Real-Time System Symposium*, Dec. 2003.

[26] K. G. Shin and H. Kim. A time redundancy approach to tmr failures using fault-state likelihoods. *IEEE Trans. on Computers*, 43(10):1151 – 1162, 1994.

[27] A. Sinha and A. P. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Proc. of Design Automation Conference*, Jun 2001.

[28] S. Thompson, P. Packan, and M. Bohr. Mos scaling: Transistor challenges for the 21st century. *Intel Technology Journal*, Q3, 1998.

[29] Transmeta. Transmeta processors. http://www.transmeta.com, 2004.

[30] O. S. Unsal, I. Koren, and C. M. Krishna. Towards energy-aware software-based fault tolerance in real-time systems. In *Proc. of The International Symposium on Low Power Electronics Design (ISLPED)*, Aug. 2002.

[31] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proc. of The First USENIX Symposium on Operating Systems Design and Implementation*, Nov. 1994.

[32] R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mossé. Energy efficient policies for embedded clusters. In *Proc. of the Conference on Language, Compilers, and Tools for Embedded Systems (LCTES)*, pages 1–10, New York, NY, USA, Jun. 2005. ACM Press.

[33] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. of The* 36^{th} *Annual Symposium on Foundations of Computer Science*, 1995.

[34] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proc. of IEEE/ACM Design, Automation and Test in Europe Conference(DATE)*, 2003.

[35] Y. Zhang and K. Chakrabarty. Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems. In *Proc. of IEEE/ACM Design, Automation and Test in Europe Conference(DATE)*, 2004.

[36] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[37] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proc. of the International Conference on Computer Aidded Design (ICCAD)*, Nov. 2006.

[38] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the Int'l Conference on Computer Aidded Design*, 2004.

[39] D. Zhu, R. Melhem, D. Mossé, and E.(Mootaz) Elnozahy. Analysis of an energy efficient optimistic tmr scheme. In *Proc. of the* 10^{th} *Int'l Conference on Parallel and Distributed Systems*, 2004.