Analysis of Probabilistic Error Checking Procedures on Storage Systems

ING-RAY CHEN AND I.-LING YEN[1]

Institute of Information Engineering, National Cheng Kung University, University Road, Tainan, Taiwan

[1]*Department of Computer Science, Michigan State University, East Lansing, MI 48824-1027, USA*

Email: irchen@iie.ncku.edu.tw

Conventionally, error checking on storage systems is performed on-the-fly (with probability 1) as the storage system is being accessed in order to improve the reliability of the storage system. However, such a procedure may needlessly cause degraded performance due to the extra processing time needed for executing the error checking code. In this paper, we consider fault-tolerant storage systems designed to provide continuous services to customers over a mission period and the design goal is (1) to maximize the cumulative number of requests that the storage system can service without failure over the mission period or (2) to be able to service at least a given number of requests without failure over the mission period with its system reliability maximized. We develop a Markov reward model to identify the design conditions under which probabilistic error checking procedures can better satisfy this design goal than conventional on-the-fly error checking procedures. The result helps determine the best time interval between successive executions of the error checking procedure to meet such design goal and is useful for designing adaptive systems that can temporarily tradeoff reliability for performance to meet design goal (2) in response to dynamic workload changes in the environment.

Received March 21 1995, revised July 26 1995

1. INTRODUCTION

A storage system with error checking/recovery capability must incorporate redundant data for discovering and recovering from errors caused by hardware or software faults. An example of such a storage system is a robust data structure (Taylor *et al.*, 1980; Davis, 1989; Li *et al.*, 1989; Fujimura and Jalote, 1993) which contains useful data as well as structural, redundant information such as extra identifier fields, additional pointers, a count for the number of elements in the data structure, etc. The extent to which such systems can detect and recover from errors depends on the amount of structural redundancy included in the data structure. For example, a single-linked linear list is 0-correctable (i.e. cannot recover from even a single error) because changing one pointer erroneously will destroy the list, thus possibly resulting in a system failure. A more robust list structure is a doubly linked list formed by adding to each node an extra identifier field and an extra back pointer field pointing to the predecessor of the node in the list. The resulting doubly linked linear list is 1-correctable (i.e. can recover from a single error) because it has two independent, disjointed sets of pointers, each of which can be used to reconstruct the list. In general, a storage system formed this way can cope with more errors by incorporating more redundant information into the system. Other examples include error correcting codes (ECCs) derived from Hamming codes (Hamming, 1950), such as those based on algorithm-based fault-tolerance (Feng *et al.*, 1993; Johansson, 1993; Du *et al.*, 1994).

Naturally, due to the use of redundancy, the reliability of the storage system is improved; however, system performance is degraded unavoidably due to the extra processing time required for performing error checking/recovery activities. This is especially the case when the error checking/recovery action is performed on-the-fly (i.e. on a per access basis). In the context of robust data structures (Taylor *et al.*, 1980), it was suggested that periodic error checking/recovery procedures be used to reduce the error checking overhead; however, neither an analysis nor a performance metric was discussed, although not performing on-the-fly error checking for robust data structures to tradeoff reliability for performance was considered a design possibility. In the literature, many research efforts have been reported in the area of performance evaluation of periodic data structure maintenance algorithms for optimizing the service rate of data structure servers (e.g. Chen and Banawan, 1992, 1993; Chen, 1995), but no performance analysis of periodic error checking algorithms for fault-tolerant storage systems has yet been carried out. Conceivably, a long error checking interval will increase the probability of multiple errors, thus making the system unreliable. On the other hand, a short interval may unnecessarily degrade system performance, especially if the error rate is low. Thus, there is a tradeoff between performance and reliability, determined by the length of the time interval between successive executions of the error checking/recovery procedure.

The paper develops a Markov reward model that helps determine the time interval between successive

executions of the error checking/recovery procedure for fault-tolerant storage systems whose design goal is to (1) maximize the cumulative number of requests that the system can service without failure over a given mission time interval or (2) meet the requirement of being able to service at least a given number of requests without failure over a given mission time interval with its reliability maximized. This design goal is required for embedded storage systems for which time-critical requests for services arrive continuously during a mission period and the system must service at least a given number of requests over the mission period, or a system failure may result because request deadlines are missed. Example application systems include dynamic adaptive systems such as CHAOS (Bihari and Schwan, 1991) that must adapt to changes in the environment to tune the system to meet required performance and reliability goals demanded by the environment, given bounded computational resources. When such a system is overloaded during a transient period it can temporarily tradeoff reliability for improved performance by adjusting the error checking interval so as to meet the design goal. In the literature, such design goal corresponds to the *performability* metric (Meyer, 1980; Pattipati *et al.*, 1993). It is a mission-specific measure of system effectiveness that combines the reliability and performance measures of the system since the system is not able to service any operation when it fails.

In this paper, 'probabilistic error checking' is a form of periodic error checking. It means that for each operation that accesses the storage system, error checking is performed with a probability q, $0 \leqslant q \leqslant 1$. This, of course, includes the special case of 'on-the-fly error checking' for which $q = 1$. In addition, when the storage system is idle, the system ultilizes its spare processing time to perform background error checking to improve the reliability of the system. We are interested in knowing under a set of design conditions, including the mission time, the request workload to the storage system, the service rate, the error rate, etc., what value of q we should choose to meet the system design goal. The q value chosen can be used to determine the length of the interval between successive executions of the error checking/recovery procedure. For example, if $q = 0.2$ and the request arrival rate is 3 requests/s, then the period of performing error checking is about once in every 1.7 s (1 error checking/5 requests $\times$ 3 requests/s). With this formulation, our analysis also implicitly considers the special case where the system does not have any error diagnostic capability, i.e. $q = 0$. Therefore, the model developed in the paper can be used to compare the performability of a system with some periodic error diagnostic capability with another system without this capability.

The problem we like to solve is mathematically formulated as follows. Let $c(t)$ be the conditional service throughput at time t conditioned on the system being alive at time t. Since $c(t)$ depends on whether the system is alive at time t, it is closely related to the reliability of the system $R(t)$, which in turn depends on how we perform error checking per access during the time interval $[0, t]$. The performability of the storage system at the end of the mission time period T is given by

$$\mathscr{P}(T) = \int_0^T c(t)\mathrm{d}t \tag{1}$$

which gives the total number of requests serviced over $[0, T]$, also conditioned on the system being alive over the time interval $[0, T]$. The problem at hand is to identify, for a given mission period T under a set of design conditions, the best q value (or the best periodic error checking interval) that maximizes $\mathscr{P}(T)$. If the system is designed to satisfy a minimum performability requirement, say $\mathscr{P}_{\min}(T)$, over a time interval $[0, T]$ while maximizing its reliability. The problem is then to select the best q value that meets the following constraint:

select q s.t. $\mathscr{P}(T) \geqslant \mathscr{P}_{\min}(T)$ while maximizing $R(T)$

The rest of the paper is organized as follows. Section 2 defines the notation and discusses model assumptions. Section 3 develops a Markov model for describing the behavior of a fault-tolerant storage system which uses a probabilistic error checking/recovery procedure and derives expressions for the conditional service throughput and performability of the system. Section 4 demonstrates the utility of the model by showing some numerical results of an example storage system. Finally, Section 5 summarizes the paper.

2. NOTATION AND MODEL ASSUMPTIONS

Our model of a storage system concerns the most commonly used type: 1-correctable. (Other types can be analyzed similarly). The storage system can process only one request at a time. Further, the storage system is assumed to be an *atomic* entity (Bernstein *et al.*, 1987) so that a request is either fully serviced or not serviced at all. The storage system receives access operations to process the information stored in the system and sends back responses on a per access basis. The time interval between arrivals of two consecutive access operations is assumed to be exponentially distributed. All access operations (reads and writes) require about the same amount of service time which is also assumed to be exponentially distributed. An error checking operation is always atomic and requires a service time that is also assumed to be exponentially distributed. Each access operation may be followed by an error checking operation which occurs with probability q. If the data accessed by an operation contains an error, it will be detected and corrected by the 1-correctable error checking/recovery code and the operation is still supplied with correct information. A system failure occurs when an operation is supplied with erroneous data. We assume that during error checking no new error will occur, i.e. the error rate during error/checking/recovery is essentially zero. Also, when the storage system is idle, the system utilizes the spare processing time to perform background error checking which is preempted immediately when a request arrives.

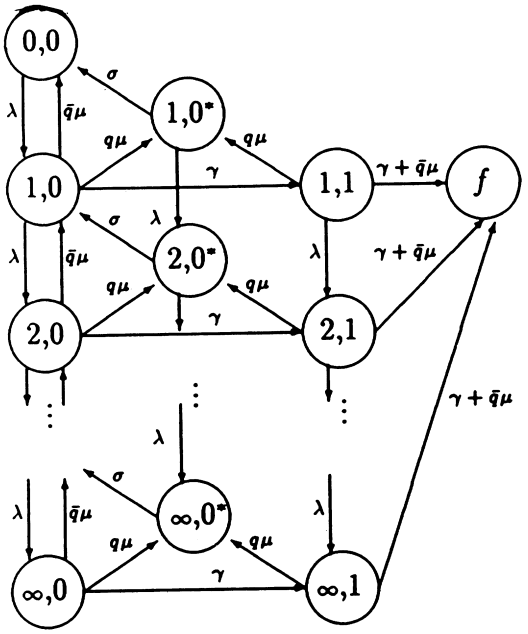

FIGURE 1. The Markov model for 1-correctable storage systems. An asterisk indicates that an error check is being performed.

The following notation is used throughout the paper.

λ	rate at which access operations arrive.
μ	rate at which an access operation is serviced by the system.
σ	execution rate at which the error checking/recovery procedure is performed.
γ	error rate, either software or hardware-induced.
q	probability of performing error checking per access, $0 \leqslant q \leqslant 1$.
$\bar{q} = 1 - q$	probability of not performing error checking per access.
s	state of the system $= (i, j)$, where $i =$ queue length and $j =$ number of errors.
$\Pr_s(t)$	Pr{system is in state s at time t, given that it was in the initial state (0,0) at time 0}.
$c(t)$	conditional service throughput of the system at time t conditioned on the system being alive at time t.
T	mission period.
$\mathscr{P}(T)$	$\int_0^T (t)\mathrm{d}t =$ performability of the system over the mission time period $[0, T]$.
f	'failed' state.
$R(t)$	reliability of the system at time $t = 1 - \Pr_f(t)$.

3. MODEL

From the problem formulation, it can be seen that the behavior of the storage system depends on two quantities, i.e. the queue length and the current error level. The queue length describes the queueing behavior of the system while the error level determines whether the system is functionally correct or not. Hence, to adequately model the system dynamics, a Markovian model can be used in which each state has these two components in its representation, the first of which is the queue size and the second is the error level.

We assume that the request interarrival time is exponentially distributed with a constant rate λ. Then, the behavior of the storage system under an error checking/recovery algorithm can be described by the two-component Markov model shown in Figure 1. States $(j, 0^*), j \geqslant 1$, mean that the system is performing error checking (in which the system may contain 0 or 1 error).

The model is constructed as follows:

1. A customer arrival increases the queue length by 1, but has no effect on the error level (the vertical transitions at rate λ downward).
2. When the system contains 0 error and is not idle (i.e. the queue is not empty), the queue length will be reduced by 1 if the system services an operation with rate μ without executing the error checking procedure which occurs with probability $\bar{q} = 1 - q$ (the upward vertical transitions at rate $\bar{q}\mu$).
3. The error level will be increased by 1 when an error occurs in the system. This occurs at rate γ (the horizontal transitions at rate γ). During an error checking period, i.e. when the system is in state (0, 0) in which background error checking runs or in states $(j, 0^*)$ in which error checking runs after data access, no new error will occur. Therefore, when the system is in these states, the error rate is zero. The Markov model that considers a non-zero error rate during these periods is similar to the one given above (Chen and Yen, 1995).
4. An error checking operation takes place with probability q following every access operation. It is an action which may occur when the system contains 0 or 1 error. In the former case, the system transits from $(j, 0)$, $j \geqslant 1$, to $(j, 0^*)$ with probability q and rate μ (the diagonal transitions at rate $q\mu$) and then from $(j, 0^*)$ to $(j - 1, 0)$ with rate σ (the diagonal transitions at rate σ). In the latter case, it transits from state $(j, 1)$ to $(j, 0^*)$ also with probability q and rate μ (the diagonal transitions at rate $q\mu$) and then to $(j - 1, 0)$ with rate σ. The queue size is reduced by 1 after the error checking/recovery procedure is executed and the user is supplied with correct information.
5. When the system contains an error and has at least one customer, it will fail if it does not execute the error checking procedure following an access operation, an event which occurs with probability $\bar{q}$ and rate μ (the horizontal transitions from state $(j, 1)$ to state f at rate $\bar{q}\mu$). Also, since the 1-correctable storage system cannot cope with more than one error, the system fails when it is not idle and the error level accumulates to two (the horizontal transitions from states $(j, 1)$ to state f at rate γ).

The initial state of the system is (0, 0) at time 0 and the absorbing state is f. The reliability of the system for any time interval $[0, t]$ is given by

$$R(t) = 1 - \Pr_f(t).$$

where $\Pr_f(t)$ denotes the probability that the system is in state f at time t. To compute the performability of the system $\mathscr{P}(T)$ over a given mission period T based on equation (1), we need to derive an expression for $c(t)$ conditioned on the fact that the system is alive at time t, i.e. the system is not in state f at time t. Now, among all states except state f, only states $(j, 0)$ and $(j, 0^*)$, $1 \leqslant j \leqslant \infty$, contribute to the conditional service throughput of the system because in state (0, 0) the system is idle while in states $(j, 1)$, $1 \leqslant j \leqslant \infty$, the system is either servicing a request erroneously or accessing data which must be subsequently checked/recovered by the error checking procedure before the operation can be fully serviced. (When an error checking operation is performed following an access operation, the system completes the service of the access operation only after it completes error checking/recovery. Therefore, completing error checking rather than accessing data in this case in effect contributes to the service throughput.) On the other hand, in states $(j, 0)$ the system can fully serve one of the j customers with rate $\bar{q}\mu$ and in states $(j, 0^*)$, with rate σ. Here, we can view $\bar{q}\mu$ and σ as reward rates associated with these last two sets of states, respectively, representing the 'service contribution' of these states. As a result,

$$c(t) = \sum_{j=1}^{\infty} \{\Pr_{(j,0)}(t) \times \bar{q}\mu + \Pr_{(j,0^*)}(t) \times \sigma\} \qquad (2)$$

where we note that the first term exists only if $\bar{q} > 0$ and, conversely, the second term exists only if $q > 0$.

The relationship among $c(t)$, $R(t)$ and q at any time t is quite interesting. As q decreases (say from $q = 1$ to $q = 0$), $R(t)$ decreases steadily because with less error checking the system is more likely to fail at time t. On the other hand, the relationship between $c(t)$ and q is dictated by the tradeoff between the first and second terms in equation (2). When t is a small value, the probability of the system being in states $(j, 1)$, $j \geqslant 1$, or state f, is very low. The system thus can obtain a higher $c(t)$ value with a smaller q value since it can use more CPU time to perform useful service work in states $(j, 0)$ without having to perform unnecessary error checking in states $(j, 0^*)$. Note that when $q = 0$, the contribution of the second term in equation (2) is zero and the first term completely dominates the second term. Thus when t is small, $c(t)$ increases as q decreases. When t is a large value, however, $c(t)$ decreases as q decreases. This is because the probability of the system being in states $(j, 1)$, $j \geqslant 1$, or state f, increases as t increases, especially when q is small. The contribution of the first term in equation (2) in this case becomes less significant. Consequently, when t is large, the system can obtain a higher $c(t)$ value if it performs error checking more frequently (i.e. uses a higher q value) to prevent a system failure, allowing the second term to contribute more to $c(t)$.

It should be mentioned that the Markov model in Figure 1 can be easily extended to handle other workload situations. An example is a closed system in which the number of users (e.g. terminals in multiprogramming environments) issuing the operation is fixed and once a user issues a request, it waits until the pending operation is serviced. Assume that there are n such users in the system, each of which issues an operation at rate Z^{-1} (Z refers to the average think time of a terminal user), then the Markov reward model (Figure 1) remains the same except for two changes. First, the number of states is now finite because the queue size cannot exceed the number of users in the system, e.g. if $n = 2$, then only eight states exist, namely, states f, (0, 0), (1, 0), (2, 0), (1, 0^*), (2, 0^*), (1, 1) and (2, 1). Second, the downward rates λ's from top to bottom along the same error level are replaced by n/Z, $(n-1)/Z$, ..., $1/Z$. Another situation is when the queue size is bounded so that an operation arriving at the moment when the queue is full will turn away and retry later. In this case, the Markov reward model remains the same except that the states in which the queue has overflowed are removed, e.g. if the queue size is 2, then only eight states exist. Therefore, the Markov model in Figure 1 is applicable to various workload and execution environment situations.

4. ANALYSIS AND RESULT INTERPRETATION

In this section, we demonstrate the utility of the model by a case study and present a physical interpretation of the results. The storage system under consideration is implemented with a disk-resident, 1-correctable, 2-detectable doubly linked list data structure containing not only useful data, but redundant information for error recovery, including back pointers, a count field storing the total number of nodes on the list, and an extra identifier field associated with each node (Taylor *et al.*, 1980). The 1-correctable doubly linked list is useful, for example, in hashing and buddy systems (Aho *et al.*, 1983).

The storage system is embedded within a larger open system. It services one customer at a time by manipulating the data stored in the data structure and is characterized by the following:

1. Customers arrive independently each requesting an operation to the storage system and the interarrival time is exponentially distributed with an average rate of 3 customers/s ($\lambda = 3\,\mathrm{s}^{-1}$).
2. The service time per operation (read or write) is exponentially distributed with an average service rate of 5 operations/s ($\mu = 5\,\mathrm{s}^{-1}$).
3. The time required to execute the error checking/recovery procedure is also exponentially distributed with an average rate of 5 executions/s ($\sigma = 5\,\mathrm{s}^{-1}$).
4. The time interval between the occurrences of two consecutive errors is exponentially distributed with an average of approximately 20 days ($\gamma = 5 \times 10^{-7}\,\mathrm{s}^{-1}$).

These parameter values represent a set of design conditions under which we are interested in determining

the value of q that can maximize $\mathscr{P}(T)$ as a function of the mission time T. These chosen values reveal the fact that there exists an optimal q value which varies as the mission time T varies. We also investigate how the optimal value of q varies when one of the parameters varies, i.e. changing the σ value from 1 to $10\,s^{-1}$ in increment of 1, while keeping the others fixed. The SHARPE tool (Sahner and Trivedi, 1991) was used to yield numerical results for $c(t)$ and $\mathscr{P}(T)$ from evaluating the Markov reward model instantiated with these parameter values. It should be noted that while only one set of design condition is reported here, we have also evaluated other sets of design conditions and all the numerical results showed a similar trend: there exists an optimizing q value which can maximize $\mathscr{P}(T)$ as T varies.

Figure 2 shows the conditional service throughput $c(t)$ as a function of t and q, $0 \leqslant q \leqslant 1$. When the system performs error checking on-the-fly, i.e. $q = 1$, the system's conditional service throughput is virtually constant because the service rate ($\mu = 5$) and the error checking/recovery execution rate ($\sigma = 5$) are much higher than the error-induced rate ($\gamma = 5 \times 10^{-7}$, about once in 20 days). Nevertheless, $c(t)$ becomes more sensitive to time when the error checking/recovery procedure is executed only probabilistically, which is modeled by the parameter q. As can be seen from Figure 2, a low q value (which corresponds to a long interval between error checking executions) will result in a high $c(t)$ value when t is small and a low $c(t)$ value when t is large. This is because when t is small, the system is likely to be reliable and, due to the reduced processing time for periodically executing the error checking/recovery procedure, the system has more processing time to service operations, thus resulting in a higher service throughout. Conversely, when t is large, the system under a low q value is likely to be unreliable because of a high probability of multiple

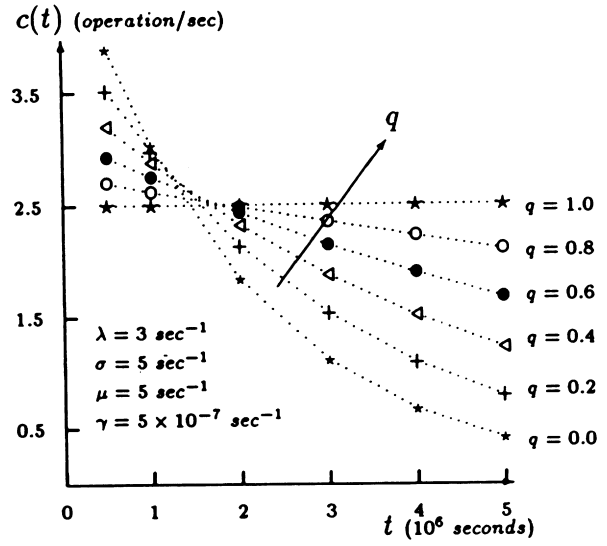

FIGURE 2. Conditional service throughput $c(t)$ of a 1-correctable storage system.

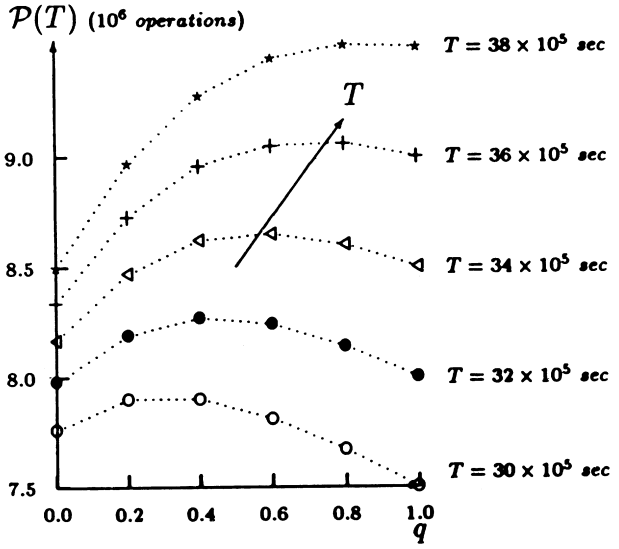

FIGURE 3. Performability $\mathscr{P}(T)$ of a 1-correctable storage system ($\lambda = 3\,s^{-1}$, $\sigma = 5\,s^{-1}$, $\mu = 5\,s^{-1}$, $\gamma = 5 \times 10^{-7}\,s^{-1}$).

errors being induced into the system due to the fact that error checking is not performed frequently enough (i.e. not performed on-the-fly). Consequently, in this case the system's conditional service throughput deteriorates because the system is unable to service any operation when it fails.

Figure 3 shows the corresponding system performability as a function of q as T varies based on equation (1). The time lines are separated by about 2 days of operational time. Obviously, the best value of q for optimizing the performability of the system depends strongly on the duration of the mission period. When $\lambda = 3$, $\mu = 5$, $\sigma = 5$ and $\gamma = 5 \times 10^{-7}$, if the expected mission period is 30×10^5 s (about 35 days) then $q = 0.2$, or, executing the error checking procedure once in approximately every five access operations, can improve the system performability by a significant extent when compared with the 'on-the-fly' case (i.e. $q = 1$) in terms of the total number of operations which can be serviced by the system over the mission period. This optimal q value increases from 0.2 to 1.0 as the mission period increases from 35 to 45 days under this set of parameter values, after which performing error checking 'on-the-fly' gives the best system performability.

Figure 4 shows how the optimal value of q varies when σ is changed while keeping the other parameters fixed. It shows that when the error checking rate is higher than the service rate per operation and the continuous mission time is below 40×10^5 s (about 45 days), the system should perform error checking on-the-fly (i.e. $q = 1$) to improve performability since the overhead of error checking is relatively low. On the other hand, when the error checking rate is lower than or about the same as the service rate per operation, the system should only perform error checking probabilistically (and thus periodically) so as to maximize the number of requests it can service over the mission time

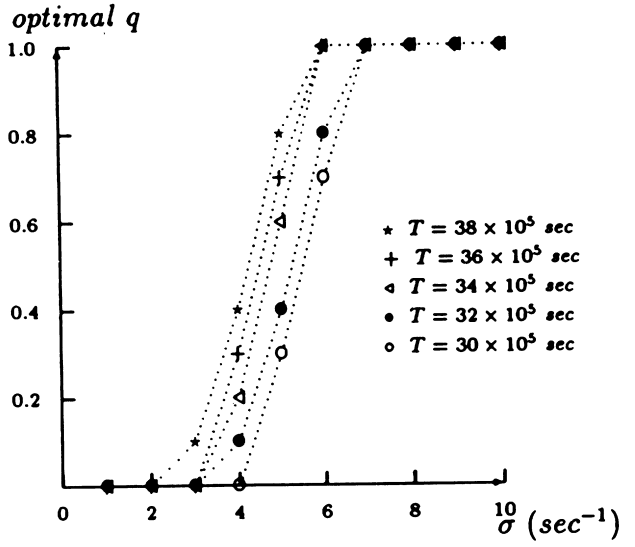

FIGURE 4. Optimal q values as a function of σ with other parameters fixed ($\lambda = 3\,s^{-1}$, $\mu = 5\,s^{-1}$, $\gamma = 5 \times 10^{-7}\,s^{-1}$).

period. In the latter case, the optimal value of q increases as the mission time T increases for the same σ value because the system is likely to be less reliable (i.e. more likely to contain an error) when T is larger, thus requiring a higher q value (i.e. more error checking) to prevent a system failure.

If the storage system is designed to meet a minimum performability requirement over a mission period, then the Markov model developed in the paper can also be used to predict the best q value that satisfies the minimum performability requirement while maximizing the system reliability. This is done by analyzing the numerical results and selecting the q value as large as possible among all q values at which the minimum performability requirement is satisfied, since $R(T)$ is monotonically increasing with increasing q. For example, in Figure 3 if the mission period is 30×10^5 s (about 35 days) and the minimum performability is $\mathscr{P}_{\min}(35\text{ days}) = 7.6 \times 10^6$ operations, then one should select $q = 0.9$ (for which $\mathscr{P}(35\text{ days}) = 7.6 \times 10^6$ operations) over $q = 0.2$ (for which $\mathscr{P}(35\text{ days}) = 7.9 \times 10^6$ operations) even though $q = 0.2$ yields a higher performability over the mission period.

5. CONCLUSION

We have developed a Markov reward model to analyze the effect of probabilistic error checking/recovery strategies on fault-tolerant storage systems whose design goal is to maximize the number of requests that the system must service without failure over a mission time interval or to meet a minimum performability requirement while maximizing the system reliability. The utility of the model was demonstrated by applying it to an example system from which the results showed that the optimizing probability of performing error checking per access, that is, q, depends not only on the workload characteristics of the execution environment (i.e. λ, μ, σ, γ, etc.) but also on the expected mission period (i.e. T) over which the service is to be provided by the storage system. The conclusion is that it is possible to select the best q value for meeting the design goal of such fault-tolerant storage systems and that the best q value is not a constant. The finding of the paper helps determine the best time interval between two successive executions of the periodic error checking procedure to meet such a design goal for fault-tolerant storage systems. The analysis result is also useful in designing adaptive systems that can temporarily tradeoff reliability for performance to meet a minimum performability requirement while maximizing the system reliability in response to dynamic workload changes in the environment.

ACKNOWLEDGEMENTS

The authors would like to thank the referees for their helpful suggestions which have greatly increased the presentation and clarity of the paper.

REFERENCES

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1983) *Data Structures and Algorithms*. Addison-Wesley, Reading, MA.

Bernstein, P. A., Hadzilacos, V. and Goodman, N. (1987) *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA.

Bihari, T. E. and Schwan, K. (1991) Dynamic adaptation of real-time software. *ACM Trans. Comp. Syst.*, **9**, 143–174.

Chen, I. R. and Banawan, S. A. (1992) A reduced Markov model for the performance analysis of data structure servers with periodic maintenance. *Comp. J.*, **35**, A363–A368.

Chen, I. R. and Banawan, S. A. (1993) Modeling and analysis of concurrent maintenance policies for data structures using pointers. *IEEE Trans. Soft. Eng.*, **19**, 902–911.

Chen, I. R. (1995) A degradable B^{link}-tree with periodic data reorganization. *Comp. J.*, to appear.

Chen, I. R. and Yen, I. L. (1995) *Probabilistic Error Checking on Storage Systems*. Technical Report NCKU-IIE-95-05, Institute of Information Engineering, National Cheng-Kung University, Tainan, Taiwan.

Davis, I. J. (1989) Local correction of helix (k) lists. *IEEE Trans. Comp.*, **38**, 718–724.

Feng, G. L., Rao, T. R. N. and Kolluru, M. S. (1993) Error correcting codes over Z_{2^m} for algorithm-based fault tolerance. *IEEE Trans. Comp.*, **43**, 370–373.

Fujimura, K. and Jalote, P. (1993) On robustness of B-trees. *IEEE Trans. Knowledge and Data Eng.*, **5**, 530–533.

Gu, D., Rosenkrantz, D. J. and Ravi, S. S. (1994) Construction of check sets for algorithm-based fault tolerance. *IEEE Trans. Comp.*, **43**, 641–650.

Hamming, R. W. (1950) Error detecting and error correcting codes. *Bell Syst. Tech. J*, **29**, 147–160.

Johansson, R. (1993) A class of (12,8) codes for correcting single errors and detecting double errors within a nibble. *IEEE Trans. Comp.*, **42**, 1504–1510.

Li, C.-C. J., Chen, P. P. and Fuchs, W. K. (1989) Local concurrent error detection and correction in data structures using virtual back pointers. *IEEE Trans. Comp.*, **38**, 1481–1492.

Meyer, J. F. (1980) On evaluating the performability of degradable computing systems. *IEEE Trans. Comp.*, **29**, 720–731.

Pattipati, K. R., Li, Y. and Blom, H. A. P. (1993) A unified

framework for the performability evaluation of fault tolerant computer systems. *IEEE Trans. Comp.*, **42**, 312–326.

Sahner, R. A. and Trivedi, K. S. (1991) *SHARPE Language Description*. Duke University.

Taylor, D. J., Morgan, D. E. and Black, J. P. (1980) Redundancy in data structures: improving software fault tolerance. *IEEE Trans. Software Eng.*, **SE-6**, 585–594.