REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 95	Semi-annual 7/16/94 1/15/95

4. TITLE AND SUBTITLE
1. Fault Tolerant Features and Experiment of ANTS distributed real-time system
2. ANTS: An approach for High-Performance and Ultra-Dependability

5. FUNDING NUMBERS
G:
N 00014-94-1-0479

6. AUTHOR(S)
P. Dominic-Savio, J.C. Lo and D. W. Tufts

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Electrical & Computer Engineering
University of Rhode Island
Kingston, RI 02881-0805

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Ballston Tower One
800 North Quincy St.
Arlington, VA 22217-5680

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
This report contains two papers. Paper (1) has been submitted to FTCS-25 and paper (2) has been submitted to IPDS '95.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The ANTS project at the University of Rhode Island introduces the concept of Active Nodal Task Seeking (ANTS) as a way to efficiently design and implement dependable, high-performance, distributed computing. This paper presents the fault tolerant design features that have been incorporated in the ANTS experimental system implementation. The results of performance evaluations and fault injection experiments are reported. The fault-tolerant version of ANTS categorizes all computing nodes into three groups. They are: the up-and-running *green group*, the self-diagnosing *yellow group* and the failed *red group*. Each available computing node will be placed in the yellow group periodically for a routine diagnosis. In addition, for long-life missions, ANTS uses a monitoring scheme to identify faulty computing nodes. In this monitoring scheme, the communication pattern of each computing node is monitored by two other nodes.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Dependable distributed system, real-time system, active mode operating system, high performance distributed computing	40
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Fault Tolerant Features and Experiments of ANTS Distributed Real-Time System

Patrick Dominic-Savio, Jien-Chung Lo and Donald W. Tufts
Department of Electrical and Computer Engineering
The University of Rhode Island
Kingston, RI 02881-0805

Abstract

The ANTS project at the University of Rhode Island introduces the concept of Active Nodal Task Seeking (ANTS) as a way to efficiently design and implement dependable, high-performance, distributed computing. This paper presents the fault tolerant design features that have been incorporated in the ANTS experimental system implementation. The results of performance evaluations and fault injection experiments are reported. The fault-tolerant version of ANTS categorizes all computing nodes into three groups. They are: the up-and-running *green group*, the self-diagnosing *yellow group* and the failed *red group*. Each available computing node will be placed in the yellow group periodically for a routine diagnosis. In addition, for long-life missions, ANTS uses a monitoring scheme to identify faulty computing nodes. In this monitoring scheme, the communication pattern of each computing node is monitored by two other nodes.

* Jien-Chung Lo is the contact author.
Tel: (401) 792-2505
FAX: (401) 782-6422
Email: jcl@ele.uri.edu

Keywords: distributed computing, MTTF, highly available systems, fault-tolerant distributed operating system, real-time systems.

This work is supported by the Office of Naval Research under contract N0014-94-1-0479.

1 Introduction

The ANTS project at the University of Rhode Island explores and evaluates the concept of Active Nodal Task Seeking (ANTS) as a way to efficiently design and implement dependable, high-performance, distributed computing. Real-time radar, active sonar, and electronic warfare systems are examples of systems which require ultra-dependability and high performance. Distributed systems are potentially dependable since there are built-in redundancies. Many fault-tolerant distributed system designs have been proposed, and some prototypes have been built. However, we would like to point out that none of these existing systems, both theoretical and experimental, can efficiently achieve both high-performance and ultra-dependability at the same time. For the above applications, a real-time requirement is also important. We need to guarantee that sufficient computational power will be available, even in the presence of component failures, and can be allocated to complete all high priority tasks within the real-time deadlines. The only direct predecessor for ANTS is the Safeguard multiprocessor system [1].

Distributed systems are inherently fault-tolerant to failures of individual components. However, utilizing this intrinsic fault tolerance is not a straightforward task. Dependable distributed systems still need special hardware or software designs to achieve the desired level of dependability. The Advanced Architecture On-board Processor by Harris Corporation [2, 3] uses self-checking RISC processors and a chordal ring. The Advanced Automation System by IBM [4, 5] needs redundant components in each subsystem due to the nature of its wide area distribution. The on-board computer of the Japanese satellite Hiten [6] uses stepwise negotiating voting, which is a combination of mutual checking by data comparison and self-checking. In some cases, software redundancy is used. For instance, Delta-4 [7] uses active replication of software programs to ensure fault tolerance. A similar strategy is used in Manetho [8], where each application process is replicated by a set.

We have proposed previously a novel locally *active* distributed system to efficiently design and implement dependable, real-time signal processing for detection and localization in active sonar and radar [9]. We called this distributed computing concept Active Nodal Task Seeking (ANTS). The fault tolerance mechanisms are built-in at the ANTS kernel level. We report in this paper the fault-tolerant features and the first implementation of ANTS system. We also report here the results from fault injection experiments. We demonstrate that the ANTS system can achieve our dependability goal, an extremely long MTTF, without significant

loss of performance.

2 An Overview of ANTS Distributed Real-Time System

In an ANTS system, an idle computing node will seek out the information it needs about pending, high priority work from partly prescribed, partly updated task tables, instead of waiting for other computing nodes to send it a job to do. Naturally, predecessor/successor conditions like those in a data flow architecture must be checked by the node. The advantage of this concept is that a true distributed system is guaranteed. All programs, including operating system, applications, and monitoring or diagnostic programs, can be executed distributively. A failed computing node will not influence the system performance as long as the *n+1* redundancy policy is enforced, *i.e.* the number of available computing nodes is always one greater than the number of required computing nodes.

2.1 General Attributes

There is no need for special hardware design to ensure the dependability of an ANTS system except perhaps in its internode communication network. However, some extra work by the programmer and/or an unconventional compiler is needed on a one-time basis, before the compiled program can be executed on ANTS. Some other attributes of ANTS are as follows [9]:

- On-line error checking is simple and straightforward. The two extreme failure modes: *fail-silent* and *fail-uncontrolled* [7] are easily detected since the failed computing node will not be actively seeking a new task.
- Fault tolerance can be attained without seriously degrading the system performance.
- There is no need for synchronization of computing nodes or programs. Since every computing node is an active unit, the ANTS system is an asynchronous system. The asynchronous nature of the operation also allows an easy implementation of recover from failure process.

A high-performance distributed computing system is not necessarily a good real-time system. Studies have shown that only 50% of the nodes, on average, in a distributed system

are busy at any given time [10]. This is a serious drawback if the system is to be used in real-time applications. Clearly, the effort to guarantee a higher computing node utilization can also help to enhance the success of real-time executions. In ANTS, the needed programs will be first carefully partitioned and compiled such that each segment of sub-program (task) contains only continuous executable codes with a fixed limitation on execution time. This one-time effort is worthwhile since no run-time scheduling or task assignment will be needed later.

2.2 The Implementation of ANTS

The first implementation of ANTS is carried out in a configuration as shown in Figure 1. The computing nodes in this case are Sun workstations, and the communication bus is an Ethernet. Of course, ANTS can be implemented on any type of parallel processing configurations. The choice of this particular system organization is due to the availability of hardware facilities.

The ANTS kernel is built on top of the SUN OS and the TCP/IP [11] communication protocol. A detailed view is shown in Figure 2. There are four major modules in the ANTS kernel: System Manager, Fault Tolerance Monitor, Task Scheduler, and Inter Node Communication Control. The ANTS kernel controls the activities at each node. For this paper, we shall concentrate more on the System Manager and the Fault Tolerance Monitor.

3 Fault Tolerant Features

The fault tolerant features reported here are those implemented at the ANTS kernel level for the experiments. Data manipulation errors are not detected by the fault tolerant techniques described here. They are detected by concurrent error detecting techniques to be implemented at the task level. For instance, in a high availability application, reasonableness checking on the computed results can be easily imposed. For high reliability applications, duplicated or triplicated tasks can be used.

3.1 Run Time Partitioning

During run time ANT nodes are partitioned into three groups: the up-and-running *green* group, the stop-and-checking *yellow* group and the *red* group for discarded faulty nodes. The

number of computing nodes allocated for the green group must always be greater than the minimum requirement at any given time. The rest of the computing nodes are then in the yellow group unless they have been found to fail, in which case they are removed to the red group. The grouping is based on the status of the computing nodes. Therefore, there is no visible physical grouping, and no need for special hardware design for switching between the two run-time groups.

There are two ways that an ANT node may be moved from the green group to the yellow group: (1) when it is time for a routine self diagnosis, or (2) when a node has been identified as faulty. Since each ANT node seeks its own work for itself, placing an ANT node in the yellow group is accomplished by making the diagnostic program the highest priority task for it through a timer. An ANT node may be moved from the yellow group to the green group if the diagnosis result proves fault-free.

All faulty nodes are placed in the red group. When no manual repair is available, the nodes will be indefinitely remained in the red group. A node in the red group can be moved back to the green group only if it has been repaired.

3.2 Concurrent Error Detection

Since each ANT node has an active role in seeking out work, malicious failures, such as fail-silent or fail-uncontrolled [7], are easy to identify. A fail-silent or fail-uncontrolled ANT node will not ask for a job for an extensive period of time. The concurrent error detection at the ANTS kernel level is based on *bus eavesdropping*. This requires a communication interconnection system where broadcast is cheap. Ethernet based networks are ideal examples of such an interconnection medium.

If a failure occurs after a new job has been acquired, then the ANT node will not report a completion of the task within a reasonable time. For easy real-time scheduling as well as concurrent error detection, we partition job into tasks such that each task can be completed within a fixed time frame. Therefore, time-out detection can be used as the first layer of concurrent error detection mechanism.

At the ANTS kernel level, we implement a monitor scheme, ANTS Fault detection and Recovery Protocol (AFDRP), that is similar to that proposed in [12]. Each ANT node monitors its two neighboring nodes. Conversely, each ANT node is monitored by its two neighboring nodes. All ANT nodes are assigned with node addresses. These addresses are

used in run time to determine the neighboring nodes.

For the case shown in Figure 4, let us consider node M. Each time M receives a message from H or L indicating that they are taking up a task for execution M updates its *Last Activity* record for that node. This forms one phase of the protocol. Each time node M finishes executing a task it enters the *Verification* cycle, as shown in Figure 5. Every node in the system runs a verification cycle to verify the correct operation of its upstream neighbor (H in this example). The verification cycle begins by checking if the node under test has been silent for more than a predefined timeout factor. If this is not the case then node H passes the test and M exits the verification cycle. If node M finds that node H has remained silent for more than the timeout factor, it proceeds with the *Confirmation Cycle.*

The confirmation cycle begins with node M communicating with the other node monitoring H (which is H2 in this example), requesting a verification of the activity of node H. When H2 receives this request it looks up its database and sends back a reply to node M. The reply consists of a *Concur* or *Differ* message depending on the activity report of node H at H2. On receiving a *Concur* reply from node H2, M enters the *Recover* and *Restart* (R&R) cycle, as shown in Figure 6. Figure 7 shows a more detailed flowchart of the removal of faulty node. The confirmation cycle effectively prevents faulty nodes from removing operational nodes from the network.

Beside the above protocol, other checking functions such as periodic self-diagnosis, memory parity checks, cyclic redundancy checks on inter node messages and hardware integrity checking can also be selectively done depending on the available resources. Detection of any faults during these checks also triggers the recovery and restart cycle.

3.3 Recovery and Re-Enlistment

Once the concurrent error detection mechanism successfully detects a fault, it triggers the recovery and restart (R&R) cycle. The first step in the R&R cycle consists of M, as in previous example shown in Figure 4, broadcasting a *Remove From Net* (RFN) message for node H. ANT nodes that receive this message enter different recovery modes based on their logical position (ANTS Address) in the network. If node H is partially functioning and receives this message it immediately enters a full diagnosis cycle, where it undergoes a thorough test of itself. If the RFN message is received by a node that is monitoring the node that is being removed, the node resets itself to monitor its new neighbor.

All the ANT nodes modify their *System Information Block* (SIB) to reflect the removal of the node in question. The next step in the R&R step consists of re-enlistment, this step is carried out only if a functional ANT node is available from the yellow group. If an ANT node S is available, node M brings it on-line by first allotting it the address of the removed node and proceeds to initialize the new node by providing it with the current Task List and other system parameters. Once on-line, node S starts normal execution which includes the monitoring of its adjacent neighbors. While providing the new node with the Task List, node M inserts the task that failed to complete on node H on top of the list, this effectively results in the task being restarted on node S. This marks the completion of the *Recover and Restart* cycle and the system returns to a normal state. If additional ANT node is not available, node M skips the reenlistment cycle and simply restarts the aborted task itself.

Since intermittent and transient failures occur more frequently during run-time [13], especially in modern VLSI-based systems [14], there is no need to remove a component once a failure has been recorded. The record keeps at each ANT node will indicate the number and frequency of failure history. If an ANT node fails frequently, it will be removed from the system to the red group. This situation may be induced by either of the following scenarios: the node is on the brink of a permanent failure and thus the increasing frequency of transient failure occurrences; or the failure is undetectable by the diagnostic program. It is impossible to design a diagnostic program with a 100% *realistic failure* coverage, although it may guarantee a 100% coverage on modeled failures.

3.4 Error Handling Capability

The monitoring scheme discussed above not only detect single isolated faulty nodes but multiple faulty nodes as well. In the following we shall use examples to demonstrate the detection of multiple node failures.

For the isolated node failures shown in Figure 8, the AFDRP has no trouble in detecting them. For each failed node, the two nodes that perform the monitoring are both fault-free. The communication can be established for the verification cycle and proceed directly to the R&R cycle.

Consider the multiple (chained) node failures case shown in Figure 8 where nodes 2 and 3 are faulty. Node 1 discovers the fault in node 2 during the *Verification* cycle and requests confirmation from node 3. While requesting confirmation from node 3, node 1 starts a timer

and if it does not receive a reply from node 3 before the timer expires it proceeds to run a verification of node 3 by requesting a status report of node 3 from node 4 and the cycle continues.

Based on the assumptions made previously, we can see that in this example node 1 will not receive a valid reply within the timeout period and node 4 will concur with node 1 about the failure of node 3. Now node 1 begins the R&R cycle. This verification process can be extended to handle as many *Chained* or *Multiple Consecutive Faults* as necessary. By extending the above two cases it is seen that any combination of the above two cases will be successfully handled by the protocol. It is also seen that this combination covers all possible fault patterns.

The main advantage of the above scheme is that the monitoring of nodes and the fault detection process add no additional traffic over the communication network. The protocols function by monitoring other system communication messages and hence fault tolerance is achieved at the least communication expense.

4 The ANTS System Manager and Task Scheduler

In this section, we provide a brief description of the ANTS system manager (ANTS-SM)and task scheduler.

4.1 Contention Resolution in Task Scheduler

The distributed task scheduler at each computing node select the next task to be executed from its own copy of task list. Because of the communication delay, it is possible that several computing nodes may select the same task to be executed near simultaneously. If such scenario occurs, the task scheduler will receive message from other computing nodes stating the the task has been selected for execution at other nodes. As shown in Figure 7, the timestamps indicating the time the task is selected are compared. The computing node with a higher number in the timestamp will abandon the execution and will proceed to select another task from the task list.

This simple mechanism has been shown to be effective during experimental runs. It is also quite efficient since only very little overhead in comparing the timestamps is needed.

4.2 Synchronization and the System Clock

In ANTS there is no need for highly synchronized clock mechanisms. The ANTS nodes run in loose synchronization, the only need for a synchronized clock in ANTS arises when timestamps have to be generated. The ANTS Task scheduler is designed such that even for the timestamps a high degree of synchronization is not required.

Based on these requirements the ANT-SM maintains a simple clock that counts in milliseconds. The clock is initialized during start up and is periodically checked at long intervals with neighboring nodes to trap possible erroneous values (this is done as part of the concurrent error detection mechanism at the kernel level). Other than this the precision of the system clocks is such that it does not need to be adjusted for periods far exceeding the system uptime.

4.3 ANTS Address Resolution and Initialization Protocol

This protocol is concerned with the process of initiating the entrance of a node into the *green group*. First the node that intends to enter the network randomly picks a slot from the list of valid ANTS addresses, it then broadcasts a query with the chosen ANTS address as part of the message. There are two possible outcomes of the query, first the requested ANTS slot may be vacant in which case one of the nodes in the network, *i.e.*, the one with the next higher ANTS address assigns itself the task of initiating the incoming node into the network. The task of initialization includes the transfer of the current task table and the System Information Base. Also if the fault tolerance feature is enabled the neighboring monitoring nodes reassign themselves to monitor the new node. The monitoring mechanism in the new node becomes active and starts monitoring its adjacent nodes.

If the selected ANTS address posted in the query clashed with that of an ANTS node that is already in the network then that node by default becomes the initiator. This node refers to its SIB and picks a valid ANTS address that is not in use and assigns it to the incoming node. Also, as in the previous case the initiating node transfers all the necessary information to the new node.

4.4 Debugging and System Diagnosis Features

The ANT-SM also provides a run time window display of the functioning of the system. Some of the data that is displayed includes the first 10 tasks on the task list, the currently executing task, the status of INC and the list of active nodes in the system. The display basically provides continuous status information of the running system besides showing some cumulative data such as the total traffic over the communication network and number of tasks executed.

The other necessary system functions in the prototype are handled by the host operating system the ANTS kernel is running on. In future versions it would be advantageous to completely eliminate the host operating system interface to the hardware and implement the required features taking into account the requirements of the ANTS system.

5 Experimental Results

The ANTS prototype was tested on Sun Sparc workstations running Sun OS. The ANTS communication mechanism was layered on top of the Internet Protocols and the intercommunication network used was an Ethernet LAN running at 10 Mbits per second. The ANTS code was written in C++ and compiled using the GNU C++ compiler. The test application program was emulated using timing loops and an inter task data transfer size of 1.5KB was assumed. Predecessor and successor relationships of tasks are either generated randomly or are assumed to resemble that of the FFT computation. In addition, some randomly generated tasks with randomly generated data dependency relationships are also used. Due to the limitation in the number of available nodes, the test runs were limited to a maximum of nine nodes. Tests were done for three different *Task Slice Time* factors and a comparison of performance with and without fault tolerance was also made to study the performance trade off implications of using the ANTS Integrity Maintenance Protocols.

5.1 Fault Tolerant Features Verifications

The concurrent error detecting mechanism and other integrity maintenance protocols were tested using simulated faults. The tests included cutting off the power supply to nodes or to parts of the hardware, intentional triggering of faults by transmission of invalid data over the bus and simulation of memory parity errors. Isolated and multiple faults were simulated

to fully test the correctness of the monitoring scheme. The observed results conformed to the specified requirements in that each fault free node reaches an accurate diagnosis of the fault conditions of the remaining nodes, without any restriction being placed on the number of faulty nodes or fault patterns.

The fault tolerant mechanism always correctly reconfigured the network in all the test cases. Under extreme load conditions, it took the ANTS system (with 9 nodes) a maximum time of one minute to reconfigure itself to a stable state. Under nominal conditions of load and faults the mean reconfiguration time was about 10 seconds. The reconfiguration time is the time interval between the occurring of a fault and the time the network becomes fully operational again, *i.e.*, the faulty nodes have been moved to the yellow group and available ANT nodes at the yellow group are re-enlisted to the green group. The reconfiguration time is related to the number of faults occurring at the same time and to the communication network latency.

Tables 1 and 2 show the typical reconfiguration times observed for the *isolated faults* and *chained faults* cases, respectively. The rather irregular variations in the reconfiguration time is due to the fact that, the monitoring protocols share the communication bus with the rest of the system. In such a situation the reconfiguration time is affected to a large extent by the application tasks that are running. The number of simultaneous or near simultaneous faults were limited to 6 in all the test cases. With more than 6 simultaneous faults and at high load levels the communication bottleneck was found to cause the system to become unpredictable.

5.2 ANTS Performance without Fault Tolerant Features

Figure 9 shows the performance figures obtained from the test results. During these test runs, the AFDRP or the Fault Tolerance Monitor is disabled. The modulation in programming ensures that these results are obtained with no interference from the disabled modules. The following inferences can be drawn from the results:

- First, as is to be expected the speed up factor is not linear. The speedup obtained gradually decreases with the addition of more nodes. The decrease in speed up is more pronounced with an increase in the number of nodes beyond 6. This effect is due to the traffic bottleneck imposed by the single communication channel. Addition of more communication channels or an increase in bandwidth of the communication media is

necessary to achieve a better performance with more nodes in the system.

- Another noticeable aspect of the performance figures is the higher linearity in speed up with higher Time Slice factors, *i.e.*, task execution time. This is evidently due to the fact that less overhead is involved with larger task fragments. The time slice factor will however have to be limited to a reasonable value to facilitate faster trapping of faulty nodes If the time slice factor were to be too large, a faulty node could unrecoverably damage the cooperative processing environment of the ANT system.

- A third prominent feature is that with a large number of nodes in the system the speed up tapers of to an almost constant value. This value is a function of the total available bandwidth of the communication medium. This compares well with other distributed computing systems where beyond a certain point the speed up curve inverts and results in worsening performance [15]. This is due to the fact that with more nodes there are more messages that contend for the communication medium. In ANTS the communication traffic is not adversely affected by the number of nodes in the system.

We also observe that the average communication delay is about 200 to 250ms. There is a strong relationship between the average task execution time, the average communication delay, and the speed up factor. Of course, when a high speed network is available, a shorter task execution time can be used.

5.3 ANTS Performance with Built-In Fault Tolerant Features

We then record the performance figures of ANTS with AFDRP enabled. The test run results are shown in Figure 10. Fig 10 clearly shows that one of the major design goals has been achieved. The ANTS is designed to exploit the inherent fault tolerant features of the distributed system. We expect that a successful implementation of ANTS should have a minimum impact on the system performance. Test runs yielded a figure of 5% for the performance degradation due to the addition of fault tolerance. We believe that this low figure is a strong evidence to support that the inherent fault tolerance has been well utilized.

We have shown previously in [9] that an ANTS system can achieve an extremely long MTTF. Part of this MTTF improvement is due to the re-enlistment of ANT nodes with transient type failures. We note that the ratio of transient faults to all faults depends to

a large extent on environmental conditions. In a documented case [6], the transient faults account for 20% to 33% in low earth orbit conditions.

This clearly brings out the effectiveness of the recovery process of the ANTS fault tolerant mechanism. The penalty paid for obtaining such an improvement in MTTF is a minimal trade-off in performance, as shown in Fig 7. The loss of performance is a function of the number of nodes in the system and the mean performance degradation with 4 and 8 nodes was 2% and 5%, respectively. This is relatively small compared to the magnitude by which the recovery and re-enlistment process extends the useful life of the system.

6 Conclusions

We have presented in this paper the first implementation of ANTS dependable, distributed, real-time system. Experimental results shows that the fault tolerant features built-in to the ANTS kernel function properly. Furthermore, the built-in fault-tolerant features have a relatively little impact on the overall system performance. The ANTS has been designed as a high availability system with an extremely long MTTF [9]. The results shown in this paper also indicate that ANTS can achieve high performance by fully utilizing the useful computing resources.

Finally, we remark here that ANTS is an adaptable system. The fault-tolerant features reported here are those implemented at the kernel level. For different mission type, other error checking schemes can be implemented at the task level, such as reasonableness checking, duplication, or triplication of tasks.

References

[1] AT&T, "Safeguard supplement," *The Bell System Tech. J.*, 1975.

[2] M. J. Iacoponi and D. K. Vail, "The fault tolerance approach of the advanced architecture on-board processor," in *Proc. 19th Int'l Symp. Fault-Tolerant Comput.*, pp. 6–12, June 1989.

[3] M. J. Iacoponi and S. F. McDonald, "Distributed reconfiguration and recovery in the advanced architecture on-board processor," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 436–443, June 1991.

[4] F. Cristian, B. Dancey, and J. Dehn, "Fault-tolerant in the advanced automation system," in *Proc. 20th Int'l Symp. Fault-Tolerant Computing*, pp. 6–17, June 1990.

[5] T. R. Dilenno, D. A. Yaskin, and J. H. Barton, "Fault-tolerant testing in the advanced automation system," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 18–25, June 1991.

[6] T. Takano, T. Yamada, K. Shutoh, and N. Kanekawa, "Fault-tolerant experiments of the "Hiten" onboard space computer," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 26–33, June 1991.

[7] M. Chereque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron, "Active replication in Delta-4," in *Proc. 22nd Int'l. Symp. Fault-Tolerant Comput.*, pp. 28–37, July 1992.

[8] E. N. Elnozahy and W. Zwaenepoel, "Replicated distributed processes in Manetho," in *Proc. 22nd Int'l Symp. Fault-Tolerant Computing*, pp. 18–27, July 1992.

[9] J. C. Lo, D. W. Tufts, and J. W. Cooley, "Active Nodal Task Seeking (ANTS): an approach to high-performance, ultra-dependable computing," *to appear in IEEE Trans. Aerospace and Electronic Syst.*, 1995.

[10] Z. M. Wojcik and B. E. Wojcik, "Rough grammar for efficient and fault-tolerant computing on a distributed system," *IEEE Trans. Software Eng.*, vol. 17, pp. 652–668, July 1991.

[11] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP*, vol. 1. Prentice-Hall, NJ, 1988.

[12] R. P. Bianchini and R. W. Buskens, "Implementation of on-line distributed system-level diagnosis theory," *IEEE Trans. Comput.*, vol. 41, pp. 616–626, May 1992.

[13] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM I. Res. Develop.*, pp. 20–27, January 1976.

[14] Y. Savaria, N. C. Rumin, J. F. Hayes, and V. K. Agarwal, "Soft-error filtering: A solution to the reliability problem of future VLSI digital circuits," *Proc. IEEE*, vol. 74, pp. 669–683, May 1984.

[15] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Computer*, pp. 57–69, November 1980.

Table 1: Reconfiguration times for multiple *isolated* faults

Number of Isolated Faults	Reconfiguration Time(in Seconds)
1	2
2	3
3	3
4	4
5	4
6	5

Table 2: Reconfiguration times for multiple *chained* faults

Number of Isolated Faults	Reconfiguration Time(in Seconds)
1	N/A
2	5
3	15
4	17
5	21
6	28

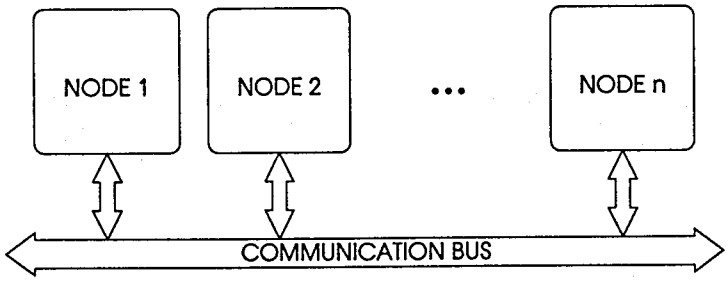


Figure 1: Architecture of the ANTS experimental system.

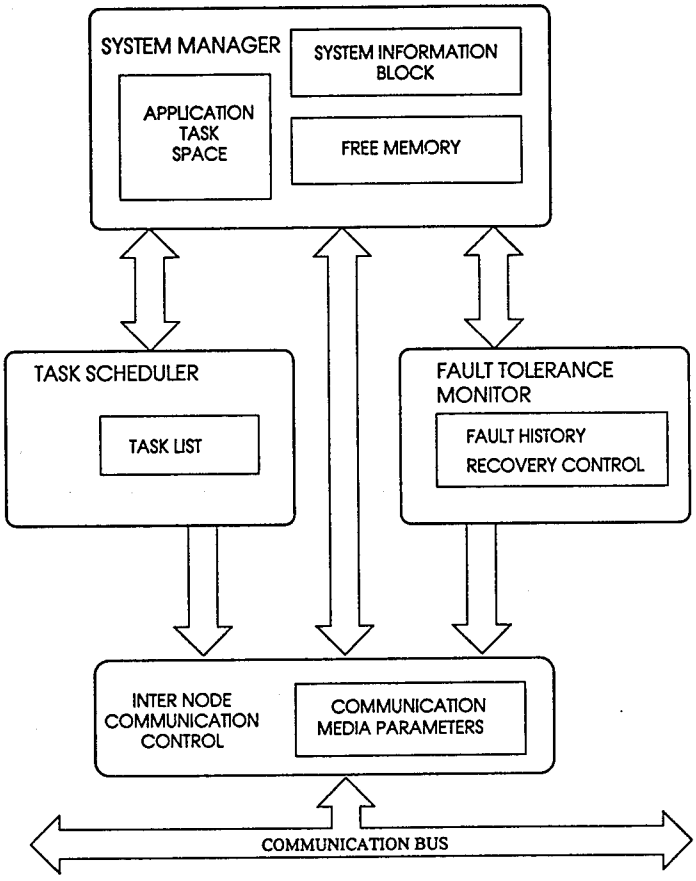


Figure 2: The ANTS kernel.

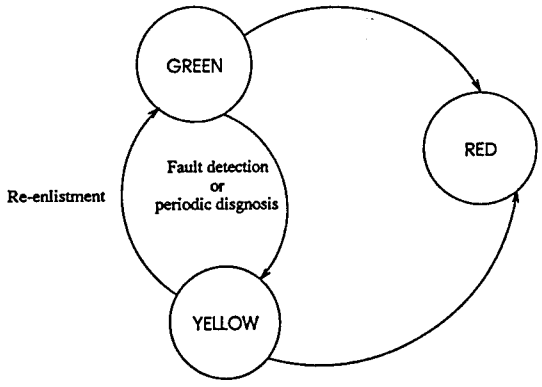


Figure 3: Run time partitioning of a fault tolerant ANTS system.

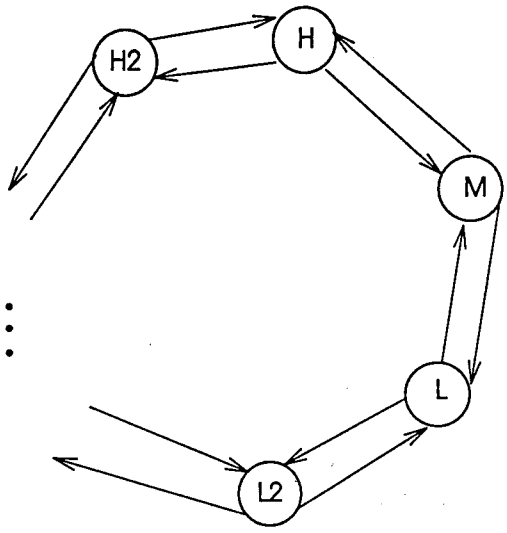


Figure 4: Monitoring for faulty ANT nodes.

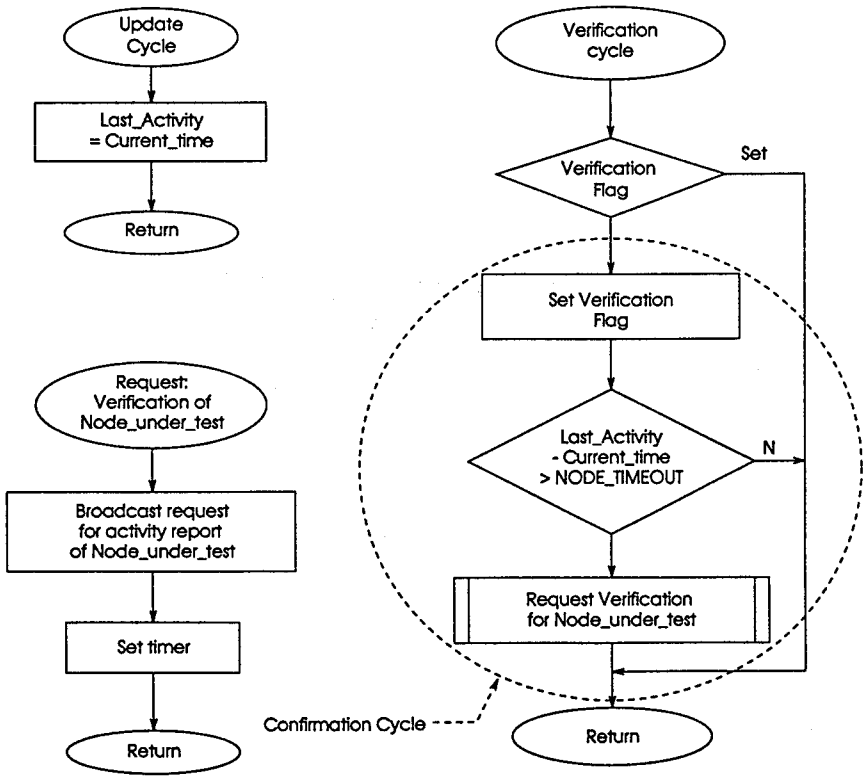


Figure 5: AFDRP: Update and Verify cycles.

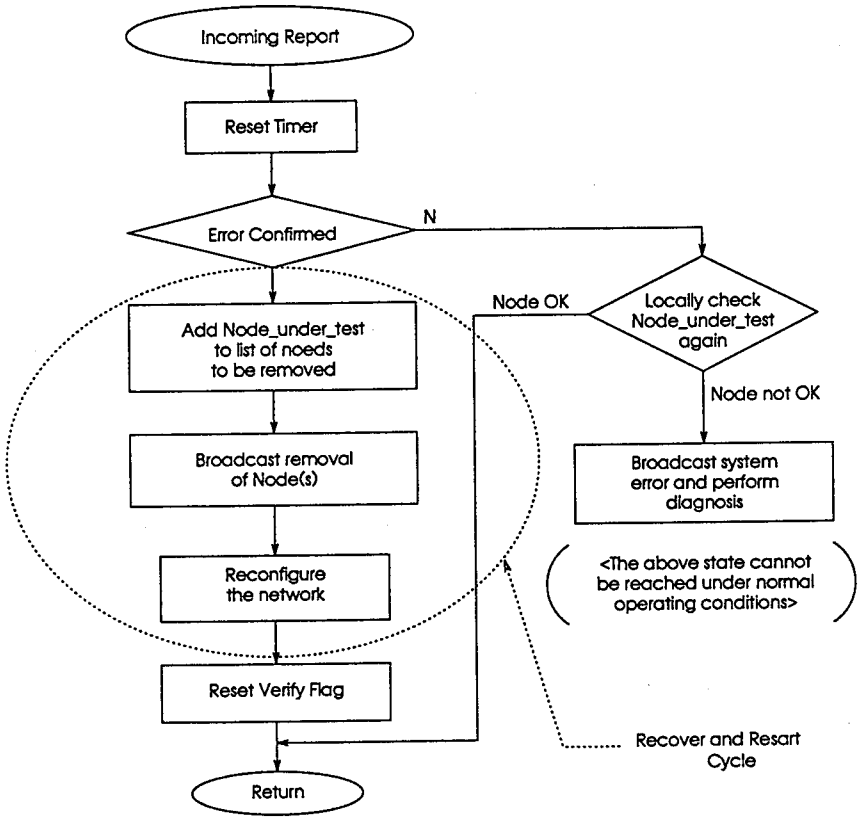


Figure 6: AFDRP: Recovery and Restart cycles.

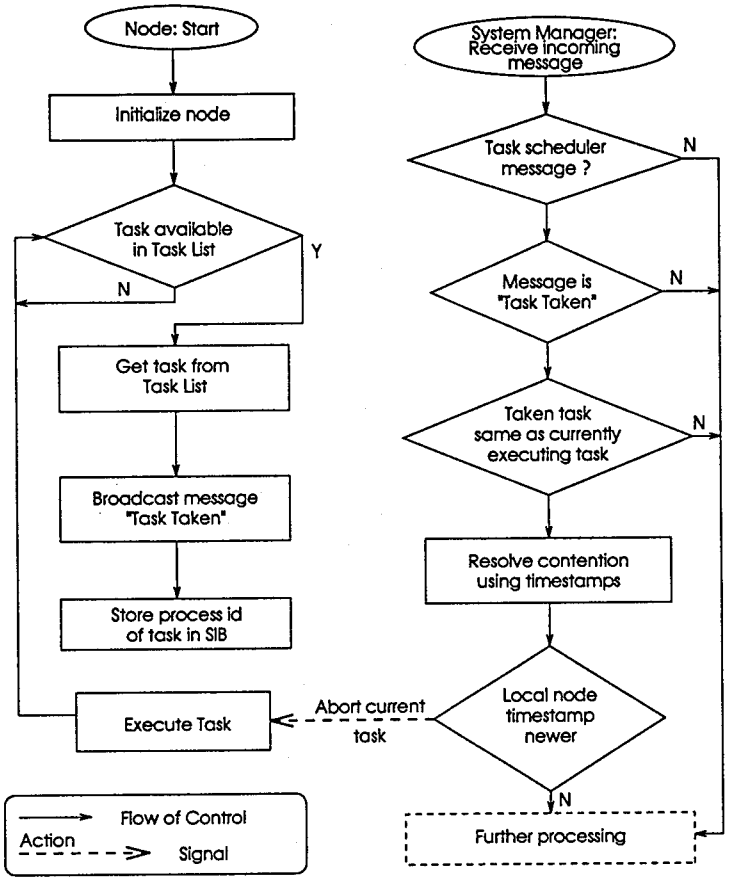


Figure 7: Contention Resolution in ANTS Task Scheduler.

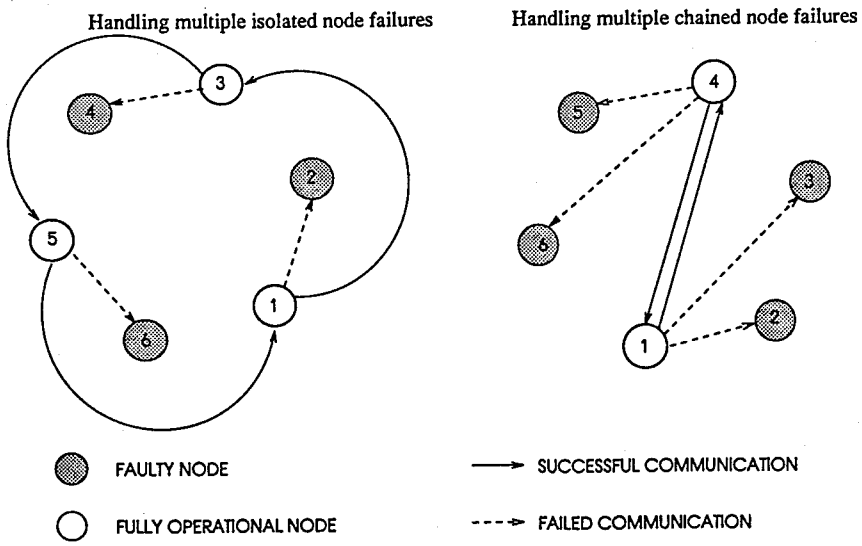


Figure 8: Detection of multiple isolated and chained ANT nodes failures.

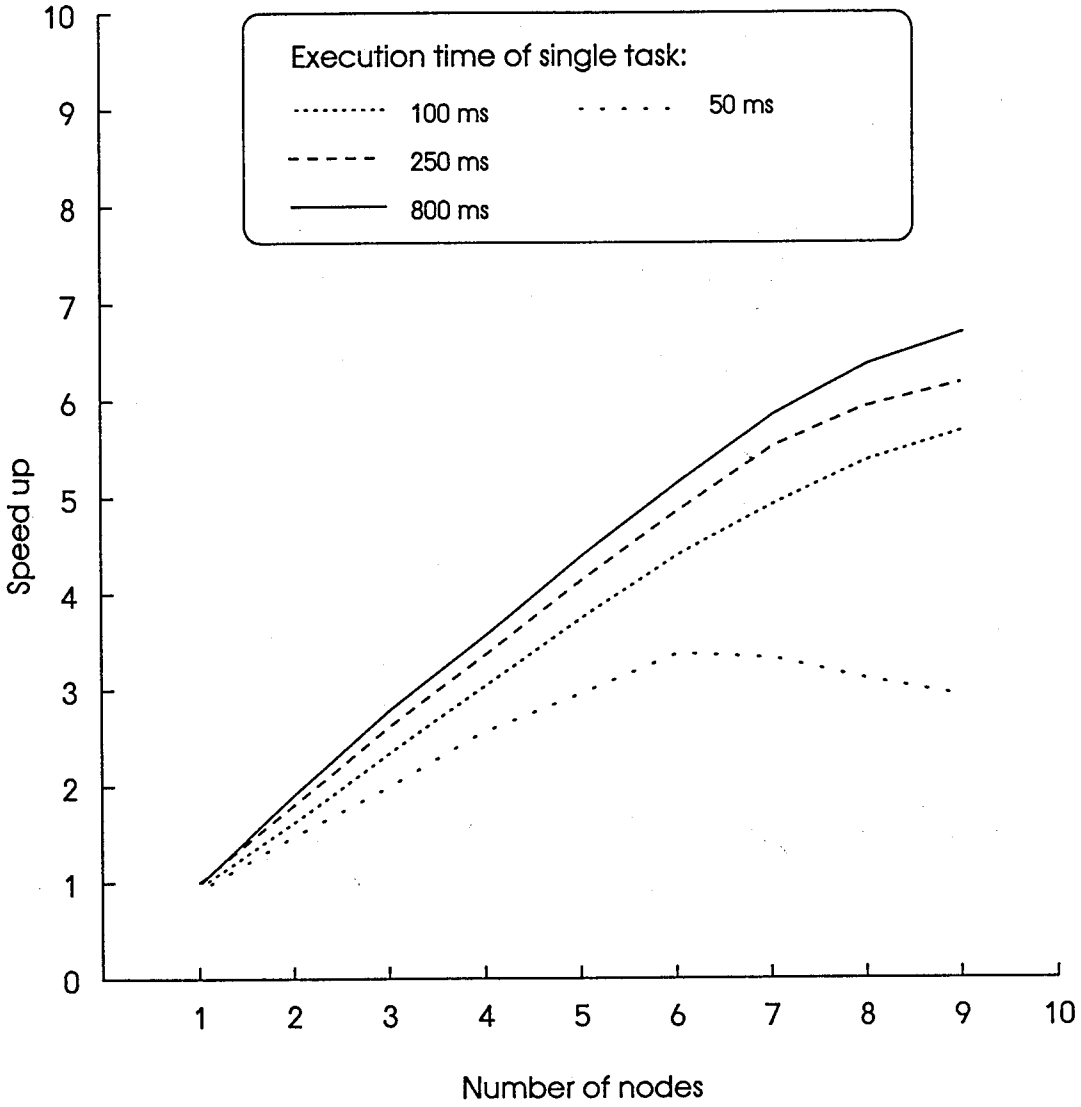


Figure 9: Speed up factors of ANTS with fault tolerant features disabled.

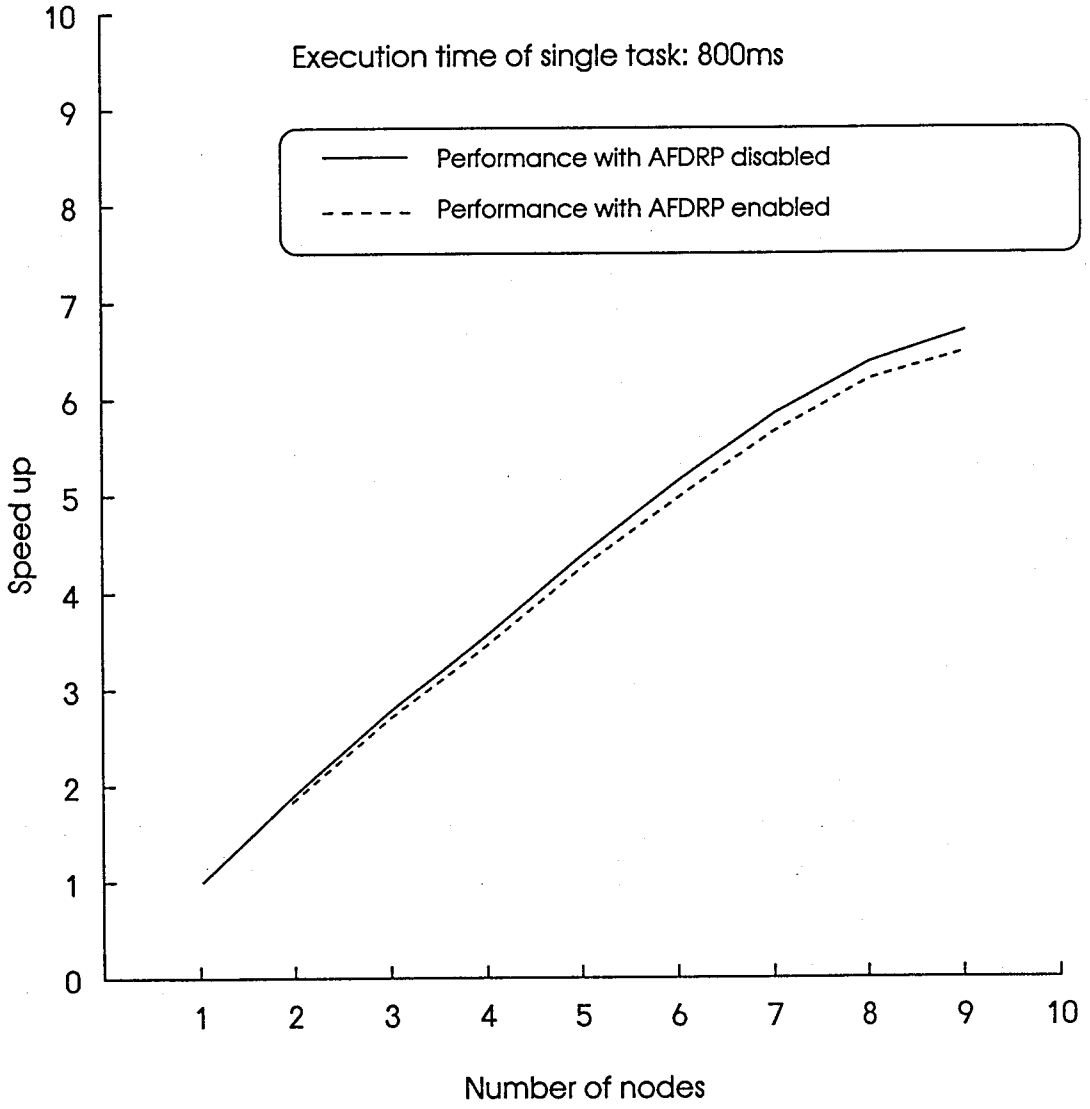


Figure 10: Trade-off in ANTS performance with fault tolerance features.

ANTS: An Approach for High-Performance and Ultra-Dependability in the Distributed Computing Environment

Patrick Dominic-Savio, Jien-Chung Lo and Donald W. Tufts
Department of Electrical and Computer Engineering
The University of Rhode Island
Kingston, RI 02881-0805

Abstract

We have previously proposed the Active Nodal Task Seeking (ANTS) concept as as a way to efficiently design and implement dependable, high-performance, distributed computing. The ANTS concept uses a completely decentralized approach in which each computing node is an active member of the system. This not only avoids the possible critical region in favor of a dependable computing, but also provides an adaptable, distributed resource management that explores the full potential in computational power of the system. This paper also provides experimental runs statistics showing the efficiency and effectiveness of the ANTS approach.

Keywords: distributed computing, MTTF, highly available systems, fault-tolerant distributed operating system, real-time systems.

This work is supported by the Office of Naval Research under contract N0014-94-1-0479.

1 Introduction

Distributed computing is potentially since there are built-in redundancies and that the processing units are distributed. However, we would like to point out that none of these existing systems, both theoretical and experimental, can efficiently achieve both high-performance and ultra-dependability at the same time [1, 2, 3, 4, 5, 6, 7]. There are numerous reasons, but, the one we believe is the main cause is that most known distributed systems are not really *distributed.* By this we mean that there is always an implied relationship between active processing units and passive processing units. This active/passive relationship gives rise to the load balancing problem [8]. Further, the dependability of the system is affected since the active units are also critical units.

Distributed systems are inherently fault-tolerant to failures of individual components. However, utilizing this intrinsic fault tolerance is not a straightforward task. Dependable distributed systems still need special hardware or software designs to achieve the desired level of dependability. The Advanced Architecture On-board Processor by Harris Corporation [1, 2] uses self-checking RISC processors and a chordal ring. The Advanced Automation System by IBM [3, 4] needs redundant components in each subsystem due to the nature of its wide area distribution. The on-board computer of the Japanese satellite Hiten [5] uses stepwise negotiating voting, which is a combination of mutual checking by data comparison and self-checking. In some cases, software redundancy is used. For instance, Delta-4 [6] uses active replication of software programs to ensure fault tolerance. A similar strategy is used in Manetho [7], where each application process is replicated by a set.

The ANTS project at the University of Rhode Island explores and evaluates the concept of Active Nodal Task Seeking (ANTS) as a way to efficiently design and implement dependable, high-performance, distributed computing [9]. By high-performance, we mean that the ANTS concept is designed to explore the full computational potential a system can provide. There is no passive unit in an ANTS system, and therefore a true *distributed* system. Each computing node has an equal role in contributing to the overall processing performance. There are several advantages that can be pointed out immediately. First, there is no need for the load balancing since the ANTS concept implies an always balanced work load for each computing node. Second, there is no critical region in an ANTS system since all computing nodes are active and that no passive unit is presented. Obviously, high-performance and dependability can be achieved at the same time with the ANTS concept.

This paper summaries the ANTS concept and experimental implementation results. The ANTS exhibits near linear speed-up curves in test runs. We also show that the impact of the inclusion of fault-tolerant features is minimum. The fault-tolerant features added to ANTS is a simple monitoring scheme similar to those in system level diagnosis. Depend on the applications, the ANTS provide the flexibility to execute duplicated or triplicated tasks when the situation should arise. All in all, sufficient evidences will be demonstrated that ANTS can indeed achieve both high-performance and dependability at the same time.

2 An Overview of ANTS Distributed System

In an ANTS system, an idle computing node will seek out the information it needs about pending, high priority work from partly prescribed, partly updated task tables, instead of waiting for other computing nodes to send it a job to do. Naturally, predecessor/successor conditions like those in a data flow architecture must be checked by the node. The advantage of this concept is that a true distributed system is guaranteed. All programs, including operating system, applications, and monitoring or diagnostic programs, can be executed distributively. A failed computing node will not influence the system performance as long as the *n+1* redundancy policy is enforced, *i.e.* the number of available computing nodes is always one greater than the number of required computing nodes.

2.1 General Attributes

There is no need for special hardware design to ensure the dependability of an ANTS system except perhaps in its internode communication network. However, some extra work by the programmer and/or an unconventional compiler is needed on a one-time basis, before the compiled program can be executed on ANTS. Some other attributes of ANTS are as follows [9]:

- On-line error checking is simple and straightforward. The two extreme failure modes: *fail-silent* and *fail-uncontrolled* [6] are easily detected since the failed computing node will not be actively seeking a new task.
- Fault tolerance can be attained without seriously degrading the system performance.

- There is no need for synchronization of computing nodes or programs. Since every computing node is an active unit, the ANTS system is an asynchronous system. The asynchronous nature of the operation also allows an easy implementation of recover from failure process.

A high-performance distributed computing system is not necessarily a good real-time system. Studies have shown that only 50% of the nodes, on average, in a distributed system are busy at any given time [10]. This is a serious drawback if the system is to be used in real-time applications. Clearly, the effort to guarantee a higher computing node utilization can also help to enhance the success of real-time executions. In ANTS, the needed programs will be first carefully partitioned and compiled such that each segment of sub-program (task) contains only continuous executable codes with a fixed limitation on execution time. This one-time effort is worthwhile since no run-time scheduling or task assignment will be needed later.

2.2 High-Performance in a Scalable Environment

As described earlier, we define the term high-performance as the capability to fully explore the potential computational power. This is a suitable definition for scalable environment where computing node can be added to gain higher performance. ANTS can adapt itself to accommodate the increase in the number of computing nodes without any hardware or software modification. For the fault-tolerant version, the ANTS can add or delete computing node during run-time without interfering with the current running jobs.

In an locally connected system such as several workstations interconnected via a local area network, this attribute of ANTS is very useful. Additional workstations can be added to obtain higher performance. Also, during run-time, some workstations that are heavily loaded with local jobs can stop its ANTS operation for the collaborated jobs. These workstations can easily re-enter the ANTS operation by again actively seeking out the next available tasks.

2.3 Fault-Tolerant Designs

The ANTS system has two levels of fault-tolerant designs: *kernel* level and *task level.* We may roughly divide the failure modes of computing nodes and communication bus in the following three categories:

1. control transfer errors,

2. communication errors, and

3. data manipulation errors.

Because there is no critical region in an ANTS system, the control transfer errors can be detected by monitoring. The monitoring technique implemented in the experimental system uses two computing nodes to monitor the outgoing communication activities of a node. The assignments of the monitoring and monitored nodes are dynamic. This monitoring technique also covers some of the communication errors, while the rest of the communication errors are handled by the communication protocol. These fault-tolerant techniques are implemented at the ANTS kernel level in the form of ANTS Fault detection and Recovery Protocol (AFDRP).

The data manipulation errors are handled at the task level. Currently, the experimental ANTS system support the option to execute duplicated tasks and triplicated tasks. Of course other special types of error detection/correction methods, such as algorithm-based fault tolerant (ABFT) for matrix operations [11], can be incorporated too.

3 The Experimental Implementation of ANTS

The first implementation of ANTS is carried out in a configuration as shown in Figure 1. The computing nodes in this case are Sun workstations, and the communication bus is an Ethernet. Of course, ANTS can be implemented on any type of parallel processing configurations. The choice of this particular system organization is due to the availability of hardware facilities.

The ANTS kernel is built on top of the SUN OS and the TCP/IP [12] communication protocol. A detailed view is shown in Figure 2. There are four major modules in the ANTS kernel: System Manager (ANTS-SM), Fault Tolerance Monitor (AFDRP), Task Scheduler, and Inter Node Communication Control.

3.1 AFDRP

At the ANTS kernel level, we implement a monitor scheme, ANTS Fault detection and Recovery Protocol (AFDRP), that is similar to that proposed in [13]. Each ANT node monitors its two neighboring nodes. Conversely, each ANT node is monitored by its two

neighboring nodes. Other checking functions such as periodic self-diagnosis, memory parity checks, cyclic redundancy checks on inter node messages and hardware integrity checking can also be selectively done depending on the available resources.

Once the concurrent error detection mechanism successfully detects a fault, it triggers the recovery and restart (R&R) cycle. All the ANT nodes modify their *System Information Block* (SIB) to reflect the removal of the node in question. More details about the SIB will be discussed later. The next step in the R&R step consists of *re-enlistment.* This involves bringing back the computing nodes which has been previously exhibited faulty behavior but has been check out complete in a self-diagnosis. Since intermittent and transient failures occur more frequently during run-time [14], especially in modern VLSI-based systems [15], there is no need to remove a component once a failure has been recorded. The record keeps at each ANT node will indicate the number and frequency of failure history. If an ANT node fails frequently, it will be removed from the system.

The monitoring scheme discussed above not only detect single isolated faulty nodes but multiple faulty nodes as well. The main advantage of the above scheme is that the monitoring of nodes and the fault detection process add no additional traffic over the communication network. The protocols function by monitoring other system communication messages and hence fault tolerance is achieved at the least communication expense. Detailed description of the fault-tolerant features can be found in [16, 17].

3.2 Task Scheduler

The ANTS Task Scheduler distributes the tasks waiting to be executed among all the nodes in the operational pool of computing nodes. The most important feature of the algorithm is the maintenance of a common coherent task list at all nodes.

3.2.1 Maintaining Coherency of Task Lists

Each node in the system has a data structure which reflects the state of all the tasks in the system and the list is assumed to be identical at all nodes. This data structure comprises of an ordered list of tasks that need to be executed. Tasks are added to this list whenever they are invoked and are deleted when they have been taken for execution. Maintaining the coherency of the task lists is done by passing messages among nodes. Three basic operations are used to establish the task list coherency.

1. *Append*: Whenever a task originates, an *APPEND* message is broadcast over the network. Along with the message, a data record containing all the information about the task that is being added is also broadcast. Each node that receives the message adds the data record to its respective task list.

2. *Delete*: Any time a node picks up a task for execution, it send out a *DELETE* message along with a data record identifying the task that is being taken up for execution. As before, every node on receiving this message deletes the task from its own task list.

3. *Add First*: This operation is similar to the APPEND, except that the task that is to be added is put on top of the task list instead of appending to the end. This is useful for inserting a high priority task and can be directed to a particular computing node.

3.2.2 Selection Policy

There are two selection policy implemented in the Task Scheduler. One type of the selection policy is for applications that need to be executed on a first come first serve basis. In this case, a *Simple Selection Policy* is used. The next task to be executed is the one that is on top of the task list. So, whenever a node is free it grabs the top most task from the task list if it is available for execution.

For applications that are data intensive, the execution of successive tasks with direct predecessor-successor relationship on different nodes require that huge volumes of data be transferred between the nodes that execute them. The communication cost involved could be reduced to a great extent in such cases by executing these tasks on the same node. The data that the tasks need are already stored locally and practically no inter-node communication is required.

When a node is free to select a task from the task list, instead of just selecting the task at the top of the list, a *look ahead* into the list is performed. Now, if another task which is a direct successor of the just completed task is found, it is chosen for execution instead of the one on top of the task list. This would result in considerable saving in terms of the data traffic between nodes. The depth of look ahead will be determined by factors such as the overhead involved, task queue size and other application specific issues. Note that a simple selection policy is obtained when the look ahead depth is one.

3.2.3 Contention Resolution

The distributed task scheduler at each computing node select the next task to be executed from its own copy of task list. Because of the communication delay, it is possible that several computing nodes may select the same task to be executed near simultaneously. If such scenario occurs, the task scheduler will receive message from other computing nodes stating the the task has been selected for execution at other nodes. As shown in Figure 3, the timestamps indicating the time the task is selected are compared. The computing node with a higher number in the timestamp will abandon the execution and will proceed to select another task from the task list.

This simple mechanism has been shown to be effective during experimental runs. It is also quite efficient since only very little overhead in comparing the timestamps is needed.

3.3 Inter Node Communication Control

The ANTS kernel Inter Node Communication facilities are built on top of the TCP Internet Protocols [12] which is available in the Sun OS. The purpose of adding this kernel on top of the available protocol is to customize the communication services available to best suit the needs of the ANTS system.

All the message handled by the INC conform to the predefined formats. Any message that does not follow the known patterns results in the node that sourced the message being put under test and diagnosis. The first byte of all messages indicates the type of message it carries. Figure 4 shows the bit patterns of the first byte and the message it identifies. Figures 5 and 6 show the message format for task scheduling and data transfer and for AFDRP, respectively.

A salient feature of the network interface is that it's network number of address is fully dynamic, in that the address can be modified by the ANTS Integrity Maintenance Protocol at run-time. The network number consists of an 16-bit integer number which leads to a maximum possible number of 65,536 nodes. A main reason for this arrangement of dynamic ANTS network number is so that no critical region exists by pointing to a fixed number as in a common local area network. The dynamic assignment of network number enables us to freely remove or insert a computing node without seriously interfere with the system performance.

3.4 ANTS System Manager

The ANTS system manager or ANTS-SM maintains a record called system information block (SIB). The SIB consists of three major categories of records as shown in Figure 7. These records are updated either due to interrupt-driven events, such as the system time, or due to messages oriented from other nodes, such as task completion record.

In ANTS there is no need for highly synchronized clock mechanisms. The ANTS nodes run in loose synchronization, the only need for a synchronized clock in ANTS arises when timestamps have to be generated. The ANTS Task scheduler is designed such that even for the timestamps a high degree of synchronization is not required. Based on these requirements the ANT-SM maintains a simple clock that counts in milliseconds. The clock is initialized during start up and is periodically checked at long intervals with neighboring nodes to trap possible erroneous values (this is done as part of the concurrent error detection mechanism at the kernel level). Other than this the precision of the system clocks is such that it does not need to be adjusted for periods far exceeding the system uptime.

3.5 Debugging and System Diagnosis Features

The ANT-SM also provides a run time window display of the functioning of the system. Some of the data that is displayed includes the first 10 tasks on the task list, the currently executing task, the status of INC and the list of active nodes in the system. The display basically provides continuous status information of the running system besides showing some cumulative data such as the total traffic over the communication network and number of tasks executed.

The other necessary system functions in the prototype are handled by the host operating system the ANTS kernel is running on. In future versions it would be advantageous to completely eliminate the host operating system interface to the hardware and implement the required features taking into account the requirements of the ANTS system.

4 Experimental Results

The ANTS prototype was tested on Sun Sparc workstations running Sun OS. The ANTS communication mechanism was layered on top of the Internet Protocols and the intercommunication network used was an Ethernet LAN running at 10 Mbits per second. The ANTS

code was written in C++ and compiled using the GNU C++ compiler. The test application program was emulated using timing loops and an inter task data transfer size of 1.5KB was assumed. Predecessor and successor relationships of tasks are either generated randomly or are assumed to resemble that of the FFT computation. In addition, some randomly generated tasks with randomly generated data dependency relationships are also used. Due to the limitation in the number of available nodes, the test runs were limited to a maximum of nine nodes. Tests were done for three different *Task Slice Time* factors and a comparison of performance with and without fault tolerance was also made to study the performance trade off implications of using the ANTS Integrity Maintenance Protocols.

4.1 Fault Tolerant Features Verifications

The concurrent error detecting mechanism and other integrity maintenance protocols were tested using simulated faults. The tests included cutting off the power supply to nodes or to parts of the hardware, intentional triggering of faults by transmission of invalid data over the bus and simulation of memory parity errors. Isolated and multiple faults were simulated to fully test the correctness of the monitoring scheme. The observed results conformed to the specified requirements in that each fault free node reaches an accurate diagnosis of the fault conditions of the remaining nodes, without any restriction being placed on the number of faulty nodes or fault patterns.

The fault tolerant mechanism always correctly reconfigured the network in all the test cases. Under extreme load conditions, it took the ANTS system (with 9 nodes) a maximum time of one minute to reconfigure itself to a stable state. Under nominal conditions of load and faults the mean reconfiguration time was about 10 seconds. The reconfiguration time is the time interval between the occurring of a fault and the time the network becomes fully operational again. The reconfiguration time is related to the number of faults occurring at the same time and to the communication network latency.

4.2 ANTS Performance without Fault Tolerant Features

Figure 8 shows the performance figures obtained from the test results. During these test runs, the AFDRP or the Fault Tolerance Monitor is disabled. The modulation in programming ensures that these results are obtained with no interference from the disabled modules. The following inferences can be drawn from the results:

First, as is to be expected the speed up factor is not linear. The speedup obtained gradually decreases with the addition of more nodes. The decrease in speed up is more pronounced with an increase in the number of nodes beyond 6. This effect is due to the traffic bottleneck imposed by the single communication channel. Addition of more communication channels or an increase in bandwidth of the communication media is necessary to achieve a better performance with more nodes in the system.

Another noticeable aspect of the performance figures is the higher linearity in speed up with higher Time Slice factors, *i.e.*, task execution time. This is evidently due to the fact that less overhead is involved with larger task fragments. The time slice factor will however have to be limited to a reasonable value to facilitate faster trapping of faulty nodes If the time slice factor were to be too large, a faulty node could unrecoverably damage the cooperative processing environment of the ANT system.

A third prominent feature is that with a large number of nodes in the system the speed up tapers of to an almost constant value. This value is a function of the total available bandwidth of the communication medium. This compares well with other distributed computing systems where beyond a certain point the speed up curve inverts and results in worsening performance [18]. This is due to the fact that with more nodes there are more messages that contend for the communication medium. In ANTS the communication traffic is not adversely affected by the number of nodes in the system.

We also observe that the average communication delay is about 200 to 250ms. There is a strong relationship between the average task execution time, the average communication delay, and the speed up factor. Of course, when a high speed network is available, a shorter task execution time can be used.

4.3 ANTS Performance with Built-In Fault Tolerant Features

We then record the performance figures of ANTS with AFDRP enabled. The test run results are shown in Figure 9. Figure 9 clearly shows that one of the major design goals has been achieved. The ANTS is designed to exploit the inherent fault tolerant features of the distributed system. We expect that a successful implementation of ANTS should have a minimum impact on the system performance. Test runs yielded a figure of 5% for the performance degradation due to the addition of fault tolerance. We believe that this low figure is a strong evidence to support that the inherent fault tolerance has been well utilized.

We have shown previously in [9] that an ANTS system can achieve an extremely long MTTF. Part of this MTTF improvement is due to the re-enlistment of ANT nodes with transient type failures. We note that the ratio of transient faults to all faults depends to a large extent on environmental conditions. In a documented case [5], the transient faults account for 20% to 33% in low earth orbit conditions. This clearly brings out the effectiveness of the recovery process of the ANTS fault tolerant mechanism. The penalty paid for obtaining such an improvement in MTTF is a minimal trade-off in performance, as shown in Fig 10. The loss of performance is a function of the number of nodes in the system and the mean performance degradation with 4 and 8 nodes was 2% and 5%, respectively. This is relatively small compared to the magnitude by which the recovery and re-enlistment process extends the useful life of the system.

5 Conclusions

We have presented in this paper a summary of the ANTS concept and some experimental results. We clearly show that the major design goal has been achieved in that the high-performance and dependability can be achieved at the same time. Of course, the fault tolerant features discussed here do not include the cost for detecting or correcting data manipulation errors. To that end, the cost of checking may be much higher than reported here, since task duplication, triplication, or alike must be used. However, we also should point out that no all application require such a restricted data manipulation error detection/correction. In some cases, reasonableness checking as suggested in [9] can be used. Our initial design goal has been aimed at the achieving of an extremely long MTTF [9]. The results presented here show that this goal can be achieved as predicted.

References

[1] M. J. Iacoponi and D. K. Vail, "The fault tolerance approach of the advanced architecture on-board processor," in *Proc. 19th Int'l Symp. Fault-Tolerant Comput.*, pp. 6–12, June 1989.

[2] M. J. Iacoponi and S. F. McDonald, "Distributed reconfiguration and recovery in the advanced architecture on-board processor," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 436–443, June 1991.

[3] F. Cristian, B. Dancey, and J. Dehn, "Fault-tolerant in the advanced automation system," in *Proc. 20th Int'l Symp. Fault-Tolerant Computing*, pp. 6–17, June 1990.

[4] T. R. Dilenno, D. A. Yaskin, and J. H. Barton, "Fault-tolerant testing in the advanced automation system," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 18–25, June 1991.

[5] T. Takano, T. Yamada, K. Shutoh, and N. Kanekawa, "Fault-tolerant experiments of the "Hiten" onboard space computer," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 26–33, June 1991.

[6] M. Chereque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron, "Active replication in Delta-4," in *Proc. 22nd Int'l. Symp. Fault-Tolerant Comput.*, pp. 28–37, July 1992.

[7] E. N. Elnozahy and W. Zwaenepoel, "Replicated distributed processes in Manetho," in *Proc. 22nd Int'l Symp. Fault-Tolerant Computing*, pp. 18–27, July 1992.

[8] M. Linvy and M. Melman, "Load balancing in homogeneous broadcasting distributed systems," in *Proc.* ACM *Comput. Network Perf. Symp.*, 1982.

[9] J. C. Lo, D. W. Tufts, and J. W. Cooley, "Active Nodal Task Seeking (ANTS): an approach to high-performance, ultra-dependable computing," *to appear in IEEE Trans. Aerospace and Electronic Syst.*, 1995.

[10] Z. M. Wojcik and B. E. Wojcik, "Rough grammar for efficient and fault-tolerant computing on a distributed system," *IEEE Trans. Software Eng.*, vol. 17, pp. 652–668, July 1991.

[11] K. H. Huang and J. A. Abraham, "Algorithm-based fault-tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, pp. 518–528, June 1984.

[12] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP*, vol. 1. Prentice-Hall, NJ, 1988.

[13] R. P. Bianchini and R. W. Buskens, "Implementation of on-line distributed system-level diagnosis theory," *IEEE Trans. Comput.*, vol. 41, pp. 616–626, May 1992.

[14] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM I. Res. Develop.*, pp. 20–27, January 1976.

[15] Y. Savaria, N. C. Rumin, J. F. Hayes, and V. K. Agarwal, "Soft-error filtering: A solution to the reliability problem of future VLSI digital circuits," *Proc. IEEE*, vol. 74, pp. 669–683, May 1984.

[16] P. Dominic-Savio, *Design of the ANTS Kernel to Implement a Fault Tolerant Dsitributed Computing System.* M.S. Thesis, University of Rhode Island, 1994.

[17] P. Dominic-Savio, J. C. Lo, and D. W. Tufts, "Fault tolerant features and experiments of ANTS distributed real-time system," in *submitted to FTCS-25*, 1995.

[18] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Computer*, pp. 57–69, November 1980.

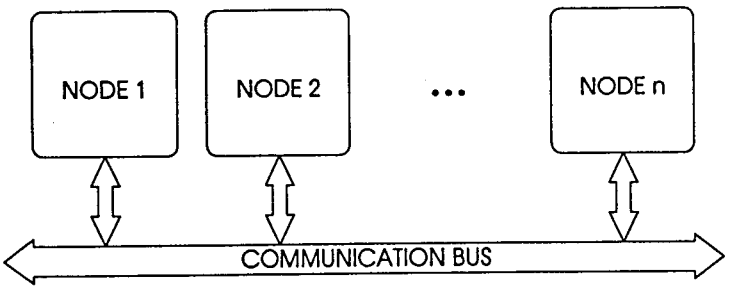


Figure 1: Architecture of the ANTS experimental system.

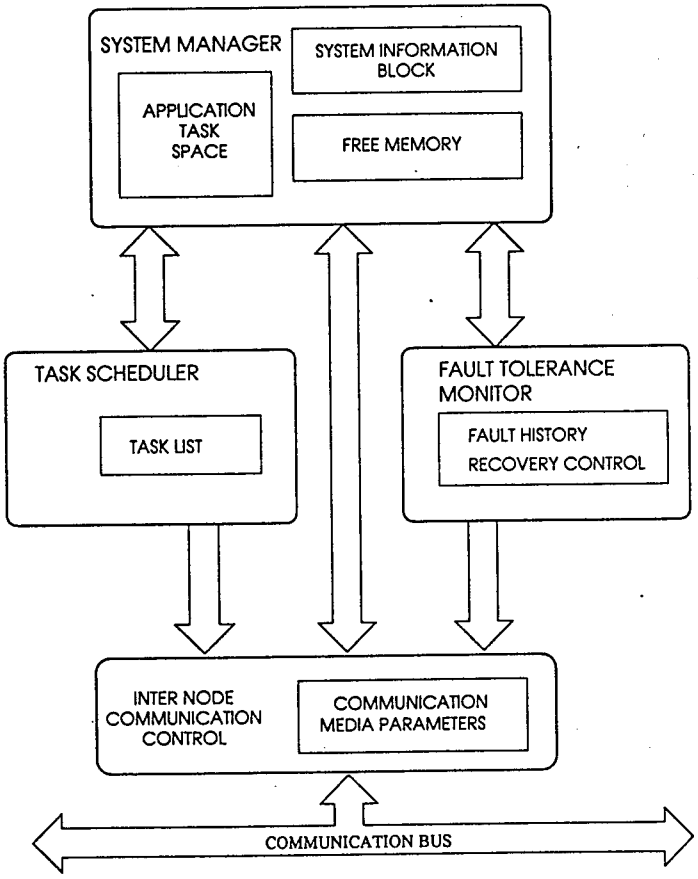


Figure 2: The ANTS kernel.

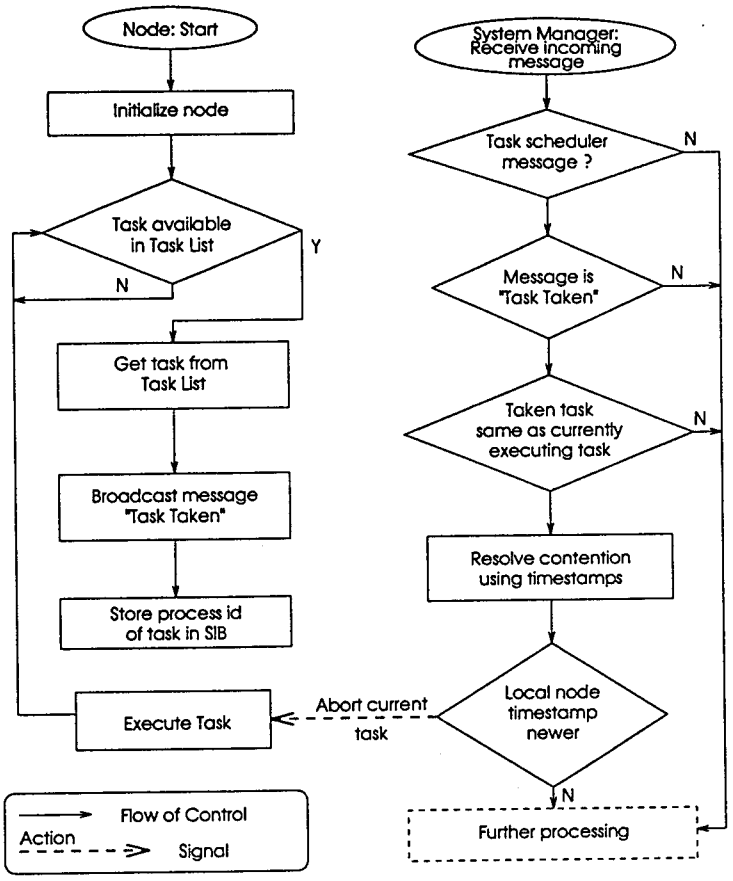
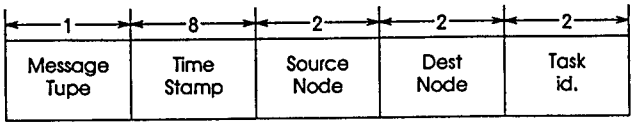


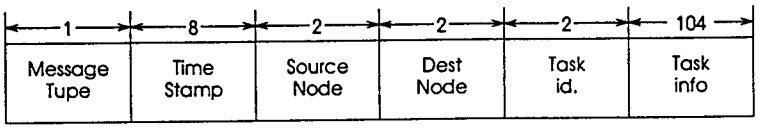
Figure 3: Contention Resolution in ANTS Task Scheduler.

Task Scheduler	New Task	0 0 0 0 1 0 1 0
	Delete Task	0 0 0 1 0 1 0 0
System Manager	Request for Data	0 0 0 1 1 1 1 0
	Data	0 0 1 0 0 1 0 1
Fault Tolerance Monitor	Request for monitor report	0 0 1 0 1 0 0 0
	Monitor Report	0 0 1 1 0 0 1 0
	Remove Nodes	0 0 1 1 1 1 0 0
		MSb LSb

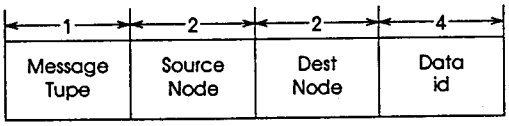
Figure 4: Pre-defined bit patterns for the first byte of an ANTS message.



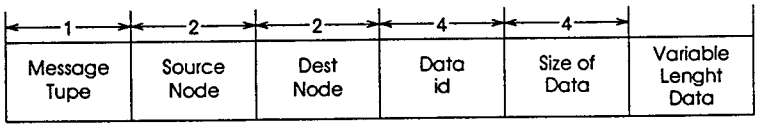
a. Format of <Delete from list> message



b. <Add to list> message.



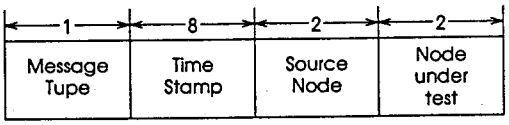
c. Reuest for data.



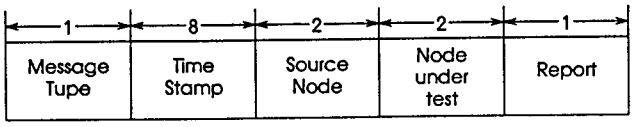
d. Reply: requested data.

Data width in bytes

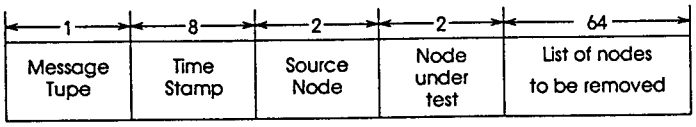
Figure 5: Message formats for Task Scheduler and data transfer.



a. Request for a report on the Node under test.



b. Reply: report of the Node under test.



c. Removal of nodes message.

Data width in bytes

Figure 6: Message formats for AFDRP.

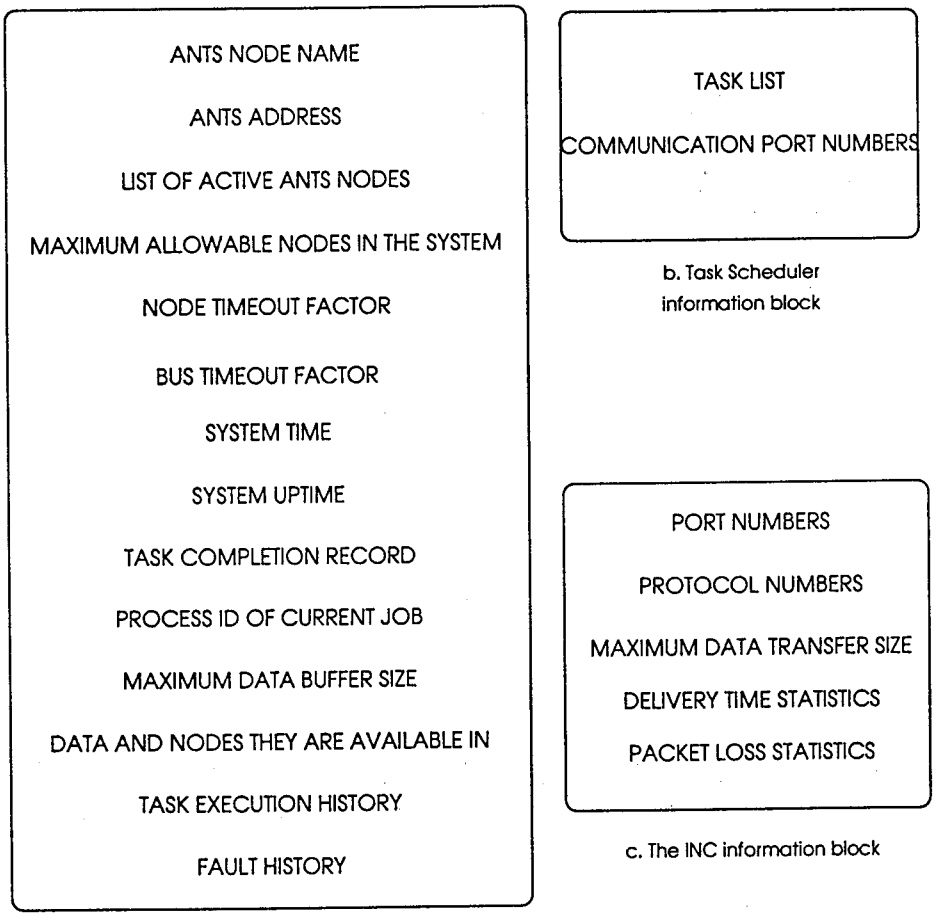


Figure 7: The System Information Block.

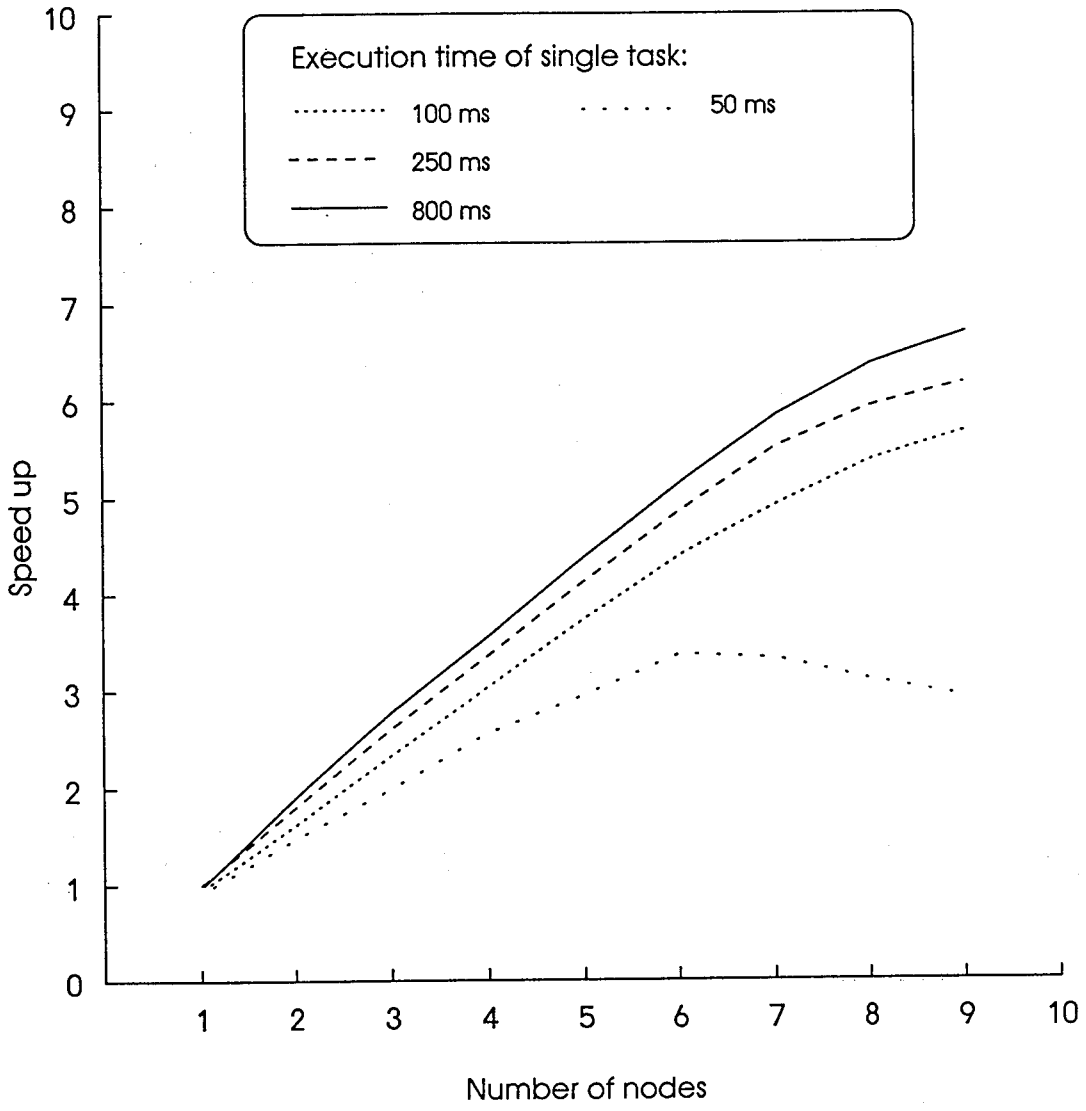


Figure 8: Speed up factors of ANTS with fault tolerant features disabled.

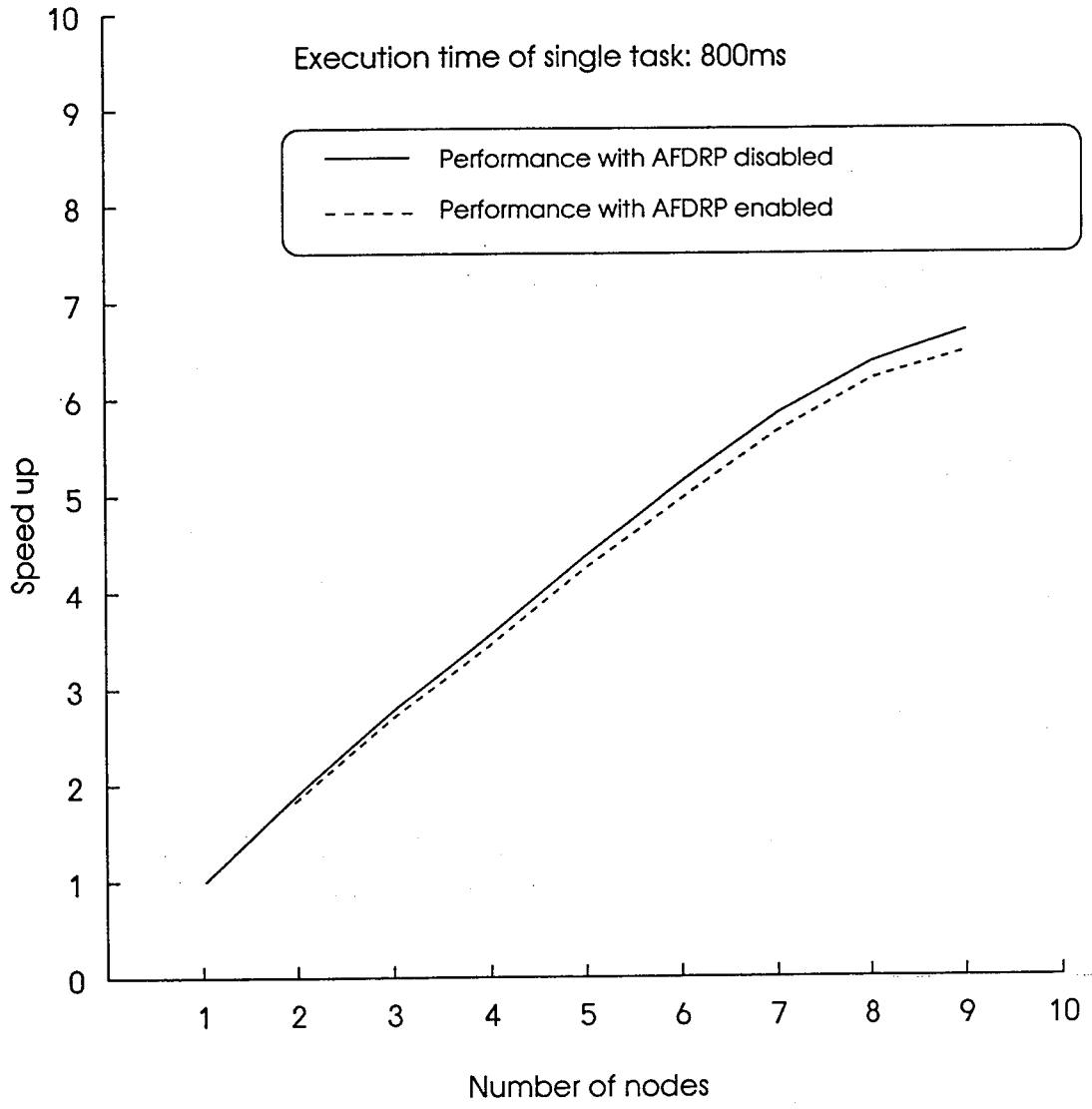


Figure 9: Trade-off in ANTS performance with fault tolerance features.