Systems and Computers in Japan, Vol. 31, No. 7, 2000
Translated from Denshi Joho Tsushin Gakkai Ronbunshi, Vol. J81-D-I, No. 11, November 1998, pp. 1219–1227

Analysis for Performance and Reliability of Fault-Tolerant Parallel Software

Eiji Sugino* and Haruo Yokota**

Japan Advanced Institute of Science and Technology, Ishikawa, Japan 923-1292

SUMMARY

We propose a technique for constructing a fault-tolerant parallel software for general commercial massively parallel computers which are not provided with special fault-tolerant functions. This technique is a hybrid of the primary/backup approach and state machine approach, and can implement parallel programs in fault tolerance by automatically converting user programs. In general, when a parallel system is to be used as a fault-tolerant computer, since parallel entities are used as redundant elements for obtaining fault tolerance, the maximum performance will decrease concurrently with the improvement of reliability. Moreover, it is necessary to consider the performance drop for processing which is supplementary to the original program in fault-tolerant implementation by software. Therefore, a gain by fault-tolerant implementation cannot be shown if it is merely demonstrated that an improvement of the reliability is obtained. In this paper, we define an evaluation index which takes into account reliability improvement and performance drop; based on this index, we study the execution environment which can tolerate practical use for fault-tolerant parallel software. © 2000 Scripta Technica Syst Comp Jpn, 31(7): 56–65, 2000

Key words: Fault-tolerant software; parallel logical type language.

*Currently at Iwate Prefectural University.
**Currently at Tokyo Institute of Technology.

1. Introduction

The MTTF of a parallel computer is $1/N$, in general, for a number N of constituent processing elements (PEs). For example, a parallel computer consisting of 1000 PEs having an MTTF of 10,000 hours has a potential capacity of 1000 times that of one PE, but the MTTF is only 10 hours. For a parallel computer which has been developed to obtain high performance, the goal can be achieved by increasing the number of PEs in this way, but its fault tolerance has been reduced. For a parallel computer whose scale is small or for parallel software whose execution time is small, this has been sufficient. However, as commercial massively parallel computers appear, parallel software requiring prolonged continuous execution has also appeared recently and fault tolerance is also being demanded in parallel computers.

In fault-tolerant computers, including Tandem, fault tolerance is realized by utilizing the parallelism of special hardware. However, fault-tolerant implementation by special hardware and a special OS is expensive; to develop an inexpensive method, there was an investigation of the realization of fault tolerance by software for existing parallel computers [1]. Moreover, in special systems, support for fault tolerance-based programming has been developed by forcing a new software development approach oriented toward that system on the users. However, as usual, the programmer must define the processing for the fault, which constitutes a large burden. In parallel software, which is much more complicated in structure, such burdens will become much larger.

As a fault-tolerant implementation hypothesizing a general commercial massively parallel computer, we have proposed a method for constructing fault-tolerant parallel

© 2000 Scripta Technica

software (FTPS) [4, 5] by automatic conversion of parallel logic programs [2, 3]. This can be regarded as a hybrid of the primary/backup approach and the state machine approach [1, 8]. We have so far performed the measurements and analyses by packaging a small-scale experimental program on an nCUBE2 [6, 7].

Research on the fault-tolerant implementation of parallel programs has included a study by Cherif and colleagues on fault-tolerant software with functional programming as a target [10]. In the model they used, the execution unit, called a module, is a function having no side effects. The module is decomposed into submodules and developed into a tree structure. Our research differs in that a program based on a process model is the target. The parallel logic programming we used can also be treated as the development of the predicate into a tree structure. However, its execution sequence is nondeterministic and is guaranteed by a synchronous function in which queuing based on dependence relations is provided by the language processing system. Moreover, in their scheme, the execution results of the submodules are stored in nonvolatile memory as soon as the majority decision is taken among replicas. Thus, when a separate replica tries to execute the same submodule later on, it suffices to utilize the stored results and parallel effects among replicas can be expected.

Kishimoto and colleagues have studied a UNIX-based fault-tolerant implementation without hypothesizing a special machine [9]. Their approach is also a fault-tolerant implementation based on process replicas; however, since they cannot guarantee identity of the processes among copies and are concerned about the timing dependence bugs, they have adopted a scheme in which the state is preserved in nonvolatile memory. In contrast, in our research, in addition to the fact that we have proposed a scheme which will guarantee meaningful identity, timing dependence bugs are prevented by distinctive features of the parallel logic programming language.

In general, FTPS based on replicas requires communications to maintain consistency among replicas, and the execution time of the FTPS will decrease due to the extra processing involved in these communications. The method proposed by us will also greatly affect the communication costs among replicas and its permissible amount can be estimated from the communication performance of the system.

In this paper, we will expand the analyses obtained through our research and study the conditions for obtaining sufficient execution performance by FTPS based on replicas in general. The process replicas will reduce the performance as a parallel computer due to a decrease in the usable number of PEs. Hence, superiority cannot be ascertained by merely comparing execution times. In this paper, we define the MWTF (Mean Workload To Failure) as an evaluation criterion defined from reliability and performance; based on this criterion, we will determine the conditions under which fault-tolerant execution becomes superior.

This paper is composed as follows. In Section 2 we will set the assumptions for the FTPS based on process replicas. We will also provide assumptions for the properties of the parallel program under consideration for fault-tolerant implementation. Next we will describe the evaluation index MWTF of the FTPS in Section 3. This becomes the evaluation criterion of the system for reliability and performance. Then, in Section 4, we will determine the conditions required in the parallel programs. In Section 5, based on these conditions, we will determine the conditions for maintaining the MWTF of the FTPS according to the evaluation criterion of Section 3.

2. FTPS Based on Process Replicas

2.1. Fault model

The parallel program under consideration is composed of a set of processes which exchange messages. Each process is mapped on the real PE and performs paralleling and parallel operation. The process is either in a normal state or in a crash state. The crash state is a state in which no transmission and reception of messages is performed as observed from outside owing to a fault of the PE which is its operation environment or of the communication channel between PEs.

2.2. Communication model

Communication between processes follows an asynchronous model. Between processes mapped on different PEs, processing in a PE will be interrupted for communication processing between PEs. For simplicity, the transmission and reception processing times in each communication are considered the same; the communication overhead is estimated by treating the number of transmissions and receptions as the amount of communication. In general, there is a difference due to the respective message lengths and there is also a difference in the processing costs of transmission and reception; however, we take this approach based on experimental experience on nCUBE2. In nCUBE2, the number of communications is predominant over the message length in the communication time [7]; moreover, in a high-level language such as the one we have used in packaging, the ratio of the message encoding and decoding times in the processing system in the communication time is large. Furthermore, if the cost in the interchange of context for transmission and reception is considered, this may be considered a realistic assumption. Besides, since the communication between processors on

the same PE takes place via the memory, the processing time is so much smaller that it can be neglected compared to that between PEs.

In the following, we assume that the amount of communication of the program is determined by the amount of work allocated to the program. This is because the amounts of transmission and reception required in the execution of the program depend on the amount of data to be processed by it, based on the assumption that the amount of data to be processed and the amount of work are proportional. In a parallel program, the amount of communication will in general change depending on the method of division of work. In the next section, we will implement the assumptions for the amount of communication of the parallel program.

2.3. Execution model of parallel program

When the program is to be executed in parallel, if the number of PEs constituting the system is N, the amount of work at maximum load in the PE becomes, by Amdahl's law, $(p/N + (1 - p))w$ $(0 \le p \le 1)$, where p is the parallelizable fraction of the amount of work w. However, the communication processing between PEs is not considered in this value.

If the work w required for communication in the total amount of work is homogeneous and distributed equally over all PEs of the N-unit configuration, the following three cases may be considered for the relation between the distribution and the amount of communication of each PE.

(1) The amount of communication of the PE increases due to division.

(2) The amount of communication of the PE decreases due to division.

(3) The amount of communication of the PE does not change because of division.

As an example of (1), the case may be considered in which the communication of all links is required by the subtasks themselves. In this case, the amount of work in each PE is $(p/N + (1 - p))w$ for one PE and is pw/N for the other $N - 1$ PEs. Since it is required that the communication is homogeneous in this work, the amount of communication of each PE depends on whether the amount of work itself or the remaining total amount of work distributed to the other PEs is larger. Namely, in a PE with an amount of work $(p/N + (1 - p))w$, it depends on the total amount of work $(p - 1)pw/N$ external to that PE and the quantity $\max((p/N + (1 - p))w, (N - 1)pw/N)$; in a PE with an amount of work pw/N, it depends on the total amount of work $(p/N + (1 - p))w + (N - 2)pw/N = (N - p)w/N$ external to that PE and the quantity $\max(pw/N, (N - p)w/N)$. Therefore, the total amount of communication flowing in the entire system is proportional to $w_c = \max((p/N + (1 - p))w, (N - 1)pw/N) + (N - 1) \times \max(pw/N, (N - p)w/N)$. If N is sufficiently large and the degree of parallelization is also sufficiently high ($p \approx 1$), we have $w_c \approx N_w$ and the total amount of communication will increase in accordance with the system scale N. In this case (1), since the amount of communication will increase as the assigned work w is further distributed, this is a program type not basically oriented toward parallel execution except in the case where the communication processing capability is very high or the system scale is small. In other words, since the coupling force among tasks is strong, it is easier to obtain performance in this type of program by executing it with one PE.

As an example of (2), the case may be considered in which the subtasks communicate only with the neighboring tasks. The amount of communication of each PE is also proportional to pw/N. In this case, the amount of communication in the entire system remains unchanged.

In case (3), no communication exists between subtasks or else (1) and (2) are mixed and cancel each other. In the case of mutual cancellation, the portion representing case (1) is merely executed sequentially and the processing efficiency will rise, resulting in case (2). Moreover, as an example of the amount of communication being unchanged, there is an OR parallel program with limitation which requires almost no communication between subtasks. This type is typical in parallel programming and is highly parallel-processing oriented. As in the split-governing method, since there is no interconnection between subprograms, no communication is required during parallel operation; however, communication is required in the gathering of the final results. Since this amount of communication depends on the amount of divided subprocessing, it will still result in case (2).

Here, type (2) is hypothesized as a parallel program type. This is because this type of program is most likely a parallel program and may be considered as a parallel processing-oriented type. Therefore, the amount of communication c contained homogeneously in the work will be expressed in terms of the communication content γ as $c = pw\gamma/N$. We may also assume that communication processing occurs only in the parallel execution portion (pw/N) and that the sequential execution portion $((1 - p)w)$ performs the communication inside PEs with coexisting parallel portions.

2.4. FTPS execution model

As an FTPS based on process replication, we will define the execution model for the FTPS technique that we have proposed. This technique is a hybrid of a typical state

machine and the primary/backup approach, and these techniques are also applicable in the analysis presented later.

A faulty PE will not be utilized again in the following. This is because it is difficult from the software viewpoint to restart the faulty PE and to perform reloading of the processing system and OS.

The replication processes are implemented in groups, and mapped into separate PEs in such a way that the groups will not overlap. Although selection in which the groups are mixed together and mapped into the PEs is also possible, it will not be considered here. This is because a process which is affected by one PE fault will spread to multiple groups and no improvement of reliability will be possible. Moreover, it is assumed that the number of PEs in each group has no imbalance. Therefore, for the total number of PEs N, if the number of replicas is $k + 1$, each group will operate in parallel on $n = N/(k + 1)$ PEs. Here, k expresses the number of faults which can be tolerated (the resiliency) and this system is called a k-resilient system. It is easily understood that performance as a parallel system will drop if k is increased in order to increase reliability, or if the number of PEs n which can be utilized for parallel execution of the program is decreased.

If the program is FTPS-implemented, it is unavoidable that the amount of processing for fault tolerance with respect to the original program will increase. With respect to the amount of work w of the original program, the amount of work of the FTPS is expressed as $w' = (1 + \alpha)w$, using the rate of increase $\alpha(> 0)$ of the amount of work for the FTPS. Therefore, the amount of work of the FTPS excluding the communication per PE is a maximum of $(p/n + (1 - p))w'$. The performance of the FTPS is also influenced by the amount of communication for the FTPS.

2.4.1. State machine

Each process of the program is replicated and run on a separate PE. The inputs from other processes are sent to all replicas and the outputs are transmitted from all replicas. Although communication between replicas is not required, the execution of the process must be deterministic. When any replica crashes, it is merely abandoned and other replicas continue to execute.

In the case of a state machine, there is no communication between replicas, that is, communication for FTPS. The increase of the amount of work for FTPS is about equal to the increase in operation required for generating the replicas and for input/output operations to the replicas. If the actual amount of work at the replicas is sufficiently large and the input/output is sufficiently small, the increase of the amount of work can be neglected. Therefore, this constitutes the most efficient approach for the FTPS. However, in general, nondeterminism is allowed in the parallel program; and in this technique where the determinism is assumed, all-sequence multicasting is required in order to keep the state of the nondeterministic process set identical between replicas, which is very difficult.

2.4.2. Primary/backup

Let one group of replicas be primary and the others be backups. The inputs from other processes are sent to the primary only and the outputs are also sent from the primary only. The primary transfers the change of the internal state to the backups and each backup changes its internal state accordingly. After the change, the backup sends an acceptance signal back to the primary. Waiting for the confirmation of the acceptance from all backups which are in normal operation, the primary moves to the next operation. When the backup crashes, it is merely abandoned; however, when the primary crashes, a new primary is selected from among the backups, or resending of the input to the primary from outside the system may be necessary.

In packaging, all-sequence multicasting as in the state machine is not required; however, atomism is required in the multicasting from primary to backup. When atomism cannot be guaranteed, consistency of the internal state between backups cannot be guaranteed.

In this technique, to avoid overlap or extinction of the outputs to the outside, an internal state in which consistency is assured must be obtained by coordination between processes (coordinated checkpointing). Otherwise, it may be necessary to roll back to the state in which consistency including the outside has been achieved during recovery from a fault. These amounts of processing are hard to estimate; however, the entire performance may be greatly affected by the amount of communication between primary and backup. Since these amounts of communication c' depend on the amounts of work in the PEs and the interval in which the internal state is taken during the execution of the work, they can be expressed by using an appropriate coefficient β, as follows:

$$c' = \beta \left(\frac{p}{n} + (1 - p) \right) w' \quad (0 < \beta) \tag{1}$$

2.4.3. Hybrid approach

Also similar to the primary/backup approach, one group of replicas is primary and the other consists of backups in this technique. However, similar to the state machine, the inputs from the other processes are sent to all replicas and the outputs are also sent from all replicas. Each replicated group can be made to correspond to the PEs of arbitrary units and parallel operation inside the group is made possible. The user process to be executed with fault tolerance is first replicated and sent to the respective groups. Moreover, this process can form a process network by forking the subprocesses in the PEs constituting the group.

If the subprocess operates deterministically, it will operate without communication between replicas, as an active replica in the state machine approach. On the other hand, if the subprocess operates nondeterministically, the process in the primary will operate as a primary replica and will be sent to the backup replica corresponding to the information of the nondeterministic operation result. Based on this information, the backup replica will reproduce the same operation as the primary replica.

The nondeterministic operation mentioned here means free operation allowed in the process; its behavior cannot be controlled from outside and can only be observed from the execution result. In general, such nondeterminism is allowed in parallel languages for the purpose of reviving the parallelism of the program. In OCCAM, for example, operation with nondeterministic input from multiple channels can be described by the ALT structure. In the parallel logic language, since the extraction of the nondeterministic operating portion of the program source is possible mechanically, automatic conversion is possible.

Furthermore, in this scheme, the state between replicas is maintained for every process, not for every PE. If the state is to be maintained for every PE, the execution scheduling inside the PE must be maintained the same and synchronization between PEs becomes necessary for every scheduling. Execution is therefore reproduced in the proposed scheme by maintaining the same state meaningfully. Therefore, when the primary is faulty, there is no guarantee that the backup is strictly in the same state; however, if it is normal, it can be guaranteed that the same state will always be reached meaningfully. For this reason, the backup will not roll back. Besides, for the commitment of output to the outside, lenient synchronization will be performed in nondeterministic execution and output operations.

Compared to the primary/backup approach, since the realizability of this approach as a parallel program is high, and since it is not necessary to consider coordinated checkpointing and rollbacks, the performance estimation is also relatively easy and is determined by the amount of communication c' described in primary/backup. However, β is determined by the nondeterminism existing inside the program.

3. MWTF, The Evaluation Index

The reliability of a nonrestructuring system can be evaluated in terms of the MTTF; however, in order to prolong the MTTF of the parallel system, it is simpler to reduce the number of PEs. The purpose of this research is to increase the number of PEs which can be utilized along with the MTTF, and the MTTF thus is not appropriate as an evaluation index. Accordingly, as an evaluation index for comparison here, we will introduce the mean executable workload or MWTF (Mean Workload To Failure). The MWTF is defined by the processing capability per unit time of the system, times the MTTF, and expresses the work capacity the system is expected to complete before a fault.

Fault-tolerant implementation of the parallel program will increase the MWTF; however, in general, processing overhead will be added and the amount of processing will also increase more than in the original program. If this increase in the amount of processing is considered, the processing capability per unit time of a system which is implemented as fault-tolerant may be regarded as decreased. This processing capability per unit time can be obtained from the actual execution time of the program. Namely, if the amount of work of the original program, the processing capability per unit time of the system, and the execution time of the program are w, W, and T, respectively, and if the processing capability per unit time of the system and execution time of the fault-tolerant program are W' and T', respectively, then $W' = (T/T')W$ can be obtained from the relations $T = w/W$ and $T' = w/W'$.

In the parallel program, the overhead for parallelization and communication processing will be added to the amount of work, and the idle time and communication waiting time due to the imbalance of parallel operation will be added to the execution time. In the following, the imbalance of parallel operation and the idle time will not be considered because of the homogeneous distribution. Therefore, the processing capability per unit time is calculated with reference to the total CPU power, the parallelism of the program, the amount of communication processing, and the communication processing performance per unit time. If the processing capability per unit time is simply considered as the total CPU power, it can be regarded, in the parallel system, as N, based on N PEs; however, as described before, since other factors will be added, this can be regarded as the maximum processing capability. The MWTF calculated from this can be regarded as the maximum MWTF.

3.1. Parallel system

The reliability of a general parallel system is calculated for a series system because execution will fail even if one of the component elements is at fault. Namely, for the reliability $r(t)$ of the PE, we have $R(N, t) = r(t)$ in general. It is assumed that the failure rate of the PE is constant and that the reliability follows the exponential distribution. Namely, if $r(t) = e^{-t}$, the MTTF of the PE is

$$MTTF_{PE} = \int_0^\infty r(t)dt = \frac{1}{\lambda} \tag{2}$$

and the MTTF of a parallel system of N-unit configuration becomes

$$MTTF_{MPP}(N) = \int_0^\infty R(N,t)dt = \frac{1}{N\lambda} = \frac{1}{N} MTTF_{PE} \quad (3)$$

Namely, since the MTTF will be $1/N$ as great as that of a single PE if the parallel system is an N-unit configuration, the maximum MWTF is equal to the MWTF of a single PE. Namely, the MWTF of the parallel system can be reduced by the MWTFs of the component PEs.

3.2. FTPS execution system

A fault-tolerant implementation based on process replication will increase the MTTF of the system at the cost of a decrease in the utilizable number of PEs of the parallel system. The FTPS execution model dealt with in this paper can be regarded as a nonrestructuring system with the respective replicated groups as the component elements. In the k-resilient system (the parallel system consisting of $k+1$ replicated groups), it can be calculated as follows (here F is the unreliability corresponding to the reliability R):

$$\begin{aligned} R_{FTS}(N,k,t) &= 1 - F_{FTS}(N,k,t) \\ &= 1 - (1 - R(N/(k+1),t))^{k+1} \\ &= e^{-\frac{N}{k+1}\lambda t} \sum_{i=0}^{k} \left(1 - e^{-\frac{N}{k+1}\lambda t}\right)^i \end{aligned} \quad (4)$$

$$\begin{aligned} &MTTF_{FTS}(N,k) \\ &= \int_0^\infty R_{FTS}(N,k,t)dt \\ &= \frac{k+1}{N}\left(\sum_{i=0}^{k} \frac{1}{i+1}\right) MTTF_{PE} \end{aligned} \quad (5)$$

Therefore, the MTTF of the k-resilient system with a total of N PEs is $(k+1)\sum_{i=0}^{k} 1/(i+1)$ times the MTTF of the original parallel system. Moreover, in general, the MTTF of the FTS is of the order of $k \log k$, from the following relation:

$$k > 0,\ \log(k+2) < \sum_{i=0}^{k} \frac{1}{i+1} < 1 + \log(k+1) \quad (6)$$

4. Conditions of Parallel Program

Since the purpose of the parallel program is high-speed execution, it is meaningless unless it is faster than sequential execution. In order to determine the conditions for high-speed implementation of this parallel program, we will study the execution time when the sequential program is parallelized.

4.1. Execution time of sequential program

First, looking at the execution of an ordinary user program on a single PE, since no communication cost between PEs exists, the factors which determine the execution time are the amount of work of the program and the processing performance per unit time of the PE.

If w is the total amount of work of the program and W is the processing performance per unit time, the ordinary program execution time T_1 on a single PE becomes $T_1 = w/W$.

4.2. Execution time of parallel program

Let us consider the case where the same program as above is executed in parallel. Here, let W be the processing performance per unit time inside the PE and let c be the communication processing performance per unit time. Then, from Section 2.3, let the amount of communication be proportional to pw/N. Therefore, if the loads are balanced and the communication delay is neglected, the communication time can be expressed, by an appropriate parameter γ, as $pw\gamma/(NC)$. Hence, the execution time T_N in parallel execution is as follows ($N > 1$):

$$T_N = \left(\frac{p}{N} + (1-p)\right)\frac{w}{W} + \frac{pw\gamma}{NC} = \frac{\Gamma_N}{N} T_1 \quad (7)$$

$$\Gamma_N = N - p\left(N - 1 - \gamma\frac{W}{C}\right) \quad (8)$$

Here, Γ_N expresses the performance degradation during parallel execution.

Let us examine this value for the cases of $p \to 0$ and $p \to 1$. When $p \to 0$, that is, the sequentiality of the program becomes higher, the amount of communication decreases and $T_N \to T_1$. This reflects the fact that sequential portion will not split unreasonably. When $p \to 1$, that is, the parallelism of the program becomes higher, it is seen from $T_N \to \frac{1}{N}(1 + \gamma\frac{W}{C})T_1$ that a communication overhead according to the amount of communication (determined by γ) will be added, not merely $1/N$ for sequential execution.

4.3. Conditions for high-speed implementation of parallel program

From the above, in order to make parallel execution faster than sequential execution, it is necessary and sufficient that $\frac{1}{N} \le T_N/T_1 \le 1$, that is, $1 \le \Gamma_N \le N$; from this, the condition $0 \le \gamma \le \frac{N-1}{W/C}$ is obtained. Hence, the communication content γ allowed in the parallel program is limited by the scale N of the parallel system as well as the processing performance per unit time W and the communication processing performance per unit time C. It can be seen clearly that the allowable communication becomes smaller as the

communication processing performance per unit time becomes increasingly inferior to the processing performance per unit time ($C < W$).

From the above condition, letting $\gamma = \frac{N-1}{W/C}\gamma'$, the following can be obtained:

$$\Gamma_N = N - (N-1)p(1-\gamma') \qquad (0 \leqq p, \gamma' \leqq 1) \tag{9}$$

5. Conditions of FTPS

From the viewpoints of reliability and speed, FTPS does not make sense unless there is a greater performance improvement than in the original parallel program. Accordingly, let us determine the conditions for the MWTF improvement of FTPS.

5.1. Execution time of FTPS

Similar to Section 4.2, by virtue of Section 2.4, the fault-tolerant processing time T'_N of the k-resilient FTPS is as follows:

$$\begin{aligned} T'_N &= \left(\frac{p}{n} + (1-p)\right)\left(\frac{w'}{W} + \frac{\beta w'}{C}\right) + \frac{p}{n}\gamma\frac{w}{C} \\ &= \frac{\Gamma'_N}{N}T_1 \end{aligned} \tag{10}$$

$$\begin{aligned} \Gamma'_N &= (1+\alpha)(1+\beta\frac{W}{C})(p(k+1)+(1-p)N) \\ &\quad + p(k+1)(N-1)\gamma' \end{aligned} \tag{11}$$

Γ'_N is the performance degradation factor during fault-tolerant execution. When the deterministic program is completely operated in the state machine, since no communication between replicas exists, $c' = 0$, that is, let the coefficient of communication between primary and backup be zero, $\beta = 0$. Moreover, since the execution on the replicas is completely the same as the original program, Γ'_N is calculated by letting the amounts of work of the programs be $w' \approx w$, namely, letting the rate of increase of the amount of work of the FTPS be $\alpha = 0$. Since a technique for realizing a program containing nondeterminism of the state machine is difficult to realize, this case corresponds to the ideal best case for a state machine only.

In the primary/backup approach, the amount of communication between replicas affects mostly the execution time. This amount is determined by β; however, efficient implementation can be achieved by choosing the checkpoint interval. Namely, if the number of checkpoints in the entire execution is s, it can be considered as a value of $\beta = \beta'/s$. However, because of coordinated checkpoints, β' will probably become larger to some extent.

The hybrid approach is the same as the primary/backup approach; however, the value of β' is smaller compared to the primary/backup approach; since s is determined by the nondeterminism of the program, it will be fixed by the program.

5.2. Calculation of MWTF

(1) Program on a single PE

In the ordinary execution of the program on a single PE, since there is no consideration of communication processing, the CPU processing performance W of the PE is regarded as the unaltered processing capability per unit time and can be obtained as follows:

$$MWTF_1 = W \times MTTF_{PE} \tag{12}$$

(2) Parallel program

During parallel execution, since a time T_N is used for processing an amount of work w, the processing performance per unit time W_N can be obtained as $W_N = w/T_N$. Hence, the $MWTF_N$ is calculated as

$$\begin{aligned} MWTF_N &= W_N \times MTTF_{MPP}(N) \\ &= \frac{1}{\Gamma_N} \times MWTF_1 \end{aligned} \tag{13}$$

(3) FTPS

Similarly, the $MWTF'_N$ of the FTPS can also be calculated:

$$\begin{aligned} MWTF'_N &= W'_N \times MTTF_{FTS}(N,k) \\ &= \frac{1}{\Gamma'_N}(k+1)\left(\sum_{i=0}^{k}\frac{1}{i+1}\right) \times MWTF_1 \end{aligned} \tag{14}$$

5.3. Condition for MWTF improvement of FTPS

In order to obtain MWTF improvement, it suffices that the MWTF improvement factor satisfy $I = MWTF'_N/MWTF_N \geq 1$, or, equivalently, the following:

$$(I =) \frac{\Gamma_N}{\Gamma'_N}(k+1)\sum_{i=0}^{k}\frac{1}{i+1} \geqq 1 \tag{15}$$

Figure 1 shows how the MWTF of the FTPS changes as the communication processing speed of the element processor becomes slower compared to ordinary processing for a 1-resilient FTPS with 100 element processors (N = 100).* W/C of the x-axis is the ratio of the processing performance per unit time (W) to the communication processing performance per unit time (C), and it approaches the origin as the communication processing becomes relatively

*Owing to the graphical presentation, we let $\sum_{i=0}^{k}\frac{1}{i+1} \approx \frac{1}{2}(\log(k+2) + 1 + \log(k+1))$

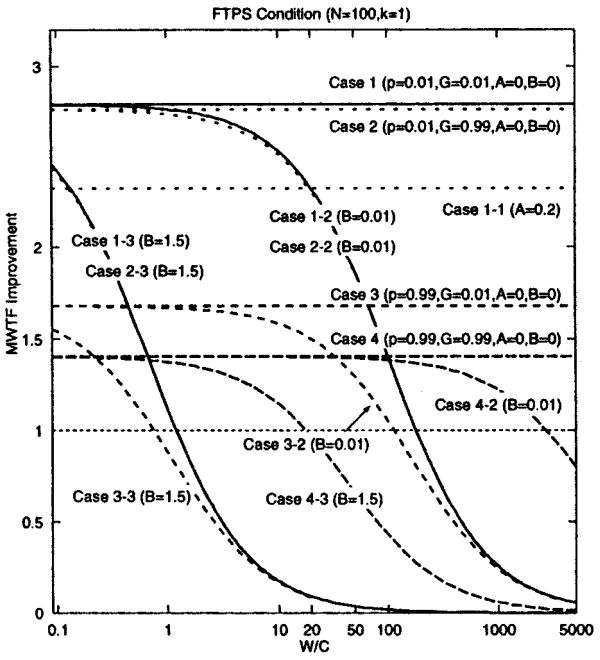

Fig. 1. MWTF improvement of FTPS (k = 1).

faster. Moreover, Fig. 2 shows, for part of Fig. 1, the change of the MWTF for a k-resilient FTPS for W/C = 20 and k = 1 to 5. Figure 1 is a semilogarithmic plot with the x-axis as the logarithmic axis and Fig. 2 is a bilogarithmic plot.

In Fig. 1, the straight lines of Cases 1 to 4 are plotted with the values of $\alpha = 0$ and $\beta = 0$; we also show the performance of the programs of the respective types obtained by the state machine approach (in the completely deterministic case). In the program of Case 1, the degree of parallelization ($p = 0.01$) and the amount of communication ($\gamma' = 0.01$) are also small and are close to the limit of the performance improvement obtained by the state machine approach. This value is obtained from the improvement factor of the MTTF and is the upper limit of the performance improvement of the FTPS based on replication. With respect to Case 1, the situation in which the amount of communication is increased ($\gamma' = 0.99$) is represented by the straight line of Case 2. It is seen from this result that an increase of the amount of communication of the original program will not affect the MWTF improvement at all. Case 3 is the case where only the degree of parallelization is raised ($p = 0.99$) and Case 4 is the case where the amount of communication and degree of parallelization are both higher. The straight line of Case 1-1 shows the case where the processing overhead of the FTPS is raised ($\alpha = 0.2$) with respect to Case 1.

Since the curves of Cases X-2 and X-3 contain communication processing for the FTPS, they show the performance of the FTPS based on the primary/backup and hybrid approaches. In Case X-2, the amount of communication for the FTPS is relatively small and corresponds to about 1% ($\beta = 0.01$) of the processing in a PE; in Case X-3, it corresponds to a situation which requires an amount of communication about 1.5 times as great ($\beta = 1.5$). Each of them drops abruptly, along with a relative drop of communication performance (increase of W/C), with Case X of the state machine as an upper limit.

Moreover, similarly to Cases 1 and 2, it is seen from Cases 1-X and 2-X that the increase of the amount of communication of the original program will not affect the MWTF improvement at all. Case 3-X, whose degree of parallelization is high, will lower the peak of the MWTF improvement compared to Cases 1-X and 2-X. However, even if the degree of parallelization is the same, the peak will drop but the value of W/C shifts to the right compared to Case 2-X in a program whose amount of communication is originally greater, as in Case 4-X. Namely, in the parallel program in which the degree of parallelization is high and the amount of communication is also greater, a large MWTF improvement by FTPS implementation cannot be expected; however, it is seen that it can tolerate utilization even if the communication performance of the system is relatively low.

It can be confirmed again from Fig. 2 that for a program whose degree of parallelization is low and whose amount of communication is small, as in Case 1-X, the MWTF will improve by a factor of the order of

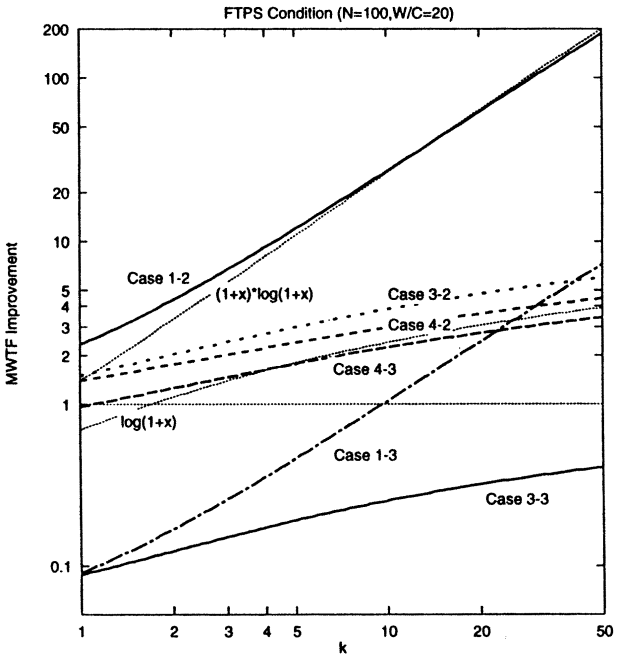

Fig. 2. MWTF improvement of FTPS (W/C = 20).

$(k+1)\log(k+1)$. On the other hand, it is seen that in a program whose degree of parallelization is higher (Case 3-X) or whose amount of communication is larger (Case 2-X), it will improve by a factor of the order of $\log(k+1)$. This agrees with the fact that if $p \to 1$ and $\gamma' \to 1$, I will be governed by $\sum^{k} 1/(i+1)$.

It is seen from the analysis that the upper limit of the MWTF improvement is reduced by an MTTF improvement of $(k+1)(1+\log(k+1))$, and that the limit becomes lower with a higher degree of parallelization of the program or a larger amount of communication. Thus, in a program whose parallelism is higher, since the parallelism of the system can be more effectively utilized, the gains realized from FTPS are small. On the contrary, when the parallelism which the program requires is smaller than the system scale, some freedom in utilizing the extra system resources owing to the FTPS for fault tolerance can be given to the user.

In the FTPS based on replication, an improvement of the MWTF can be obtained when the relative communication speed is faster. The fact that the relative communication speed is faster means that the fraction of processing on the processor that is blocked for communication is small. This trend becomes larger when the communication between replicas increases further, as in the primary/backup or hybrid approaches. In primary/backup, there are uncertain elements in the estimation, such as coordinated checkpointing or rollback processing, and the performance comparison generally cannot be made; however, the hybrid may be considered superior in terms of ease of realization. Moreover, in the above results, the state machine has been shown to have the best characteristics; however, attention must be paid to the fact that the present analysis is limited to the case of a completely deterministic program for the state machine only. In the general case containing nondeterminism, the realization in the state machine is difficult and the hybrid approach may be a realistic selection.

Moreover, in a system whose communication performance is low, the MWTF can be improved by increasing the number of replicas (value of k), but this lowers the potential performance of the system. The most effective solution may be to raise the relative communication performance of the system by increasing the execution units. This is equivalent to decreasing the actual number of communications by packing multiple communications between processors.

6. Conclusions

Based on the MWTF defined, we have analyzed in this paper the characteristics of the MWTF improvement when fault-tolerant software based on a process pair is realized in a parallel computer. In the analysis, it is assumed that ordinary communication between PEs of the user program and communication between PEs for fault tolerance have the same cost; however, it can be seen that the amount of communication required for fault tolerance will greatly affect the MWTF improvement. Moreover, when communication processing between PEs is slower than ordinary processing inside PE, the amount of communication for the support of fault tolerance will be severely limited. With respect to the increase of resiliency, the MWTF improvement is only of the order of log; however, the range of the applicable system performance balance will widen.

In the analysis, since the relative communication performance will become faster when the execution units of the program are large, it is seen that it is oriented to coarse-grained parallel programs. But since the parallel logic processing treated by us is fine-grained parallel processing oriented, performance improvement by fault-tolerant implementation is hard to achieve. The performance may be raised by arranging several executions as one transaction to increase the number of execution units; however, this realization is a subject of future study.

In this paper, we have analyzed the characteristics of the FTPS in terms of the processing speed of the parallel computer and the MWTF defined from the MTTF. Since the advantages of the parallel computer are not only speed but also abundant computing resources, we also plan to include the computing resources in the analysis.

REFERENCES

1. Guerraoui R, Schiper A. Software-based replication for fault tolerance. J IEEE p 68–74, April 1997.
2. Ueda K, Chikayama T. Design of the kernel language for the parallel inference machine. Comput J 1990;33:494–500.
3. Fujise T, Chikayama T, Rokusawa K, Nakase A. KLIC: A portable implementation of KL1. Proc FGCS'95, p 66–79, 1994.
4. Sugino E. A study on fault-tolerant implementation by parallel logic program conversion in a loosely coupled parallel system. Doctoral dissertation, Japan Advanced Institute of Science and Technology, 1997.
5. Sugino E, Yokota H. Construction of fault-tolerant software in parallel logic language. Proceedings of the Parallel Processing Symposium, JSPP'96, p 211–218.
6. Sugino E, Yokota H. A study on execution performance of fault-tolerant parallel programs in loosely coupled parallel computers. Fault-Tolerance Study Group of the IEICE Japan, Tech Rep IEICE 1997;FTS97:16–29.
7. Sugino E, Yokota H. Execution overhead of fault-tolerant parallel program for loosely coupled parallel systems. Proc JSPP'97, p 361–368.

8. Schlichting RD. An introduction to fault-tolerant software. Invited Talk of the International Workshop on Fault Tolerance in Information Systems (FIS95).
9. Kishimoto M, Nakajima J, Ohashi K, Kanazawa Y, Tsuchiya Y, Imai Y. Highly available operating system by process replication. Trans Inf Process Soc Japan 1997;38:2251–2261.
10. Cherif A, Suzuki M, Katayama T. A novel replication technique for detecting and masking failures for parallel software: Active parallel replication. IEICE Trans Inf Syst 1997;E80-D.
11. Chandra TD, Toueg S. Unreliable failure detectors for reliable distributed systems. J ACM p 225–267, March 1996.

AUTHORS (from left to right)

Eiji Sugino (member) graduated from the Department of Mathematics of Tokyo Metropolitan University in 1983. He then joined the SSL of Fujitsu Ltd. He moved to the Institute for New-Generation Computer Technology in 1986 and returned in 1990. In 1992, he enrolled in the first half of his doctoral course in the School of Information Science of Japan Advanced Institute of Science and Technology and completed the second half in 1997. He then became a research associate at the institute. He has been a lecturer in the Department of Software and Information Science of Iwate Prefectural University since 1998. He has a Doctor (Information Science) degree. He is engaged in research on packaging schemes of parallel programming language processing systems and fault-tolerant software. He is a member of the Information Processing Society of Japan.

Haruo Yokota (member) graduated from the Department of Physical Electronics of Tokyo Institute of Technology in 1980 and completed his master's course in information science there in 1982. He then joined Fujitsu Ltd. After serving at the Institute for New-Generation Computer Technology and Fujitsu Laboratories, Ltd., he became an associate professor at the Japan Advanced Institute of Science and Technology in 1992, and has been an associate professor in the Information Science and Engineering Research Division of the Graduate School of Tokyo Institute of Technology since 1998. He has a D.Eng. degree. He is engaged in research mainly on database and data engineering-oriented parallel architectures. He is a member of the Information Processing Society of Japan, the Japanese Society for Artificial Intelligence, IEEE, and ACM.

Copyright of Systems & Computers in Japan is the property of Wiley Periodicals, Inc. 2004 and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.

Copyright of Systems & Computers in Japan is the property of Wiley Periodicals, Inc. 2004 and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.