Efficient Implementation Techniques for Gracefully Degradable Multiprocessor Systems

Jyh-Charn Liu, *Member, IEEE Computer Society*, and Kang G. Shin, *Fellow, IEEE*

***Abstract*—We propose the *dynamic reconfiguration network* (DRN) and a *monitoring-at-transmission* (MAT) bus to support dynamic reconfiguration of an N-modular redundancy (NMR) multiprocessor system. In the reconfiguration process, a maximal number of processor triads are guaranteed to be formed on each processor cluster, thus supporting gracefully degradable operations. This is made possible by dynamically routing the control and clock signals of processors on the DRN so as to synchronize fault-free processors. The MAT bus is an efficient way to implement a triple modular redundant (TMR) *pipeline voter* (PV), which is a special case of the *voting network* proposed in [1]. Extensive experimental results have shown to support our design concept, and the performance of different cache memory organizations is evaluated through an analytic model.**

***Index Items*—Clock synchronization, cluster-based multiprocessors, N-modular redundancy (NMR), pipelined voter (PV), reliability, voting.**

I. Introduction

With the advance of very large scale integration (VLSI) technology, large multiprocessor systems are becoming available on commercial market at reasonable prices. These multiprocessor systems are ideal candidates for providing high-performance, scalable computing services for a broad range of time- and safety- critical applications. Typical examples of these applications include radar signal processing, command and control centers, on-line optimization of telecommunication and urban transportation networks [2], [3]. In these applications, computational tasks should be completed before their deadline to avoid catastrophe or any significant degradation of quality of system service. Faults that might occur to the computing system should therefore be quickly detected and isolated in such a way that normal computation can be resumed in a bounded time.

Most failure recovery strategies for distributed systems are static in nature, and thus are not very efficient for multiprocessor systems [4], [5]. Moreover, commercial fault-tolerant multiprocessor systems are usually designed for transaction-oriented applications (e.g., [6], [7], [8], [9]) which do not consider the deadline constraints and thus are not well suited for time-critical applications. For its simplicity and high fault coverage, the hardware-based majority voting architecture is widely accepted for the design of real-time, non-stop fault-tolerant systems [10], [11], [12], [13]. In this architecture, a functional unit is replicated N times, forming an *N-modular redundant* (NMR) unit, and copies of each critical program are executed in lock-step by redundant units. The NMR architecture is very flexible, and it can be integrated with different fault detection/correction mechanisms. For example, in the Hitachi FT-6100 system [13] a single-error-correction/double-error-detection (SEC/DED) code is used in its main memory, but its central processor unit is protected with an NMR architecture.

Several NMR-based fault-tolerant multiprocessor systems have been built. Some of the most notable systems include the FTMP [10], Fault-Tolerant Processor (FTP) [14], and C.vmp [15]. Flexibility (in providing fault-tolerance) and performance are two main issues associated with the implementation of an NMR-based multiprocessor system. The NMR architecture is conceptually simple, but its implementation may have a major impact on system performance. In an NMR system, clocks of the N redundant units need to be synchronized with one another so as to ensure lock-step execution of copies of a program on the redundant units. For its cost-effectiveness, the *All Digital Phase-Locked Loop* (ADPLL) [16] technique is widely used, which can synchronize the N modules at a clock rate lower than the running clocks of processors, e.g., the FTP and the Transputer [17] architectures. For clocks of the redundant modules to be phase-locked to each other by an ADPLL, the phase difference of the N clocks are compared at a fixed time interval of every k pulses of the running clocks, called one *synchronization cycle*. Using the phase-comparison results of the synchronization cycles, the running clocks are adjusted, by addition/deletion of clock pulses, to compensate for the phase difference of the synchronization cycles. This way the phase difference between N clocks sources can be upper bounded by a physical time margin S_u. Unlike the conventional analog PLL technology, the ADPLL can be implemented with digital circuitry, and it has good tolerance to the time skew between redundant units.

Despite its cost-effectiveness, a major problem associated with the ADPLL-based NMR architecture is that the peak voting frequency of redundant units in a conventional voter is constrained by S_u. To overcome this problem, Parhami proposed a pipelined, multiple stage cellular voter architecture, called *voting networks*, to support different majority voting rules [1]. We will call a triple modular redundancy (TMR) voting network a *pipelined voter* (PV). A PV consists of a majority voter at its output and a set of input buffers, each of which is associated with a status bit to indicate the readiness of

Manuscript received October 13, 1992; revised June 1993.
J.-C. Liu is with the Dept. of Computer Science, Texas A&M University, College Station, TX 77843-3112; e-mail liu@cs.tamu.edu.
K.G. Shin is with Real-Time Computing Laboratory, Dept. of Electrical and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122.
IEEECS Log Number C95019.

data. For their flexibility, ADPLL and PV will be used in our study as a basis for the implementation of an NMR-based multiprocessor system.

Two main issues, state recovery and reconfiguration, need to be considered for the design of a gracefully degradable multiprocessor system. A large multiprocessor system may consist of hundreds of processors. Therefore, time efficiency is a main concern for fault recovery of large-scale multiprocessor systems. The time required to restore the processor state is relatively small. However, if the size of main memory is very large (as is usually the case), then the time spent on majority voting for memory-state recovery can be substantial. To overcome this problem, a memory paging technique was proposed [18], so that only the memory pages being modified need to be voted on for state realignment. An important question left unanswered in [18] is "What is the optimal page size for state realignment?" To address this question, we develop an efficient algorithm to derive the optimal memory page size.

To deal with the system reconfiguration issue, we use dynamic reconfiguration which can maintain a maximal number of fault-free redundant modules in the system. To fully utilize the architectural features of contemporary multiprocessor systems, we propose two important supporting mechanisms, both of which are considered as the system reliability hard core, for the implementation of an NMR-based multiprocessor system. The first mechanism is the *monitoring-at-transmission* (MAT) bus for cost-effective implementation of PV. The other is the *dynamic reconfiguration network* (DRN) for dynamic reconfiguration of clocking and control signals of redundant units. The MAT bus can be implemented by the wired–OR/AND logic or other similar techniques, and the MAT feature can be readily found in some of existing commercial products. On a MAT bus, each of NMR processor modules monitors the bus transaction when they output data to the bus. If an inconsistency is detected between the bus value and the output value in a functional unit, then a fault is detected, and the computation is interrupted to recover from the fault. The MAT bus is well suited for the widely-used cluster-based multiprocessor architecture. A processor cluster consists of a set of processors interconnected via a broadcast bus, and the processor clusters are interconnected via a different system network. A DRN can be implemented with ADPLLs and multiplexor-demultiplexor circuits. When some processor modules fail, the failed modules can be decoupled from the DRN, and the remaining fault-free units can be regrouped together into new NMR units. For simplicity, we will use the TMR model as an illustrative example throughout the paper.

The rest of the paper is organized as follows. The dynamically-reconfigurable architecture and the algorithm for optimizing the memory-page size are described in Section II. Experimental results and performance evaluations of the proposed architecture are presented in Section III. Concluding remarks are made in Section IV.

II. System Architecture

A. System Organization

Processor modules—each of which consists of a processor and its cache memory—and memory modules are the two basic functional units to be considered in our design, as in most commercial systems [19]. In a processor triad, three processor modules are synchronized with one another, and all the external data writes are voted on by the PV to mask faults in the processor modules. Like the design of the C.vmp architecture [15], the PV is placed between the cache and main memory to form a processor triad (see Fig. 1) so that both the write-back and write-through cache coherence protocols can be incorporated. Coordination between the PV and cache memory is described by the flowchart in Fig. 2.

The main difference between the MAT-bus voter and a PV is that, on the MAT-bus voter, an inconsistency between redundant processors is detected, not masked, in each data vote. For processor modules in a processor triad to take a vote on their outputs, they first place data into their buffers and then make a transmission request to the bus arbiter. The bus arbiter can grant the bus transmission after all the ready bits in the same triad are asserted, and all processor modules will then monitor the bus transmission. If any processor detects an inconsistency between data on the bus and its own output value, the processor invalidates the transaction, and the data voting will be retried. If the inconsistency remains even after several retries, a permanent fault is assumed detected and the processor reconfiguration procedure will be invoked.

In the regrouping process of fault-free processors on the MAT bus, fault-free processors may need to be first decoupled from their own processor triads, so that they can later be coupled with other fault-free processors to form new processor triads. After fault-free processors are dynamically grouped into triads, their control and clock signals need to be phase-locked into each other. The dynamic reconfiguration network (DRN) is designed to serve this purpose. A DRN consists of a set of *DRN modules* (DRNM) connected to one another forming a logical ring. Each DRNM has five access ports, three of which are directly connected to three processor modules, and the other two are connected to other DRNMs on the ring. There is a three-input ADPLL circuit in each DRNM for synchronization of the clocking signals from any three of the five access ports. When a processor cluster is free of any faults, processor modules directly connected to a DRNM are synchronized with one another on the DRNM to form a processor triad. However, once a fault is detected in a processor triad, the failed processor module will be disconnected from the DRNM associated with the processor triad. Then, the DRNM will coordinate with other DRNMs to form new processor triads after the clock and control signals of the non-faulty processor modules are properly grouped with each other by the reconfiguration algorithm to be discussed shortly.

Fig. 3 depicts an example DRN architecture, in which $DRNM_1$, $\cdots$, $DRNM_n$, are organized into a ring. $DRNM_i$ consists of an $ADPLL_i$ circuit and five access ports, $R_1^i, \ldots, R_5^i$,

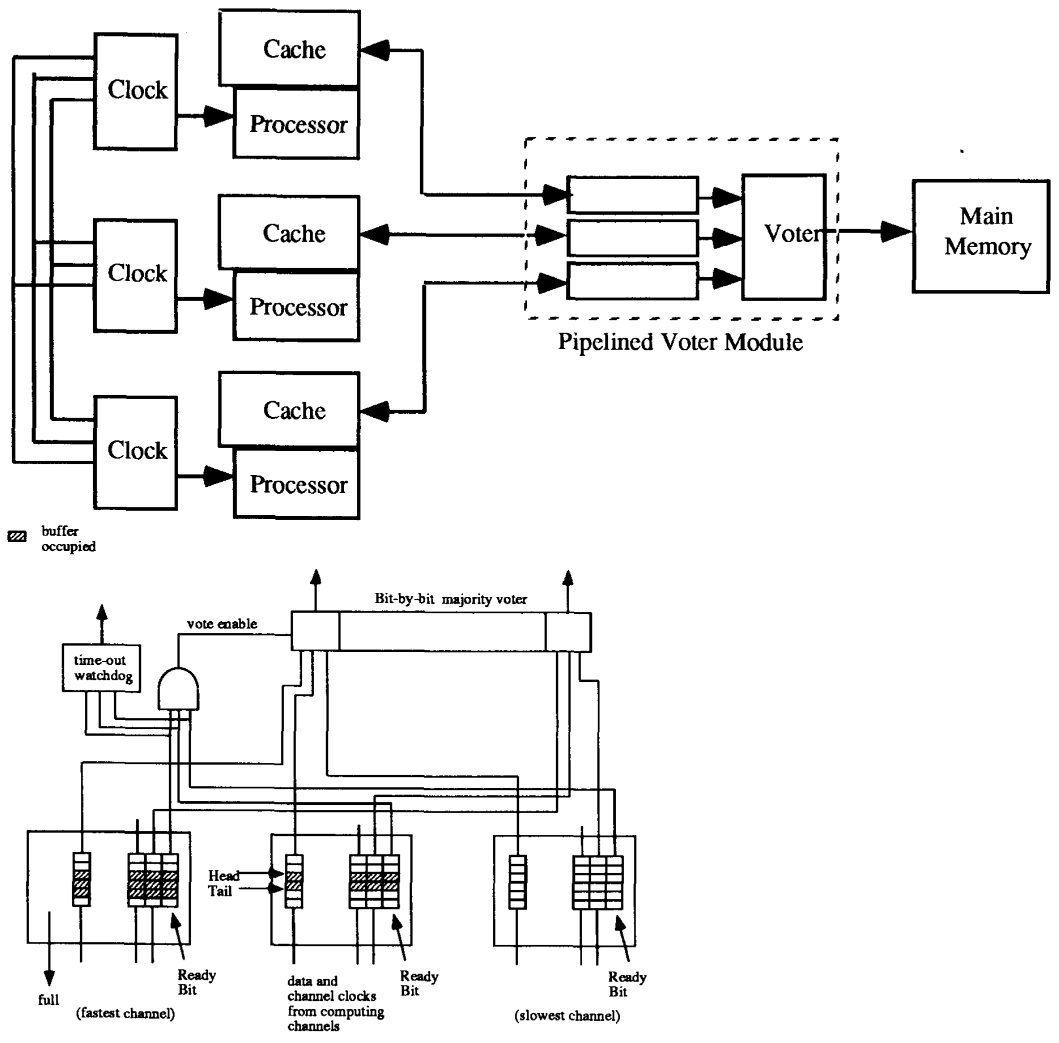


Fig. 1. The basic organization of a PV.

$i = 1, 2, ..., n$. To form a ring between DRNMs, R_5^i is connected to R_1^{i+1}, $i = 1, 2, \cdots n$, and R_1^n is connected to R_5^1. The clock and control signals of three processor (memory) modules are directly connected to three of the five access ports: R_2^i, R_3^i, R_4^i. Clocks from any three of the five access ports of $DRNM_i$ can be synchronized to each other on $ADPLL_i$. Therefore, when necessary, any of the clocking signals from $\{R_1^i, R_2^i, R_3^i, R_4^i\}$ can be routed to $ADPLL_{i+1}$ along R_5^i, so that on $ADPLL_{i+1}$ the clock can be synchronized with any two other clocking signals from $\{R_2^{i+1}, R_3^{i+1}, R_4^{i+1}, R_5^{i+1}\}$. Similarly, the clocking signal from any of access ports $\{R_2^i, R_3^i, R_4^i, R_5^i\}$ can be routed to R_1^i, so that it can be synchronized with any two other clocking signals from $\{R_2^{i-1}, R_3^{i-1}, R_4^{i-1}, R_5^{i-1}\}$ on $ADPLL_{i-1}$. The direct path between R_1^i and R_5^i is a bypass between $DRNM_{i-1}$ and $DRNM_{i+1}$ for them to directly exchange/synchronize clock signals.

B. System Reconfiguration

The DRN architecture is managed by the *neighborhood-grouping* reconfiguration algorithm, which is guaranteed to find a maximal number of processor triads on a processor cluster. Let $f_1, f_2, f_3, \cdots f_n, f_i \in \{0, 1, 2, 3\}$, denote the numbers of fault-free processor modules on the n DRNMs of a cluster, then we can get $\left\lfloor \frac{\sum_{i=1}^{n} f_i}{3} \right\rfloor$ processor triads using this algorithm.

For convenience, a DRNM with i fault-free processor modules connected to its access ports is called an i-DRNM. Essentially, the neighborhood-grouping algorithm is based on a first-fit principle for processors on 2-DRNMs and 1-DRNMs to be grouped with each other along a ring direction. An *error* signal connected to every DRNM through a wired-OR broadcast line is used to inform every DRNM about initiation and termination of a reconfiguration process. A reconfiguration

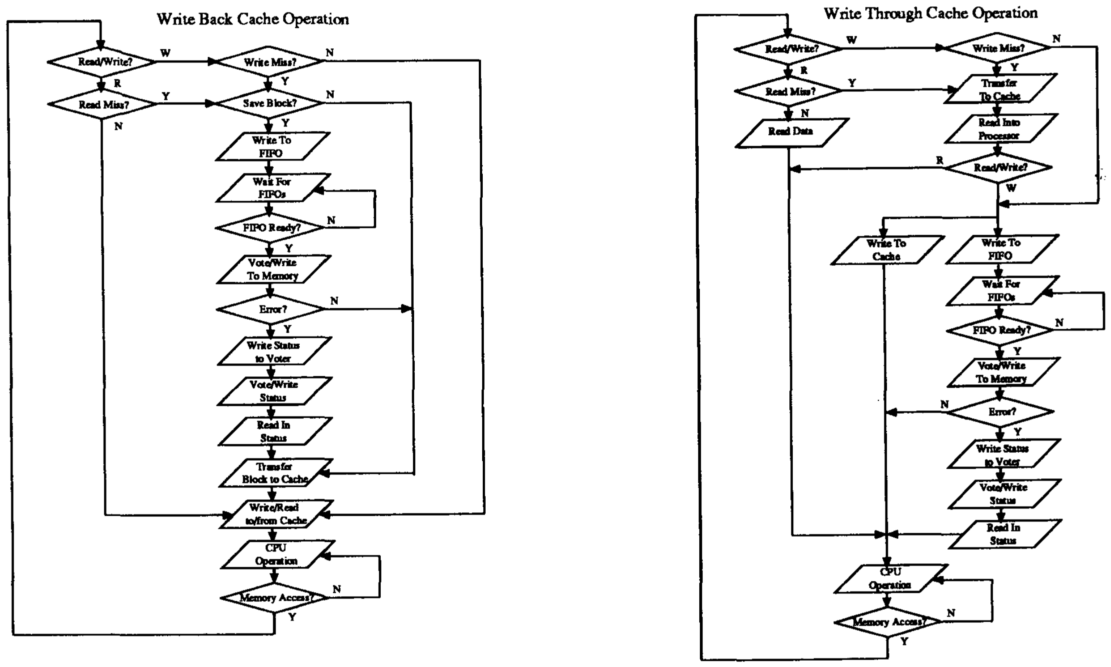


Fig. 2. The cache coherence protocols with PV.

process is initiated by the DRNM which detects the latest failed processor module by asserting the error signal. The reconfiguration process is terminated when the error signal is reset. Once the error signal is asserted by a DRNM, all the 2-DRNMs and 1-DRNMs connected on the ring will participate in the reconfiguration process. Since a DRNM can accept at most two external clock sources, no triad can be formed on 0-DRNMs, and thus 0-DRNMs do not participate the reconfiguration process. Moreover, including 3-DRNMs in the reconfiguration process will add extra overhead without gaining any flexibility, because if any processor in a 3-DRNM is to be grouped with other fault-free processors, then we will need a new fault-free processor from another processor triad to make up the loss.

For convenience, the DRNMs participating in the reconfiguration process are relabeled along the ring direction as D_1, D_2, $D_3 \cdots D_h$, $h \geq 1$, where D_1 is the coordinator, R_1^i of D_i is connected to R_5^{i+1} of D_{i+1}, and R_1^h of D_h is connected to R_5^1 of D_1. Any DRNM between D_i and D_{i+1} is in the bypass mode. Three different types of messages, *invite, join,* and *done*, are passed between DRNMs along the directed ring direction, assuming that D_i sends its messages to D_{i+1} through R_1^i, and D_i receives messages from R_{i-1} through R_5^i. The invite-message is used by a message sender to indicate that it needs one more processor module to form a new triad. The join-message is used by a message sender to indicate that it has only one processor module which can be combined with other processor modules to form a new triad. Finally, the *done-message* is used to indicate that all the processor modules between the initiator and the message sender have been combined into triads. D_i, $i \geq 1$, responds to the three different types of messages based on the following rules.

- **invite-message**: D_i grants the request and routes the clock-control signal of a non-faulty processor module to D_{i-1} through R_5^i. If D_i does not have any more non-faulty processor module, then it passes a done-message to D_{i+1}.
- **join-message**: 1) If D_i is a 1-DRNM then it sends an invite-message to D_{i+1}, and waits for the response from D_{i+1}. If the request is granted, a triad can be formed on D_{i+1}. Otherwise, if the error-signal is reset then the reconfiguration process is aborted without forming any triad. 2) If D_i is a 2-DRNM, then it accepts the join-request from D_{i-1} to form a new triad, and it will send a done-message to D_{i+1}.
- **done-message**: If D_i is a 2-DRNM, then it sends an invite-message to D_{i+1}. Otherwise, it sends a join-message to D_{i+1}.

When the reconfiguration process is initiated, D_1 sends an invite-message to D_2 if it is a 2-DRNM; otherwise it sends a join-message to D_2 if it is a 1-DRNM. A new triad can be formed on D_1 if the invite-message is granted, and the clock-control signals (of a non-faulty processor module) from R_1^1 of D_1 will be synchronized with that of the two non-faulty processor modules on D_1. Similarly, if the join-message of D_1 is granted, then the only non-faulty processor on D_1 can be grouped with other processor modules to form a new triad on some other DRNM. After D_i, $i \geq 1$, sends out a request, it waits for a response from D_{i+1}. New triads can be formed either on

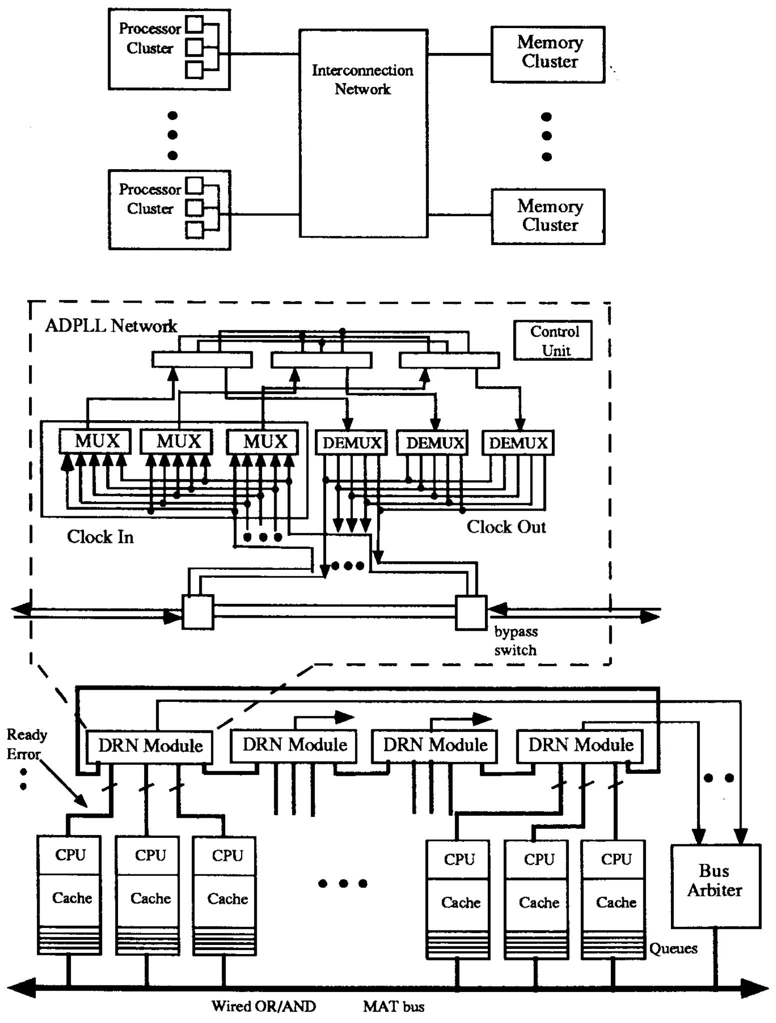


Fig. 3. The DRN in a clustered multiprocessor system.

D_i or on some other node if the request is granted. However, the reconfiguration process will be terminated with no new triad formed if the error-signal is reset by D_1 before the grant message is received by D_i. When D_1 receives an invite/join-request from R_5^1, it implies exhaustion of non-faulty processor modules, and D_1 needs to terminate the process, by resetting the error signal, with no new triad formed.

The proposed algorithm is given in the C-language like pseudo code as follows.

```
Neighborhood-Grouping Algorithm ($DRNM_i$)/* that runs on $DRNM_i$ */
IF ($DRNM_i$ is a 3-DRNM or 0-DRNM) then activate its bypass circuit
    and wait until one of its processor modules fails;
IF (one processor module of $DRNM_i$ fails) set reconfiguration:= True;
IF (reconfiguration is True)
{IF ($DRNM_i$ is the initiator)/* $f_i$ the number of non-faulty processor
modules on $DRNM_i$*/
        {IF ($f_i = 2$) {send the invite-request along $R_1^i$;
          wait until (the response from $R_1^i$ OR $R_5^i$)
          IF (ACK from $R_1^i$) {accept the clocks from $R_1^i$ to form a new
           triad;
              wait for any message from $R_5^i$;
              reset the error-signal;
              return(1);}/* a triad is formed on $DRNM_i$ */
          ELSE IF (invite/join request from $R_5^i$)/* No triad can be
           formed */
              {reset the error-signal; return(0);}; /* no triad can be
                formed */
      IF ($f_i = 1$) send the join-request along $R_1^i$;
          wait until (the response from $R_1^i$ OR $R_5^i$)
          IF (ACK from $R_1^i$) {route the clock of its only fault-free
           processor to $R_1^i$;
          wait until (a response from $R_5^i$); reset error-signal; return(0);}
      IF ($f_i = 0$) {send done-message along $R_1^i$; wait for message from
          $R_5^i$; reset error-signal;} return(0);
      }
```

```
IF ($DRNM_i$ is not the initiator) wait until a request received from $R_5^i$;
{       IF (an invite-request received) {grant the request; route the clock
        of one of the non-faulty processor modules on $DRNM_i$ to $R_5^i$;
            IF ($DRNM_i$ has no more fault-free processor module)
            Send a "done-message" to $R_1^i$;
            ELSE IF ($DRNM_i$ has one fault-free processor module)
            Send a join-request to $R_1^i$;
            wait until the request granted or the error-signal is reset;
              return(0);
            }
        IF (a join-message received)
        {IF ($f_i = 2$)
            {grant the request;
            accept the clock-control signals from $R_5^i$ to form a triad;
            send the "done-message" to $R_1^i$, and
            wait until the error signal is reset; return(1);
            }
            IF ($f_i = 1$)
            {send the invite-request to $R_1^i$;
                    IF (ACK is received) accept the clocks from $R_1^i$ and $R_5^i$
                      to form a triad;
                    wait until the error-signal is reset; return(1);
                    ELSE IF (REJECT is received OR error-signal is
                      reset) no triad is found,
                    reconfiguration is terminated;
                    return(0);
            }
        }
        IF (a done-message received)
        {IF ($f_i = 2$)
            {send the invite-request to $R_1^i$;
                    IF (ACK is received) accept the clocks from
                    $R_1^i$ to form a triad;
            wait until the error-signal is reset; return(1);
            }
            IF ($f_i = 1$)
            {send the join-request to $R_1^i$;
                    IF (ACK is received) {route the clock of the processor
                    module
                    on $DRNM_i$ to $R_1^i$;}
                    wait until the error-signal is reset; return(0);
              }
        }
}
```

We now prove that the neighborhood grouping algorithm is deadlock-free based on the following simple argument. As mentioned earlier, both the MAT bus and the DRN are the reliability hard cores, implying that DRNMs will not fail during the reconfiguration process. In the neighborhood grouping algorithm, each of the DRNMs participating in the reconfiguration process receives from, and generates a message to, its neighbors along the same ring direction. Thus, the reconfiguration initiator will receive a message and will terminate the reconfiguration process in at most n steps, and all processors will proceed with their subsequent computational steps.

The process for reconfiguring processor modules is illustrated with an example plotted in Fig. 4. In this example, 15

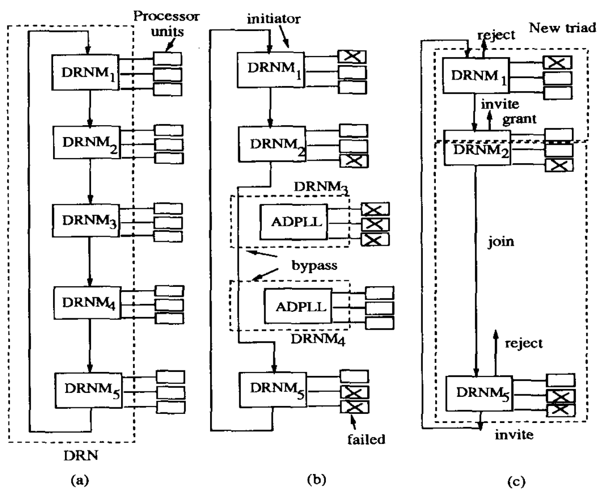


Fig. 4. Processor triad configurations under different conditions: (a) no processor module fails, (b) after a new processor on *DRN* M_1 fails, and (c) after the reconfiguration.

processor modules in a cluster form five processor triads. After several processor modules had failed, we have taken a snapshot of the system state immediately after the latest failure occurred to a processor module on $DRNM_1$, i.e., Fig. 4(b). After detecting the faulty processor, $DRNM_1$ orders other processors to begin the reconfiguration process. After their state information is properly saved, all the existing triads on the 1- and 2-DRNM are decoupled from each other, and the neighborhood-grouping algorithm is initiated. $DRNM_3$ and $DRNM_4$ will not be involved in the reconfiguration process since they have three and zero faulty processor modules, respectively. After an invite-message is sent from $DRNM_1$ to $DRNM_2$, the request is granted by $DRNM_2$, since it has two non-faulty processor modules. The new processor triad is formed on $DRNM_1$, and $DRNM_2$ will then send a join-message to $DRNM_5$ since it has only one non-faulty processor module available. $DRNM_5$ does not have enough processor modules to form a new triad. Thus, it sends an invite-message to its next neighbor, $DRNM_1$. Since a processor triad is already formed on $DRNM_1$, $DRNM_1$ terminates the reconfiguration process by resetting the error-signal. As a result, no new processor triad can be formed on $DRNM_5$. The final configuration is plotted in Fig. 4(c).

When an NMR system is used for hard real-time applications, it is important to know the worst-case timing behavior for the scheduling of fault recovery routines. For this purpose, we briefly analyze the time complexity of processor-cluster reconfiguration process as follows. Let N_f denote the total number of 2– and 1–DRNMs on a processor cluster. After a faulty processor module is detected in a processor triad, the state of processor triads on the 2– and 1–DRNMs needs to be saved into the main memory before the reconfiguration process can begin, and this will take $N_f T_{save\text{-}state}$ time units to complete. It will then take $N_f T_{neighbor}$ time units for the *DRNM*s to be grouped with each other based on the nearest-neighbor algorithm, where $T_{neighbor}$ is the worst-case time for one 2– or 1–DRNM to make its reconfiguration decision. After the reconfiguration decision is made, the clocks of the processors in the same processor triad need to be phase-locked to each other on the ADPLL of a 2– (1–)DRNM before normal computation can be resumed on the processors. The different clock sources can be phase-locked by resetting the clock sources of the processors and waiting until the phase difference between the different sources falls into a pre-specified time skew. We denote this phase-locking time as $T_{phase-lock}$, and it is upper-bounded by $N_f\, T_{phase-lock}$. Finally, the state of processor triads needs to be reloaded, and the time complexity of this operation is $N_f\, T_{state-restore}$. Summarizing the above discussion, we can express the total time complexity of processor reconfiguration as $N_f\,(T_{save-state} + T_{neighbor} + T_{phase-lock} + T_{state-restore})$.

C. System State Alignment

The DRN is designed to dynamically phase-lock the clocks and control signals of redundant processor modules during reconfiguration. However, phase-locking clocks is only necessary, but not sufficient, to ensure consistency of processor states. Note that even under normal operation, redundant processors in a triad may have inconsistent states if they do not handle external events at the same time, since the physical time skew between the redundant processors may not be negligible. Hence, the redundant processors need to handle external events at identical logical steps to ensure their mutual consistency. For instance, when an I/O interrupt signal is asserted, processors need to wait for each other in order to enter an identical state for processing the interrupt event. This can be achieved by two design approaches: 1) a precise interrupt architecture for processors so that they will flush their pipelines before handling the interrupt, and 2) the interrupt requests are processed by each processor at the beginning (end) of each synchronization cycle. Similarly, when processors need to read external data, they have to write the input data into main memory that will be voted on by PVs. A watchdog timer is also needed to detect stalled redundant processors due to failed ready bits of the PV.

Realignment of processor state is relatively easy with a small performance penalty. When a fault is detected (masked) by the PV, the system can either ignore the failed processor or generate an interrupt signal to the processors for realignment of their internal states. That is, the cache memory needs to be flushed, and the registers' contents would be written back to the main memory. Since all the data must pass through the PV during the flushing of cache and registers, faults in any one of the redundant units will be masked. After the masked data is written into the main memory, processors can read back the registers and resume their computation. This way all the transient faults in processors and cache memories can be masked quickly. However, if faults continue to be detected in a module, the faulty module should be retired. Before the reconfiguration process begins, the old state information needs to be saved first, and then, after completing the reconfiguration, the newly-grouped functional units need to read in their state information altogether to resume lock-step execution of the program.

In a system with large main memory, realigning the memory state based on majority voting may become very time-consuming. The inefficiency of majority voting can be alleviated by SEC/DED codes within each memory module, and each memory module can have a backup copy to cope with permanent failures. After a permanent fault occurred, and memory modules are regrouped, the state of the new backup module can be made identical to that of the original module in a word-by-word read/write manner. It should be noted, however, that transient faults are the predominant causes of memory failures [20], [21]. The memory can be periodically scanned to recover from the single-bit transient faults with the SEC/DED codes. However, SEC/DED codes are not perfect; for example, they usually do not handle faults in the control/address and other decoding circuitry. Therefore, the voting technique is still useful for realignment of memory state in recovering from transient faults.

To reduce the memory realignment time, a memory system can be partitioned into pages, each of which is associated with an update tag bit to indicate its status [18]. Let the main memory of size W be partitioned into K pages. Only those pages that have been updated need to be realigned. That is, the memory realignment time can be expressed as

$$t_{ra} = \left(K + F\frac{W}{K}\right)t_v,$$

where t_v is the time to take a vote, and F a random variable denoting the number of recovery pages to be realigned, $0 \le F \le K$. We assume that W is an integral multiple of K, and the inaccuracy resulting from such an approximation is found to be negligible.

Although the memory partitioning technique has been successfully implemented in [18], there remains an important question unanswered: "What is the optimal page size to balance between the page realignment time and the tag scanning time of the memory pages?" Since only the faulty pages need to be realigned, if the page size is too small, the time overhead of page scanning becomes the dominating performance overhead as compared to the actual page realignment time. On the other hand, if the page size is too large, then the page-realignment time may become excessive. It is impossible to deterministically guarantee an upper bound of the realignment time. So, we propose to guarantee that *for the mission period t, the probability of memory realignment requiring longer than a time period T is less than a given ε*. That is, depending on the memory failure rate, one can make a tradeoff between the time for scanning the page tags and the time to realign the faulty pages by minimizing the realignment cost. This is formally stated as:

$$\begin{array}{ll} \textbf{min} & Z(t) = K + F(t)\dfrac{W}{K} \\ \textbf{subject to} & P_K\big(Z(t) > T\big) < \varepsilon \\ & K \in \mathbf{I}^+,\ K < W, \end{array}$$

where $F(t)$ is a random variable denoting the number of faulty pages at time t, and K is the number of memory pages. This nonlinear integer optimization problem could be solved by a conventional optimization algorithm, but the computational time would be prohibitively high. To reduce the computational time without loss of its solution accuracy, a two-step approach is proposed here. At the first step, the objective function is approximated as a continuous function, and its optimization solution is solved likewise. Then, at the second step, the exact optimal value is derived using an exhaustive search technique in a bounded vicinity of the approximate optimal page size. We will show that the exact optimal page size can be obtained quite efficiently by our scheme, as an example to be shown shortly. We now explain the key idea on the approximation technique.

Theorem 2.1 : *When $K^* \gg 1$, we have $K^* \approx \sqrt{f_K W}$, where K is an arbitrary integer, $1 \ll K < W$.*

In this Theorem, f_K denotes the maximal number of faulty pages that can be realigned without violating the constraint of the optimization problem, and K is a randomly-chosen page size, assuming that the SEC/DED codes have perfect fault detection capability. Proof of this theorem is fairly involved, and thus is given in Appendix A. Using this theorem, we can get the optimal page size in three steps. At the first step, we choose a random page size K, and then get f_K using Lemma A.2 of Appendix A. Then, at the second step, we can get the approximate optimal solution as $\sqrt{f_K W}$ by Theorem 2.1. Finally, the exact optimal page size can be obtained through an exhaustive search routine in a bounded vicinity of the approximate solution.

An example cost function $Z(t)$ is plotted in Fig. 5. The curve shown in Fig. 5 is $K + f_K\lfloor \frac{W}{K} \rfloor$. It can be seen that the integral constraint on both K and $\lfloor \frac{W}{K} \rfloor$ causes a sawtooth curve in $[K - \Delta K, K + \Delta K]$, $\forall K$, but has only a small impact on the global curve shape. In this example, $\varepsilon = 10^{-5}$, and thus $f_K = 10$. Thus, $K' = \sqrt{10 \times 4 \times 10^6} = 6324.5$. Through an exhaustive search, it is found that there are multiple optimal solutions, and the one closest to K' is 6320. The discrepancy between the exact solution and the estimated solution using Theorem 2.1 is less than 0.1%, which is due to the integral constraints on K and $\frac{W}{K}$.

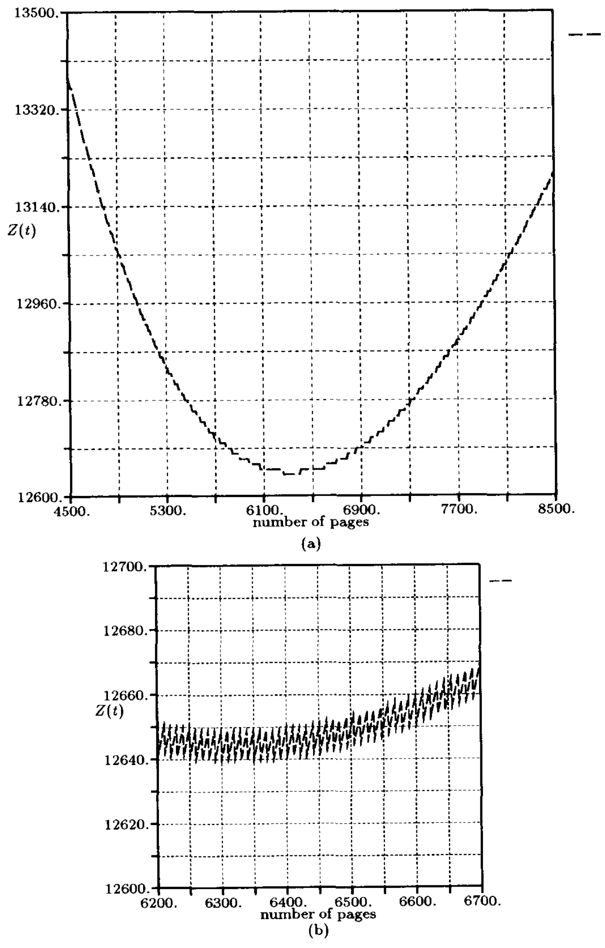


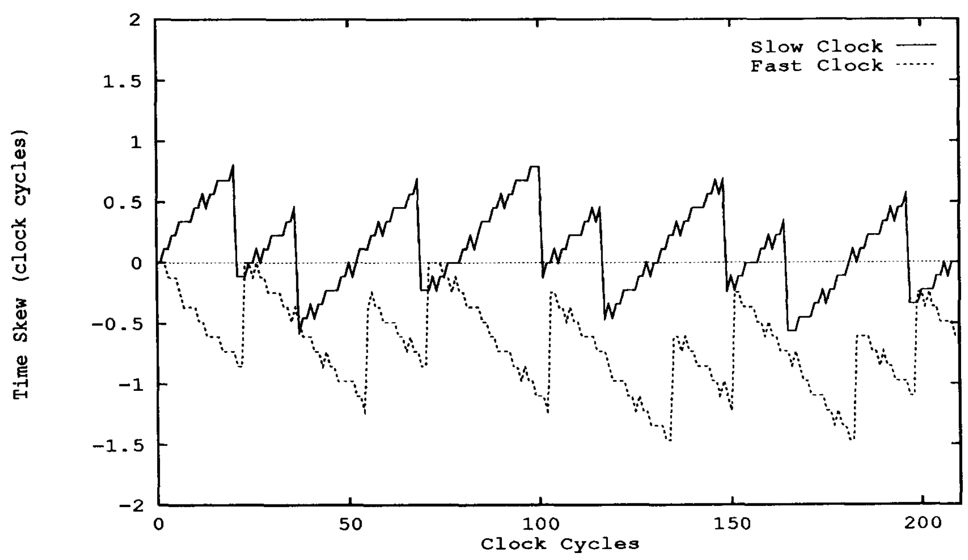
Fig. 5. The cost function of a redundant system with perfect fault detection capability. (a) A macroscopic view of the cost function, and (b) the microscopic view of the cost function around the optimal page size.

III. Experiments and Performance Results

In this section, we present an experimental implementation of the PV and compare the performance of PV under two cache replacement policies. To validate our key design concepts, we developed prototypes of a 65C816-based processor triad and an ADPLL circuit, and logic simulation of the PV using the Galaxy logic simulation tool [22].

The 65C816-based processor triad was built with an ADPLL circuitry, and data could be voted on every synchronization cycle. Upon detection of an error, an interrupt signal will be generated for each processor to handle the event of error detection and to force the faulty unit to be retired. The ADPLL was implemented with three programmable generic array logic (GAL), the 20V8 GALs, each of which was associated with one 16 MHz (high-speed) clock source. Each of the high-speed clock sources was down scaled, called a *synchronization clock*, by a divide-by-16 counter to be broadcast for mutual synchronization. Each ADPLL adjusted its rate when the skew between the synchronization clocks exceeded the duration of two high-speed clock pulses. Over the several months of our experiments, the ADPLL circuits showed remarkably stable behavior. No single loss of synchronization event was registered, and the time skew between different sources was maintained within two high-speed clock pulses. The only constraints on the performance of the ADPLL was the delay of logic circuits. It was found in a similar logic simulation that much higher speed clocks can be synchronized using high-speed logic devices.

We then examined the performance of the PV through logic simulation with the Galaxy CAD tool [22]. In our simulation



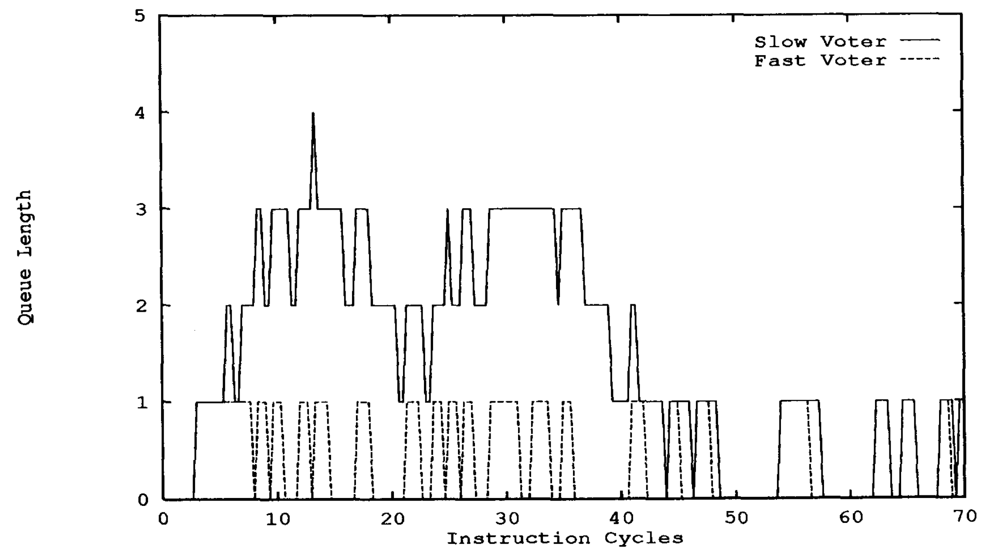


Fig. 6. (a) The time skew between the three clock sources, and (b) the queue lengths of the PVs for different write rates.

each processor has four registers and an ALU, and three processors were grouped into a processor triad. The instruction set included load/store of data between main memory and registers, and simple ALU operations between the registers. Different memory write frequencies were randomly generated in test programs ranging from 10% to 50% in our experiments. The three clock cycles of the processors were set at 164, 170, and 176 simulation time units, a 7% frequency difference between the fastest and slowest clocks, where the simulation time unit can be scaled to different physical time units as needed.

The time skew between the three clock sources is plotted in Fig. 6(a), in which the fast and slow clocks are referred to as the clock sources of 164 and 176 time units, respectively. The simulated PV had an 8-word FIFO buffer and a majority voter. The PV was controlled by a simulated control unit so that the effective propagation delay in the majority voter could be altered in the range of twice faster or slower than one instruction cycle.

First, we studied the impact of clock skew on the clocking effect in the data buffer. For clock skew less than two clock cycles we found no significant impact on the queue length. Two separate experiments were run to examine the effect of different voting latencies on the queue length. In the first experiment, the PV allowed a vote to take place in one half of the execution time of a load/store instruction. As shown in Fig. 6(b) which represents a random snapshot of queue lengths of the PV with the write ratio set to 50%, the queue length was at most one under very high write ratios. In the second experiment, each vote takes one and half of the instruction execution time. On some rare occasions where write operations occur consecutively, up to five outstanding data items were recorded in the PV. It should be noted that, when the cache memory is added to each processor, the main memory read/write ratio is expected to be reduced, and thus, the queue lengths are expected to be further reduced.

We examined the fault masking capability of the PV using a simple fault injection circuit. In our experiment, only one voter was used for data voting, and the voter is driven by the fast clock source. After a fault was injected, the fault detection (masking) latency was recorded as part of the simulation output, and then manually calculated due to the lack of an automatic event trace registration facility.

Different faults were injected to write-control signals, address lines, data lines, and the clock signals of the target system. The processor was set to execute a load or store instruction in four cycles, and the data manipulation instructions in three cycles. In most of the simulation runs the voter cycle time was set to be less than one instruction cycle to simulate fast cache memories. Therefore, the main part of the voting latency was contributed by the time required to write the voted data to memory. For slower memories, the voting latency was simulated to take twice as long as one instruction cycle. The data collected in these experimental runs assumed that the voter took six clock cycles to complete the voting process.

Transient, intermittent, and permanent faults were tested. Injection of permanent faults was relatively straightforward. Permanent faults were created by the fault injector after a random period, and the target signal line was set to either a stuck-at-0 or -1, and then, the number of cycles after the fault occurrence was monitored and recorded. Although transient faults were also injected, only a small fraction of faults were detected when they caused data faults. Almost all transient faults on the address bus, data bus, or control signals did not create any error in the computational results, and thus were not detected in the voting process. Hence, results on only a few instances of transient faults being detected were not reported as they lack statistical significance. Intermittent faults were created by forcing a stuck-at-1 (-0) after a few clock cycles. Clock faults were created by forcing the faulty module's clock line to be stuck-at-1 or stuck-at-0. A similar phenomenon existed initially in the intermittent fault simulation runs, but eventually many more fault detections were registered, and thus were reported here. The detection times of different faults are plotted in Fig. 7.

In general, the fault detection time first decreased with the write ratio, and then increased after a certain threshold. The reason for this trend is that, when the write ratio was very low, it took a long time before the faulty data could be voted on by the PV. With a further increase of write ratio, the injected faults might be overwritten by new write commands, thus increasing the fault detection time. In all cases, the variances between different runs of experiments are fairly large, indicating the unpredictable nature of fault detection/masking. It is also observed that the watchdog timer was triggered in many cases before the data voting could take place, because of the clock skew. Therefore, the watchdog timer needs to be properly adjusted to avoid excessive false alarms.

We now compare the performance of PV under the write-back and write-through cache replacement policies using the average memory access time T_M as the performance parameter. T_M is determined by the voter architecture, clock skew, and the speed difference between the main and cache memories. The main memory is assumed to be k-way interleaved, and no performance loss is assumed in case of a cache read-hit. The processor is blocked in case of a cache miss. Under the write-through protocol, data items to be written into the cache and main memory are first voted on by the PV, and the voted result will be stored into the main memory if no error is detected. Since the main memory is assumed to be interleaved, there is a random waiting time before the data item can be stored into the main memory, and all subsequent outputs from the PV will be blocked.

For its performance analysis, a PV can be modeled as a queueing system, where the write-buffer of the PV, and the main memory are, respectively, represented by the queue and the server. The waiting time T_v for redundant data to become ready in the PV is determined by the clock skew between the redundant processors. Let S denote the maximum time skew between redundant units, and clocks are adjusted once every t_δ seconds, then the average waiting time due to the clock skew can be expressed as

$$E(T_v) = \int_0^S \frac{1}{t_\delta} t\,dt = \frac{S^2}{2t_\delta}.$$

The average waiting time before the voted data can be stored into the main memory is

$$\sum_{i=0}^{k} \frac{1}{k}\frac{i}{k} t_c = \frac{t_c}{2},$$

where t_c is the main memory cycle time. The average memory write time is thus $E(T_w) = \frac{t_c}{2} + \frac{S^2}{2t_\delta}$. Since we know the average service time of the server and the arrival rate of the customers (write-requests), we can get the average number of data items waiting in PV as $Q = \lambda E(T_w) = \lambda(\frac{t_c}{2} + \frac{S^2}{2t_\delta})$, where λ is the arrival rate of memory-writes.

We note that under the write-through protocol, the processor can proceed with its computation without performance loss in case of cache hit, as long as the PV is not full, so that the data item can be directly loaded into the PV. Processor performance loss occurs only in case of a cache miss, since the PV must be flushed before a new cache block can be read in, and thus, the performance loss is affected by the number of outstanding data items to be voted on in the PV. Assuming that CPU blocking has a negligible effect, the average memory access time can be expressed as

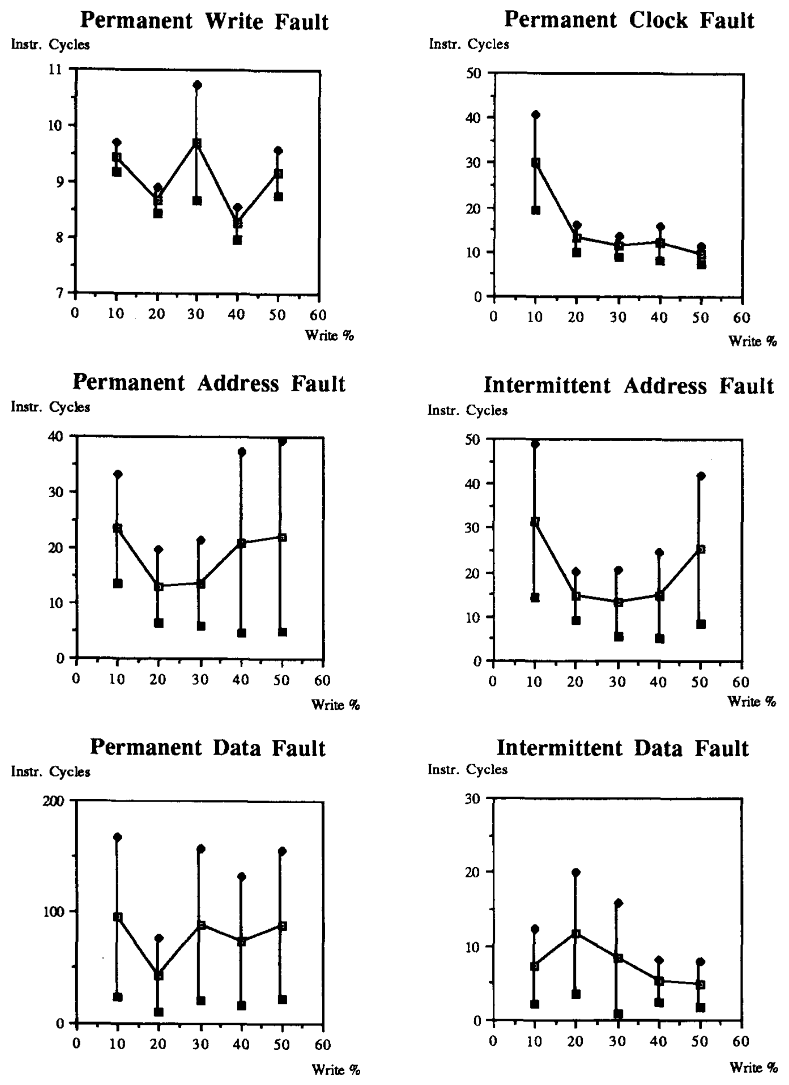


Fig. 7. The fault detection times of a processor triad for different write rates.

$$T_M^T = ht_{cache} + (1-h)\left[Q\frac{t_c}{2} + (\frac{1}{2}+\frac{b}{k})t_c + t_{cache}\right],$$

where h denotes the cache-hit ratio, b denotes the cache block size, and P_w denotes the fraction of memory-write instructions. The three terms multiplied by the factor $(1 - h)$ denote the average delay for flushing the PV, reloading the cache, and accessing the needed data item, respectively.

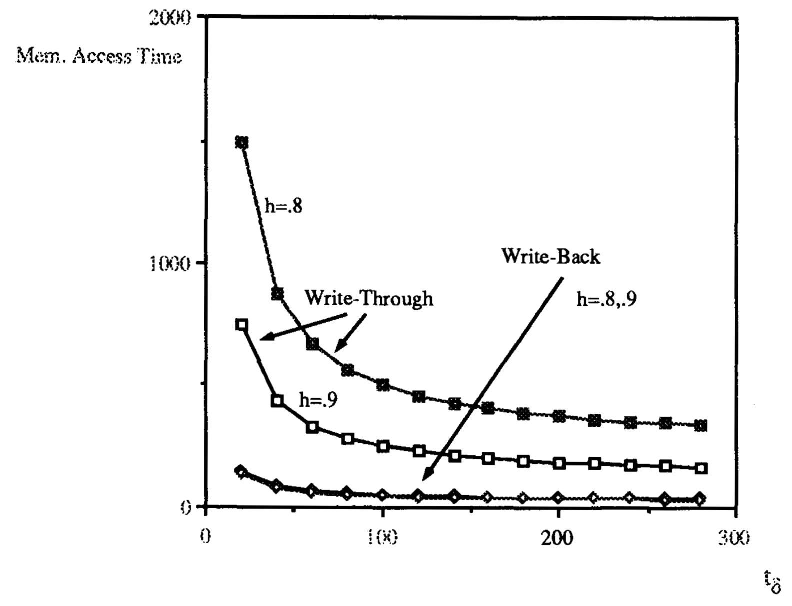
We now derive the average memory access time under the write-back cache coherence protocol. In the write-back protocol, a dirty block will be written into the main memory when a cache miss occurs. Assuming that cache blocks are randomly replaced, the average memory access time under the write-back protocol can be expressed as

$$T_M^B = ht_{cache} + P_{dirty}\left(E(T_W)+(b-1)\frac{t_c}{k}\right)+(1-h)\left[\left(\frac{1}{2}+\frac{b}{k}\right)t_c + t_{cache}\right], \quad (1)$$

where P_{dirty} is the probability that the cache block to be replaced is dirty. P_{dirty} is derived as follows. Assuming that the cache consists of n blocks and all blocks have an equal probability to be accessed in each cycle, the probability of a cache miss at the ith cycle since the last cache miss is h^{i-1} $(1 - h)$. Thus, the probability of a dirty cache block being replaced is

$$\sum_{i=1}^{\infty}\left(1-\left(h\left(1-\frac{P_w}{n}\right)\right)^{i-1}\right)h^{i-1}(1-h).$$

That is, the total probability of a dirty block being replaced is



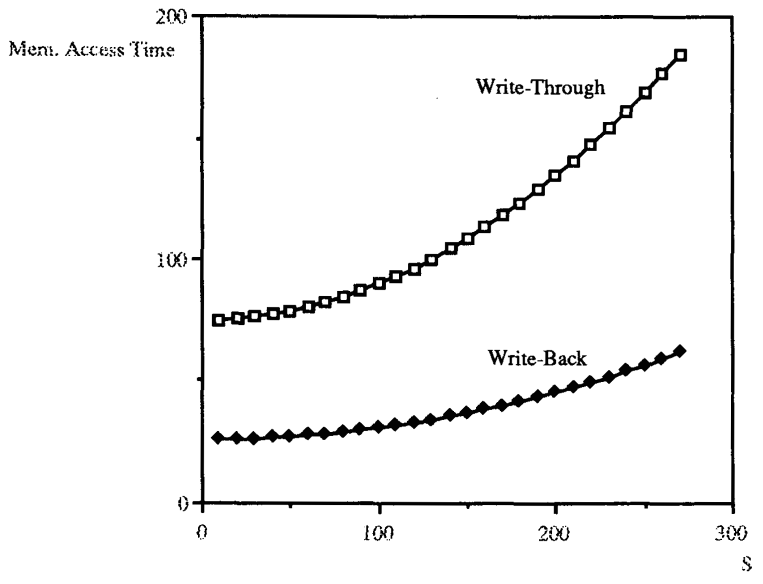


Fig. 8. The performance difference between the two cache coherence protocols.

$$P_{dirty} = \sum_{i=1}^{\infty}\left(1-\left(h\left(1-\frac{P_w}{n}\right)^{i-1}\right)h^{i-1}(1-h)\right) \quad (2)$$

$$= 1-\frac{1-h}{1-h^2\left(1-\frac{P_w}{n}\right)}. \quad (3)$$

The performance difference between the two protocols can be expressed as

$$T_M^B - T_M^T = P_{dirty}(E(T_w)+(b-1)\tfrac{t_c}{k}) - (1-h)P_w E(T_w)\tfrac{t_c}{2},$$

where $P_{dirty} \approx \frac{h}{1+h}$ when $n \gg 1$. As depicted in Fig. 8, the average memory access time of the write-back protocol is not sensitive to the time skew between the redundant units, nor to the memory-write frequency. On the other hand, the average memory access time of the write-though protocol increases sharply with the clock skew and memory-write frequency. Thus, the write-back protocol is better suited for NMR multiprocessor systems.

IV. Conclusion

In this paper, we have presented the MAT bus and the DRN as two architectural support mechanisms for dynamic reconfiguration in an NMR-based multiprocessor system. We have shown that the MAT bus can implement the fault detection, instead of fault masking, function of the PV. The MAT bus and DRN together provide a flexible platform for the dynamic grouping of processors. The MAT bus-based architecture is rationalized by the fact that faults occur much less frequently than data reads/writes. Thus, to improve the system throughput, data from redundant computing units can be put into the PV buffer, and proceed with their normal computation. Experimental results showed that the proposed scheme is feasible and introduces only a very small performance overhead. Our model also showed that combination of the PV with the write-back protocol can virtually eliminate the performance loss resulting from data voting.

Appendix A: Proof for Theorem 2.1

We prove Theorem 2.1 in four steps. We first exploit the basic attributes of the probability function on the number of faulty pages in the system, in Lemma A.1. Then, we will find the maximum number of faulty pages that can be realigned without violating the timing constraint, in Lemma A.2. In the third step, we will show in Lemma A.3 that if the system has a large number of memory pages, the probability of f pages being faulty is insensitive to the change of page size. So, as the last step, we can conclude that for an arbitrary large integer K, we get f_K close to f_{K^*}, where K^* is the optimal page size being calculated. Therefore, even though K^* is unknown, we can still estimate f_{K^*} as f_K. By plugging f_K into the objective function and then taking its derivative, we can get the approximate optimal value of K^*, i.e., Theorem 2.1. Details of these steps are explained as follows.

The probability that f pages become faulty in the system by time t is $Pr(F(t) = f) = \binom{K}{f} R_p^{(K-f)}(t)(1 - R_p(t))^f$, where $1 - R_p(t)$ is the probability that one or more of the redundant memory pages are faulty. Let λ and q denote the failure rate of a memory word and the number of redundant modules, respectively, we have $R_p(t) = e^{-q\frac{W}{K}\lambda t}$. Let $\psi = Wq\lambda t$, and K be a fixed constant, then the conditional probability that f recovery pages need to be realigned is

$$P_K(f) = P_K\left(Z(t) = K + f\frac{W}{K}\right) = \binom{K}{f} e^{-\psi\left(1-\frac{f}{K}\right)}\left(1 - e^{-\frac{\psi}{K}}\right)^f \quad (4)$$

$K = k$ is a feasible solution if and only if $Pr(Z(t) > T) < \varepsilon$. If $T \geq Wt_v$, the recovery page design is trivial, because the memory can be easily realigned by voting on every word. If $T < Wt_v$, only a limited number of pages should be realigned. If $K^* \not\gg 1$, then an exhaustive search for K^* suffices. On the other hand, if $K^* \gg 1$, as is in most large systems, an approximate value of K^* can be found through the following simple optimization technique.

Lemma A.1: *Given ψ and K, $P_K(f)$, the probability of f faults occurring to the system, is a monotonically decreasing function of f when $\frac{K-f}{f+1}(e^{\frac{\psi}{K}} - 1) < 1$, $1 < f \leq K$. The sufficient condition for $P_K(f)$ to be a monotonically decreasing function of f is $(e^{\frac{\psi}{K}} - 1) < \frac{2}{K-1}$, $K > 1$.*

Proof: $P_K(f)$ is a nonnegative, monotonically decreasing function if $\frac{P_K(f+1)}{P_K(f)} < 1, \forall f$. Using (4), we have $\frac{P_K(f+1)}{P_K(f)} = \frac{K-f}{f+1}(e^{\frac{\psi}{K}} - 1)$, or $P_K(f)$ is monotonically decreasing if $\frac{K-f}{f+1}(e^{\frac{\psi}{K}} - 1) < 1$. Note that $0 \leq (e^{\frac{\psi}{K}} - 1) \leq 1$ when $\frac{\psi}{K} \leq 0.693$. Since $\frac{K-f}{f+1} \leq \frac{K-(f+1)}{(f+1)+1}$, and the maximum value of $\frac{K-f}{f+1}$ is $\frac{K-1}{2}$, $K > 1$, the sufficient condition for the ratio test to hold is $(e^{\frac{\psi}{K}} - 1) < \frac{2}{K-1}$, $K > 1$. □

Lemma A.2: *When Lemma 4.1 holds, $Pr(t_{ra} > K + f\frac{W}{K}) < P_K(f)\frac{1-\mu_f^{K-f}}{1-\mu_f}$, where $\mu_f = \frac{K-f}{f+1}(e^{\frac{\psi}{K}} - 1)$.*

Proof: When Lemma A.1 holds, $P_K(f + 1)/P_K(f) < \mu_f$, and $\mu_f < 1$. Since $\mu_f < \mu_{f+1}$, $\forall f$, we have

$$\sum_{i=f}^{K} P_K(i) < P_K(f)\left(1 + \mu_f + \mu_f^2 \cdots + \mu_f^{K-f}\right)$$

or

$$P\left(t_{ra} > K + f\frac{W}{K}\right) < P_K(f)\frac{1-\mu_f^{K-f+1}}{1-\mu_f}. \qquad \square$$

Note that $\psi \ll K$ holds for most realistic parameter values. When Lemma A.1 holds, and K and ε are given, $f_k = \inf f_i$, which is the maximum number of faulty pages such that $P(t_{ra} > K + f_i\frac{W}{K}) < \varepsilon, \forall i$, can be determined by applying Lemma A.2 repeatedly. The next Lemma states a key condition that can greatly simplify the optimization problem.

Lemma A.3: *If two integers K_1 and K_2 are both much greater than constants 1, ψ, and f, then $P_{K_1}(f) \approx P_{K_2}(f)$, where $P_{K_i}(f)$ is the probability that the number of faulty pages F is f, when the number of recovery pages is K_i.*

Proof: $P_K(f) = \binom{K}{f} e^{-\psi(1-\frac{f}{K})}(1 - e^{\frac{-\psi}{K}})^f$. When $K \gg f$, $\binom{K}{f} \approx \frac{K^f}{f!}$, and $e^{-\psi(1-\frac{f}{K})} \approx e^{-\psi}$. Furthermore, when $K \gg \psi$, we get $1 - e^{\frac{-\psi}{K}} \approx 1 - (1 - \frac{\psi}{K}) = \frac{\psi}{K}$. Combining these expressions together we get $P_K(f) \approx \frac{K^f}{f!} e^{-\psi}(\frac{\psi}{K})^f$, or $P_K(f) \approx e^{-\psi}\frac{\psi^f}{f!}$. That is, $P_K(f)$ is predominantly determined by f, and is insensitive to K. Thus, $P_{K_1}(f) \approx P_{K_2}(f)$. □

Lemma A.3 is valid for a broad range of K values. Note that when K_1, $K_2 \gg 1$, the values of $P_{k_i}(f)$s are very close to each other. When Lemma A.1 holds, $P_{K_1}(f_1) < P_{K_2}(f_2)$, where $f_1 > f_2$. In these examples, the system has $W = 4M$ words of memory, $q = 4$ modules, $\lambda = 0.75$ byte/10^9 hours, and $t = 150$ hours. Thus, $\psi = \lambda tqW = 1.8$. Thus, $|P_{500}(F = 1) - P_{13500}(F = 1)| < 0.05$, and $|P_{500}(F = 3) - P_{13500}(F = 3)| < 0.001$. Let K^* denote the optimal value of K, the most desirable property of Lemma A.3 is that when $K^* \gg 1$, we get $f_K \approx f_{K^*}$, and thus, K^* can be found by Theorem 2.1.

Theorem 2.1: *When $K^* \gg 1$, $K^* \approx \sqrt{f_K W}$, where K is an arbitrary integer, $1 \ll K < W$.*

Proof: From Lemma A.3, we get $P_{K_1}(f) \approx P_{K_2}(f), \forall f$. Thus, when $K^* \gg 1$, we have $f_K \approx f_{K^*}$, or f_{K^*} can be found by applying Lemma A.2 to an arbitrary K such that $P(f > f_K) < \varepsilon$. Clearly, for a given ε, $f_K \approx \hat{f}, \forall K \gg 1$, where $\hat{f}$ is some constant. The cost function $Z(t)$ to be minimized can be expressed as $\min_{K \gg 1}(K + \hat{f}\frac{W}{K})$. Since the objective function is convex when K is continuous, the optimal solution of real-valued Ks is $K' = \sqrt{f_K W}$. Then, K^* can be found by an exhaustive search in $[K' - \delta, K' + \delta]$, where δ is very small compared to K, and it is yet to be found. □

Acknowledgments

The work reported in this paper was supported in part by NASA under Grant NAG-1-1220, by the ONR under Grant N00014-91-J-1115, and by the National Science Foundation under Grants MIP-9012549 and MIP-9203895.

References

[1] B. Parhami, "Voting networks," *IEEE Trans. Reliability*, vol. 40, pp. 380-394, Aug. 1991.

[2] J. Kim, J.-C. Liu, P. Swarnam, T. Park, Y. Hao, and T. Urbanik, "The area-wide real-time traffic control (ARTC) system: A distributed computing system," *IEEE Computer Software Application Conf.*, pp. 263-268, Chicago, June 1992.

[3] J. Kim, J.-C. Liu, P. Swarnam, and T. Urbanik, "The area-wide real-time traffic control (ARTC) system: A new traffic control concept," *IEEE Trans. Vehicular Tech.*, vol. 42, no. 2, pp. 212-224, May 1993.

[4] M. Iacoponi and S. McDonald, "Distributed reconfiguration and recovery in the advanced architecture on-board processor," *Digest of Papers, FTCS-21*, pp. 436-443, 1991.

[5] J. Stankovic and D. Towsley, "Dynamic relocation in a highly integrated real-time distributed system," *Proc. Int'l Conf. Distributed Computing Systems*, pp. 374-381, May 1986.

[6] G. Barigazzi and L. Strigini, "Application-transparent setting of recovery points," *Digest of Papers, FTCS-13*, pp. 48-55, 1983.

[7] D.B. Hunt and P.N. Marinos, "A general-purpose cache-aided rollback error recovery (CARER) technique," *Digest of Papers, FTCS-17*, pp. 170-175, 1987.

[8] R.E. Ahmed, R.C. Frazier, and P.N. Marinos, "Cache-aided rollback error recovery (CARER) algorithms for shared memory multiprocessor systems," *Proc. 17th Int'l Symp. Computer Architecture*, pp. 82-88, 1990.

[9] C. Chen, A. Geng, T. Kikuno, and K. Torii, "Reconfiguration algorithms for fault-tolerant arrays with minimum number of dangerous processors," *Digest of Papers, FTCS-21*, pp. 452-461, 1991.

[10] A.L. Hopkins, T. Smith, and J. Lala, "FTMP—A highly reliable fault-tolerant multiprocessor for aircraft," *Proc. IEEE*, vol. 66, pp. 1221-1239, Oct. 1978.

[11] *The Stratus Computer System*, Stratus Inc., July 1992.

[12] *The Series 400 Sequoia Systems*, Sequoia Inc., July 1992.

[13] *FT-6100 Fault Tolerant Computer TPR Architecture*, Hitachi Ltd., July 1992.

[14] T.B. Smith, "Fault-tolerant processor concepts and operation," Tech. Report, Charle Stark Draper Lab., CSDL-P-1727, May 1983.

[15] D.P. Sieworek, et al, "C.vmp: A voted multiprocessor," *Proc. IEEE*, vol. 66, Oct. 1978.

[16] W. Greer and B. Kean, "Digital phase-locked loops move into analog territory," *Electronic Design*, pp. 95-100, Mar. 1982.

[17] *The Transputer Databook*, Inmos Corp, 1989.

[18] D.J. Adams and T. Sims, "A tagged memory technique for recovery from transient errors in fault-tolerant systems," *Proc. Teal-Time Systems Symp.*, pp. 312-321, 1990.

[19] D.P. Siewiorek, "Fault-tolerance in commercial computers," *Computer*, July 1990.

[20] S.R. McConnel, D.P. Siewiorek, and M.M. Tsao, The measurement and analysis of transient errors in digital systems," *Digest of Papers, FTCS-9*, pp. 67-70, 1979.

[21] D.P. Siewiorek and R.S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Equipment Corp., Bedford, Mass., 1992.

[22] J. Beetem, *Galaxy CAD User Manual*, Univ. of Wisconsin, Madison, Wis., 1992.

Jyh-Charn Liu (S'84-M'89) received BS and MS degrees in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 1979 and 1981, respectively. He earned the PhD degree from the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor, Mich., in 1989.

Dr. Liu has been an assistant professor in the Computer Science Department, Texas A&M University, College Station, Texas, since 1989. He was a system engineer of the Siantek Co. Taipei, Taiwan, in 1983. He was a research associate for the Real-Time Computing Lab., the EECS department, University of Michigan in 1989, and he was involved with the evaluation of the Advanced Information Processing System for NASA.

Dr. Liu's current research interests include fault-tolerant computing, parallel and distributed systems, real-time systems, and Intelligent Vehicle and Highway systems (IVHS). He is a member of the IEEE Computer Society.

Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, N.Y., in 1976 and 1978, respectively.

Dr. Shin is a professor and the director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Mich. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, N.Y. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, the Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California, Berkeley, and International Computer Science Institute, Berkeley, Calif. He was chairman of the Computer Science and Engineering Division, Electrical Engineering and Computer Science Department at the University of Michigan for three years beginning in January 1991.

Dr. Shin is the author or coauthor of more than 250 technical papers (more than 110 of these are in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. In 1987 he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989 he received the Research Excellence Award from the University of Michigan. He founded the Real-Time Computing Laboratory in 1985. He and his colleagues at the Real-Time Computing Laboratory are currently building a 19-node hexagonal mesh multicomputer called HARTS to validate various architectures and analytic results in the area of distributed real-time computing.

Dr. Shin is an IEEE fellow and was program chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), general chairman of the 1987 RTSS, guest editor of the August 1987 special issue of *IEEE Transactions on Computers* on real-time systems, program co-chair for the 1992 International Conference on Parallel Processing, and has served on numerous technical committees. He chaired the IEEE Technical Committee on Real-Time Systems from 1991-1993 and is a Distinguished Visitor of the IEEE Computer Society, an editor of *IEEE Transactions on Parallel and Distributed Computing*, and an area editor of the *International Journal of Time-Critical Computing Systems*.

Dr. Shin has been applying the basic research results of real-time computing to manufacturing related applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. He recently initiated research on the open-architecture Information Base for machine tool controllers.