Concise Papers

Efficient Algorithms For Selection of Recovery Points in Tree Task Models

Subhada K. Mishra, Vijay V. Raghavan, and Nian-Feng Tzeng

Abstract—**In this note we develop efficient solutions to the problem of optimally selecting recovery points. These solutions are intended for models of computation in which task precedence has a tree structure and a task may fail due to the presence of faults. For the binary tree model, an algorithm to minimize the *expected* computation time of the task system under a uniprocessor environment has been developed. The algorithm has time complexity of $O(N^2)$, where N is the number of tasks, while previously reported procedures have exponential time requirements. The results have been generalized for an arbitrary tree model.**

Index Terms—**Checkpoints, dynamic programming, error recovery, reliability, rollback.**

I. Introduction

Rollback and recovery strategies have been widely used in several software systems to provide reliability and fault tolerance. These techniques have, in particular, been used to support integrity and high availability of complex software systems such as a DBMS and an OS. Recovery from a failure typically involves saving the (relevant) state of the system so that when an error is detected, the system can be restored to a previously saved state and the execution restarted from that point. A survey of earlier work on analytical models for investigating rollback and recovery strategies can be found in [1]. The model proposed by Gransky *et al.* [4] for backward error recovery, for instance, is a structured approach for providing fault tolerance in general in software.

A task system representing a computation is a pair $(T, <)$, where T is a set of tasks and $<$ is a precedence relation (directed acyclic graph). In many cases, the computation can be modeled by a task system where the precedence relation is a tree. In practice, such computations arise quite frequently, particularly when the underlying algorithm involves divide and conquer, dynamic programming, or when the main data structure used in an application is a tree. A common example from the database environment is the processing of a query that involves the *join* of a number of relations.

Recovery points (or *checkpoints*) have been the basis of several recovery schemes. The positioning of the recovery points involves certain trade-offs with respect to objectives such as *minimum completion time, minimum recovery overhead,* and *maximum throughput.* Some of these trade-offs have been presented in [1]. Several aspects of recovery point selection, including their performance, have been under investigation. References to the literature on recent research can be found in [3].

The need for higher reliability of software is growing also in areas other than real-time processing (e.g., database applications, information systems) and this trend is likely to grow in the future, particularly as software becomes more complex. In such environments, the minimum computation time is more appropriate an objective rather than satisfying recovery time constraints.

The selection of recovery points in a task system modeled by a reverse binary tree has been studied by Chen *et al.* [3]. The authors analyze the complexity of placing recovery points in such a way that the *expected* computation time, in the presence of faults, is minimized. They also suggest that the approach can be extended to an *m-ary* tree model, for an arbitrary m, of the task system. Both uniprocessor and multiprocessor systems have been taken into consideration as alternatives for the processing environment.

Manuscript received June 22, 1990; revised March 13, 1991. Recommended by N. G. Leveson.

The authors are with the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA 70504.

IEEE Log Number 9100234.

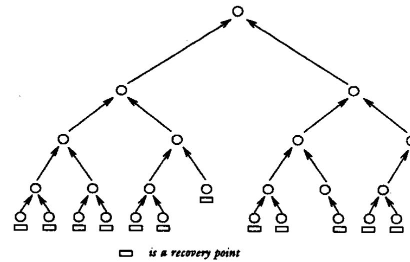

Fig. 1. Model of task system with recovery points.

In this note we develop an algorithm to find an optimal assignment of recovery points; i.e., to find the assignment of recovery points that minimizes the *expected* computation time of the task system. The discussions are based on the binary tree task model and the results are readily extended to an *m-ary* tree. While the earlier studies have not given algorithms that are better than exponential time complexity, we show how an algorithm that has $O(N^2)$ time complexity, where N is the number of tasks in the task system, can be designed for the binary tree task model.

Dynamic programming has been used previously under a slightly different model of computation to minimize the maximum and expected time spent in saving program states [2]. As discussed in [3], however, a direct application of dynamic programming to our problem results in an algorithm that requires exponential time.

In Section II we set up the model of the task system and define the problem; i.e., the optimal assignment of recovery points. Section III discusses the problem in the context of dynamic programming. In Section IV we formulate the new solution. Section V describes the algorithm to compute the optimal assignment and analyzes and derives the time complexity of the algorithm. In Section VI we extend the solution to the case of *m-ary* trees. The conclusions are stated in the last section.

II. Model of the Task System

The model of computation used in this study is similar to that in [3]. Specifically, the task system is modeled as a reverse binary tree (i.e., the reverse tree represents the precedence relationships). As shown in Fig. 1, each node in the tree represents a task or process. The leaf nodes represent tasks that are the starting points of the computation in the system; these tasks can be executed concurrently. A task represented by a parent node can begin execution only after all the tasks represented by its child nodes have completed execution. The computation is said to have finished when the task at the root node completes execution. (Henceforth we use "node" to mean "task represented by the node.")

A recovery point (RP) associated with any node is set up prior to the execution of that node. Under uniprocessor environment, intertask communication is negligible and therefore ignored.

Associated with each task are the following:

t_i the time required to complete task i (in the absence of faults),
s_i the time required to set up a recovery point before task i,

0098-5589/91/0700-0731$01.00 © 1991 IEEE

r_i the time required to rollback computation to the recovery point before task i, and

p_i the probability that task i completes without failure.

Failures are assumed to occur independently of each other and can be detected by appropriate acceptance tests (AT) (the time taken to carry out the AT is assumed to be part of t_i).

We further define the following notations for a uniprocessor system environment where task executions may not overlap in time: let T_i be the expected computation time of the task system represented by the subtree rooted at task i; E_i be the expected computation time of task i alone after it is invoked; and K_i be the expected time *starting* when task i fails its AT *until* it resumes computation.

The following expressions for the computation times defined above can easily be derived based on simple principles of probability [3]:

1) if there is a recovery point before task i,

$$K_i = r_i \tag{1a}$$

$$E_i = t_i/p_i + (1/p_i - 1)K_i \tag{1b}$$

$$T_i = T_{il} + T_{ir} + E_i + s_i \tag{1c}$$

2) otherwise,

$$K_i = K_{il} + E_{il} + K_{ir} + E_{ir} \tag{2a}$$

$$E_i = t_i/p_i + (1/p_i - 1)K_i \tag{2b}$$

$$T_i = T_{il} + T_{ir} + E_i \tag{2c}$$

where il and ir are the root nodes of the left and right subtree of node i, respectively.

III. Optimal Placement of Recovery Points

Based on the model defined previously, the optimization problem can be formally stated as follows: let there be N nodes in the task system modeled by a reverse binary tree under a uniprocessor environment; given t_i, s_i, r_i, and p_i for each node, select a set of nodes at which to place the recovery points so that the *expected* computation time of the task system is a minimum.

An immediate observation is that each leaf node must be preceded by an RP, since otherwise it would not be possible to recover from failure of a leaf node. Even then, an exhaustive evaluation of the $2^{N/2}$ possible assignments would lead to time complexity $O\left(N \cdot 2^{N/2}\right)$ (each evaluation, i.e., computation of T_{root}, would require time $O(N)$ and there are approximately $N/2$ internal nodes).

It is easy to see that, for any given i, E_i is a constant function of K_i (from (1b) and (2b)). Therefore a bottom-up construction of the optimal solution will require only T_i and K_i to be computed at each step. The aforementioned equations thus reduce to

1) if there is a recovery point before task i,

$$K_i = r_i \tag{3a}$$

$$T_i = T_{il} + T_{ir} + t_i/p_i + (1/p_i - 1)K_i + s_i \tag{3b}$$

2) otherwise,

$$K_i = t_{il}/p_{il} + K_{il}/p_{il} + t_{ir}/p_{ir} + K_{ir}/p_{ir} \tag{4a}$$

$$T_i = T_{il} + T_{ir} + t_i/p_i + (1/p_i - 1)K_i. \tag{4b}$$

An intuitive way to partition the problem would be: let T_i^k be the optimal expected time of computation for the subtree rooted at i, with a maximum of $k < \eta(i)$ recovery points allowed, namely,

$$T_i^k = \min\Big\{ t_i/p_i + (1/p_i - 1)K_i + s_i + \min_{\substack{j1, j2 \geq 0 \\ 0 \leq j1 + j2 \leq k-1}} \left[T_{il}^{j1} + T_{ir}^{j2}\right], \; t_i/p_i + (1/p_i - 1)K_i + \min_{\substack{j1, j2 \geq 0 \\ 0 \leq j1 + j2 \leq k}} \left[T_{il}^{j1} + T_{ir}^{j2}\right] \Big\}$$

where $\eta(i)$ is the number of nodes in the subtree rooted at i, and K_i's are computed using (3a) and (4a). According to the aforementioned, the solution to a given problem instance with, say, v recovery points would be T_{root}^v.

Consider two sets of solutions for the subtrees of i: an optimal solution (T_{il} and T_{ir}) and any other (not necessarily optimal) solution (T'_{il} and T'_{ir}). Notice that, while computing T_i, in the first case (i.e., when there is an RP before i), K_i is a function of i, whereas in the other, it depends on K_{il} and K_{ir}. While computing T_i using (4), if the corresponding solutions to the subtrees are such that K'_i is less than K_i, for the choice of no RP at i, it is possible that suboptimal solutions for the subtrees (i.e., T') give rise to a smaller T_i. Based on this observation, one can conclude that the *principle of optimality* [5] does not apply for this formulation. In the next section, we discuss this aspect of the problem in details.

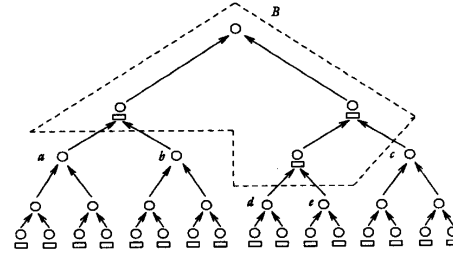

Fig. 2. Partitioning scheme that leads to exponential time complexity (optimal solutions to subtrees rooted at a, b, c, d, and e can be combined with the solution for B).

One way to avoid the possibility of not considering such suboptimal solutions is to partition the problem as follows (see Fig. 2). A recovery point is associated with *each* leaf node in one of the subtrees, say B, and those are the *only* recovery points in B. In such a case, none of the suboptimal solutions (see above) to the *rest* of the subtrees need to be taken into consideration while constructing the solution to the problem. This is because a recovery from failure of any node in B does not require the execution of any nodes of the *other* subtrees (because of the RP's at the boundary). Such a partitioning scheme has been used in [3]. However, that scheme requires an exponentially large number of combinations to be evaluated, leading to a computationally expensive solution.

IV. Proposed Partitioning Technique

Interestingly, a solution with a polynomial time complexity can be formulated, as discussed in the following. We observe that T_i in (3b) is optimized when the solutions for the subtrees, i.e., T_{il} and T_{ir}, correspond to the solutions with least value of T among all possible solutions. On the other hand, to obtain the optimal solution when using (4b), we must use those solutions to the subtrees that contribute the least after the effects of both T and K are considered. This becomes apparent when we rewrite (4b) as the following:

$$T_i = t_i/p_i + T_{il} + (1/p_i - 1)(t_{il}/p_{il} + K_{il}/p_{il}) + T_{ir} + (1/p_i - 1)(t_{ir}/p_{ir} + K_{ir}/p_{ir}). \tag{5}$$

S0. $hi \leftarrow$ Height of the tree
S1. Initialize: Compute K_i and T_i for leaf nodes $((N/2)+1 \leqslant i \leqslant N)$
S2. Compute K_i and T_i for internal nodes $(1 \leqslant i \leqslant N/2)$
 Let d be the depth[2] of node i.
 Let B_i be the subtree rooted at node i.
 Let β be the number of leaf nodes in B_i.
 Let γ be the number of nodes in B_i.
 S2a. /* When the number of RP's is less than β */
 for $0 \leqslant j < \beta$ **do** $(0 \leqslant j < 2^{hi-d})$
  Set $K_{i,j}$ and $T_{i,j}$ to ∞.
 S2b. /* When the number of RP's is between β and γ */
 for $\beta \leqslant j \leqslant \gamma$ **do** $(2^{hi-d} \leqslant j < 2^{hi-d+1})$
  for $k = 0$ (No RP before i) or $k = 1$ (RP before i) **do**
   for l RP's allocated to the left subtree of $B_i (0 \leqslant l \leqslant \min(j-k, 2^{hi-d}))$
    Compute $K_{i,j}$ and $T_{i,j}$ using both solutions (with minimal T and minimal f) for the left and right subtrees of B_i.
    Save the two sets of values: those with smaller T_i as $T_{i,j,1}$, and those with smaller f as $T_{i,j,0}$ and the corresponding $K_{i,j}$ values.
 S2c. /* When the number of RP's is between γ and v */
 for $\gamma < j \leqslant v$ **do** $(2^{hi-d+1} \leqslant j \leqslant v)$
  Set $K_{i,j}$ and $T_{i,j}$ to $K_{i,\gamma}$ and $T_{i,\gamma}$ respectively.

Outline of Algorithm *rpassign*.

By grouping these terms appropriately, T_i can be expressed as the following:

$$T_i = t_i/p_i + f_{il} + f_{ir} \tag{6}$$

where

$$f_j = T_j + (1/p_i - 1)(t_j/p_j + K_j/p_j), \qquad j = il \text{ or } ir. \tag{7}$$

Observe that in (6) the three terms which constitute T_i depend, respectively, on i (root of the subtree), the solutions to its left subtree and the solution to its right subtree. Thus an optimal solution to the subtree rooted at i must consist of such solutions to its subtrees that the sum of f_{il} and f_{ir} is a minimum.

Therefore solutions with minimal value of f must be used to construct the solution at a higher level. This will ensure that the solution computed for the parent node is optimal. Such a scheme will require, for every subtree, computation of the solution with a least value of f along with the solution with minimal T.

We thus arrive at the following proposition which forms the basis of the *principle of optimality.*

Proposition 4.1: In an optimal solution, the solutions to the two subtrees must either correspond to a minimal value of function f or be optimal (i.e., with minimal expected computation time).

A bottom-up[1] formulation corresponding to (5) can now be stated as: let $T^k_{i,0}$ be the expected computation time for the subtree rooted at i, when f is a minimum, and $T^k_{i,1}$ be the optimal expected time (i.e., with least T), with a maximum of $k < \eta(i)$ recovery points allowed, where $\eta(i)$ is the number of nodes in the subtree rooted at i. The set of solutions that must be taken into consideration are

$$\{t_i/p_i + s_i + (1/p_i - 1)r_i + [T^{j1}_{il,z1} + T^{j2}_{ir,z2}]\} \qquad j1, j2 \geqslant 0, \qquad 0 \leqslant j1 + j2 \leqslant k-1$$

and

$$\{t_i/p_i + [T^{j1}_{il,z1} + T^{j2}_{ir,z2} + (1/p_i - 1) \cdot (t_{il}/p_{il} + K^{j1}_{il,z1}/p_{il} + t_{ir}/p_{ir} + K^{j2}_{ir,z2}/p_{ir})]\} \qquad j1, j2 \geqslant 0, \qquad 0 \leqslant j1 + j2 \leqslant k$$

where $z1, z2 \in \{0, 1\}$.

[1] This is not a pure bottom-up formulation since one of the expressions that must be computed for a subtree requires information about the parent node (p_i in (7)).

To restate, at each node (except the root) two solutions are selected: $T^k_{i,0}$ is that solution for which function f is a minimum and $T^k_{i,1}$ is the solution yielding a minimal value. And the solution we are seeking is $T^v_{root,1}$.

V. Algorithm *rpassign* and Its Time Complexity

An outline of algorithm *rpassign,* which is employed to compute the optimal expected computation times, is given above. Without loss of generality, we assume a *complete* binary tree model in the discussions in this section. The notations used are: 1) N: the number of nodes in a complete binary tree; 2) v: maximum allowed number of recovery points; and 3) $K_{i,j,z}$ and $T_{i,j,z}$: expected value of K_i and T_i, given a maximum of j recovery points, with $z = 1$ being the value when T_i is minimal, and $z = 0$ being the value when f is minimal.

In the outline, S1 requires $v \cdot (N+1)/2$ steps. S2a requires 2^{hi-d} steps, and S2c requires $v - 2^{hi-d+1} + 1$ steps. S2b requires $(2^{hi-d+1} - 2^{hi-d}) \cdot 2 \cdot 2^{hi-d}$ steps corresponding to the ranges of j, k, and l (assuming that the computation within the *for* loops take constant time). Since v can, at most, be $N-1$, i.e., $2^{hi+1}-1$, and number of nodes at depth d is 2^d, the time complexity can be expressed as

$$T(N) = O(N^2) + \sum_{d=0}^{hi} 2^d [2^{hi-d} + (2^{hi+1} - 2^{hi-d+1}) + (2^{hi-d+1} - 2^{hi-d}) \cdot 2 \cdot 2^{hi-d}]. \tag{8}$$

Dividing throughout by 2^{hi+1}, we get:

$$= O(N^2) + 2^{hi+1} \sum_{d=0}^{hi} [1/2 + (2^d - 1) + 2^{hi-d}] \tag{9}$$

$$= O(N^2) + 2^{hi+1} \sum_{d=0}^{hi} [2^d - 1/2 + 2^{hi-d}] \tag{10}$$

$$= O(N^2) + 2^{hi+1} \cdot [2^{hi+1} - 1 - (hi+1)/2 + 2^{hi} \cdot 2 \cdot (1 - 1/2^{hi+1})]. \tag{11}$$

[2] Reference depth is zero for *root.*

This reduces to (2^{hi+1} replaced by N):

$$T(N) = O(N^2) + N(N-1) - \frac{hi+1}{2}N + N(N-1). \quad (12)$$

Clearly, the previous expression is $O(N^2)$ and thus we have formulated a polynomial time algorithm. In terms of space complexity, the algorithm would require a table each of whose entries holds two values. Also, the bookkeeping is slightly higher, because values of both K and T must be stored for each node. Although this means a higher price in terms of memory, it is possible to solve the problem in polynomial time. In any case, the space requirement is also a polynomial function of N. Particularly, it is in the exact order of $4 \cdot N \cdot N$.

A complete form of the algorithm along with a more detailed analysis of the time complexity is included in [7].

VI. Solution for M-ary Tree Model

In order to extend the aforementioned solution to m-ary trees, we must redefine K_i, E_i, and T_i as follows:

1) if there is a recovery point before task i,

$$K_i = r_i \quad (13a)$$

$$E_i = t_i/p_i + (1/p_i - 1)K_i \quad (13b)$$

$$T_i = T_{i1} + T_{i2} + \cdots + T_{im} + E_i + s_i \quad (13c)$$

2) otherwise

$$K_i = K_{i1} + E_{i1} + K_{i2} + E_{i2} + \cdots + K_{im} + E_{im} \quad (14a)$$

$$E_i = t_i/p_i + (1/p_i - 1)K_i \quad (14b)$$

$$T_i = T_{i1} + T_{i2} + \cdots + T_{im} + E_i. \quad (14c)$$

Without specifying the details of the algorithm and analysis, we state that an algorithm designed along exactly the same lines as ***rpassign*** would have a time complexity given by

$$T(N) = O(N^2) + \sum_{d=0}^{hi} m^d [m^{hi-d} + (m^{hi+1} - m^{hi-d+1}) + (m^{hi-d+1} - m^{hi-d}) \cdot 2 \cdot \left(m^{hi-d}\right)^{m-1}] \quad (15)$$

corresponding to (8) above. Notice that this is a general expression and (8) is the case for a binary tree which is obtained by substituting 2 for m. This expression reduces to $O(N^m)$, which again is polynomial. However, as might be expected, the algorithm has a growth rate which increases with the arity of the tree.

VII. Conclusions

In this note we have formulated a solution for the problem of assigning recovery points in a task system modeled by a reverse binary tree. We have also shown that the corresponding algorithm has polynomial time complexity, particularly $O(N^2)$. While the complexity results have been based on a complete tree, it is easy to extend them to the general case (i.e., where a subset of nodes are absent) without compromising with the complexity of the algorithm. The results have been extended to the case of an m-ary tree, for an arbitrary m, where the time complexity increases to $O(N^m)$.

Although we have restricted our discussions to concurrent execution of tasks in a uniprocessor system, the results can be extended to a task model in multiprocessor environment that is similar to those proposed in [3], [4], and [6].

Minimizing the computation time is chosen as our objective because for many applications, the system response time (which is related to the computation time) tends to be of primary concern. Under the real-time environment, satisfying recovery time constraints may be an additional goal, which has not been taken into consideration in this work.

Acknowledgment

The authors wish to thank the anonymous referees whose comments helped improve the presentation of this paper.

References

[1] K. M. Chandy, "A survey of analytical models of roll back and recovery strategies," *IEEE Trans. Comput.*, vol. 8, pp. 40–47, May 1975.
[2] K. M. Chandy and C. V. Ramamoorthy, "Rollback and recovery strategies for computer programs," *IEEE Trans. Comput.*, vol. C-21, pp. 546–556, June 1972.
[3] S.K. Chen, W.T. Tsai, and M.B. Thuraisingham, "Recovery point selection on a reverse binary tree task model," *IEEE Trans. Software Eng.*, vol. SE-15, pp. 963–975, Aug. 1989.
[4] M. Gransky, I. Koren, and G.M. Silberman, "The effect of operation scheduling on the performance of a data flow computer," *IEEE Trans. Comput.*, vol. C-36, pp. 1019–1029, Sept. 1987.
[5] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* Rockville, MD: Computer Sci. Press, 1978.
[6] B. Indurkhya, H.S. Stone, and L. Xi-Cheng, "Optimal partitioning of randomly generated distributed programs," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 483–495, Mar. 1986.
[7] S.K. Mishra, V. Raghavan, and N. Tzeng, "Efficient algorithms for selection of recovery points in tree task models," Center for Advanced Computer Studies, Univ. Southwestern Louisiana, Lafayette, LA, Tech. Rep. TR 90-5-6, 1990.