Korean Management
Science Review
Vol.11, No.1, February 1994.

정보시스템 관리에 있어서 최적 배치 조정 정책†

김기수*

Optimal Policies for Batch Control Operations in Information Systems†

Kisu Kim*

ABSTRACT

For an Information System to be successful the continual control of the database system (DBS) is very important. In general, such control operations are performed periodically in batches, even in real time systems. We explain DBS related such control operations and describe the decision problem in each of them. Specifically, backup, checkpointing, reorganization, and batch update operations are considered. Then we develop a general model of the batch backup situation to determine the optimal backup timing. Other operations are considered as special cases of the general batch backup operation. Under two common control policies, the optimal timings of the batch backup operation are derived and compared. We show that, in general, the control limit policy is superior to the fixed time interval policy in terms of the long-run average cost per unit time. Some practical considerations about the implementation of optimal policies are also mentioned.

1. INTRODUCTION

Information systems (IS) management in modern enterprises employing the database techniques has evolved into a critical function encompassing computer, management, and systems sciences. An extensive literature has been developed by researchers and practioners in the planning, analysis, design, and implementation of a database system (DBS). However, the backbone of successful

† This paper was supported by NON DIRECTED RESEARCH FUND, Korea Research Foundation, 1992.
* Dept. of Business Education, Yeungnam University

IS management over time is the continual control of the DBS after its implementation. Relatively few attempts has been made to this control phase of IS management.

In general, IS deteriorate with time unless control actions are taken continuously. However, it is almost impossible to take control actions on IS continuously because of the time and cost consumed for them. Therefore, many IS control operations are performed periodically in batches, even in real time systems. A common decision IS managers must make is how often to perform batch control operations in order to balance the costs of batch operation against the costs of IS deterioration. Generally, the shorter the time between batch operations is, the better the efficiency or the performance of IS is. But the batch operations themselves consume system time and resources. Convesely, less frequent batch operations decrease the costs of control operations, but incur more costs for IS degradation. The purpose of this paper is to present a general model for determining the optimal frequency (or timing) of these batch control operations.

Although there may be many different batch control operations in the IS management, we concentrate on the control operations related to DBS, one of the most important part of modern IS. Specifically, batch backup, checkpointing, batch update, and database reorganization operations are considered. This paper introduces such DBS related batch control operations and decision issues related with them. Then, we develop a general model to determine optimal timings for those batch control operations. There are two commonly used control policies. One is a fixed time interval policy and the other is a control limit policy. In a fixed time interval policy, control operations are taken at fixed time intervals regardless of the status of the system state. On the other hand, a control limit policy calls for control operations whenever the system state reaches a certain limit. If the system state changes deterministically two policies are exactly the same. However, if the system state changes stochastically two policies may result in different effects on the system management. Intuitively, it seems to us that a control limit policy is more effective than a fixed time interval policy in a system which changes stochatically with wide variation, because the former utilizes more information about the system state than the letter does. However, implementing a control limit policy requires continuous observation of the system state which may not be possible for some systems. On the other hand, a fixed time interval policy is easy to implement because it does not require continuous monitoring of the system state. We will consider both policies for comparison purpose and derive the optimal batch control timings under these policies. It is shown that if the system state could be observed continuously the optimal control limit policy is superior to the optimal fixed time interval policy.

The structure of this paper is as follows. In Section 2, we introduce different DBS related batch control operations and describe their managerial problems. The model is presented in Section 3. In Section 4, we analyze the model and derive optimal policies. The optimal control limit policy and the optimal fixed time interval policy are derived in Section 4.1 and Section 4.2, respectively. Section 4.3 contains the comparision of two optimal policies and nemerical examples. We conclude in Section 5 with summary and some practical considerations.

2. Problem Description

In this Section, we describe several DBS related bach control operations and their managerial problems, with previous related works. This Section provides us the backgound and the necessity of our research.

2.1. Batch backup and checkpointing

The need for highly reliable DBS, such as banking transaction-processing systems and space-related process control systems, is rapidly increasing. In general, high reliability may be achieved by using highly reliable components, adding hardware redundancy, or providing software techniques for failure recovery. In this paper, we are interested in the performance of such software techniques. Failures in DBS can be divided into three classes : transaction failures, system failures, and media failures. A transaction failure occurs when, for example, an attempt to access privileged data is made. In such case the recovery manager of the DBS forces the transaction to terminate. A system failure, like a power failure or system crash, causes the DBS to be unavailable for a period of time and loss of the contents of main memory, after which the system has to be restarted. The third type of failure, a media failure, occurs when data stored on a disk volume is lost. Fires and head crashes are examples of such a failure [15]. In this paper, we focus on system and media failures.

Different database recovery techniques are available for different types of failures [3]. Three common recovery techniques which cope with system and media failures are incremental dumping, audit trails, and backup/current versions. Incremental dumping is a recovery technique against media failures. It involves a periodic copying of database to the archival storage (backup copies of files). In order to maintain the consistency of the database, transactions are not processed during dumping. Once the dump is taken, all processed transactions are then dated and recorded on a transaction log (audit trail). The audit trail recovery technique records a sequence of actions on the database. This technique allows both the restoration of the state of the system to the correct state prior to a crash and the backing out of transactions. The backup/current versions recovery technique keeps two copies of active files. One copy contains the current values and thus is called the current version. The other copy is a backup version and contains previous values. When the system crashes, the backup version is used to restore a file to its previous values. A combination of backup/current versions and audit trail techniques can be used to restore the system to the correct state that preceded a system failures. Similarly, incremental dumping with audit trail can handle media failures. This combination of techniques are refered as rollback and recovery. After a failure, the system rolls back to a previous state which was stored as a backup version or as a previous dump, depending on the type of failure, and then recovers by processing the audit trail.

The rollback and recovery technique—as a tool for dealing with system failure—periodically saves the pages in main memory on disks and records all activities on a reliable log tape (audit trail) that never fails ; this can be approximated by retaining two or more copies simultaneously. The operation of saving the system state on disk is called checkpointing. When the main memory fails, the system recovers by using the log tape and the copy saved at the most recent checkpoint to bring the system to the correct state that preceded the failure. This process is called error recovery, and consists of loading the most recent copy from disk to memory and then reprocessing all the activities, stored on the log tape, that took place since the most recent checkpoint and prior to failure. During checkpointing and error recovery, the system is not available to users.

Media failures may be handled similarly, except that the two relevant levels in the memory hierarchy are disk and archival storage rather than main memory and disk. This database recovery procedure involves a periodic dumping of the database to a backup file. Once the dump is taken, all processed transactions are then dated and recorded on a transaction log, so that in the event of a data loss the latest dump may be recopied and all transactions reapplied. Recopying the latest dump is generally quite fast, requiring as little as a few minutes for an entire disk pack. The reprocessing of transactions, however, may require several hours if the time since the dump has been long. Database recovery cost can be reduced considerably with the use of an after-image log, in which a sequential file is used to store an identified and dated copy of each new or changed database page, record, or record segement at the time that it is modified. With this file, transaction reprocessing is not required once the dump is restored. Rather, the log is sorted by identifier and date, and the latest version of each modified record is selected and written directly into the database [1].

In the rollback and recovery procedures discussed above, an important problem is the determination of when to save the system state (the pages in main memory) on disk or to dump the database to a backup file. Frequent savings or dumpings shorten any necessary recovery time when the system fails, but themselves consume time and resources. Lohman and Muckstadt [9] studied the trade-offs between dumping costs and recovery costs by means of an analytic model that provides guidence in selecting an efficient dumping frequency. Recently, the problem of placing the checkpoints "optimally" in time-optimal checkpointing strategies—has received considerable attention [2,5,7,15]. The most frequently used objectives are to maximize system availability or to minimize the mean response time per transaction. The models differ in their assumptions regarding the distribution of interfailure time, checkpointing time, and recovery time, whether the failure is detected instantaneously, and weather failures may occur during checkpointing and error recovery. Shin et al. [13] applied the same basic concept of checkpointing in DBS to real−time tasks. In previous studies, it has shown that checkpointing can greatly reduce the mean time of running a long program on an unreliable computing system [4,8]. Checkpointing a real−time task means occassionally saving the state of the task on other save devices such as tapes, disks, or even other (redundant) memory modules. The state of a task includes values of data variables and contents of the internal registers. When a module fails and the failure is detected either by the acceptance test or the on−line detection mechanism, the most recently saved checkpoint for the task running on this module will be loaded to a good module, and the task then resumes execution from that checkpoint. Most of recent works on checkpointing mentioned above concern system availability or the mean response time as the efficiency measure. By contrast, this paper considers different costs associated with management of DBS and the objective is minimizing the expected long−run average total cost of batch control strategies. In this sense, this paper is similar to [9]. However, this paper models with more generalized update arrival process and cost function, and also applies two different control policies and compares them.

2.2. Database Reorganization and Batch Updating

The issue of database reorganization has been surveyed by Sockut and Goldberg [14]. This survey introduces the basic concepts of database reorganizaton, including classification of reorganization types and why it is performed. It also includes database administration considerations in connection with reorganization such as recognizing the need to reorganize, deciding what new structures are to result from reorganization, deciding when to perform reorganization, etc. They introduced several research efforts which have modeled database performance deterioration, improvement through maintenance, and maintenance costs. As time goes on, access time and cost increase owing to both file growth and deterioration of storage structures. At some point, the maintenance reorganization should be performed to reduce operating costs. Research efforts include policies for deciding when to perform maintenance so as to minimize the total cost of access and maintenance over a period of time [14].

The problem of finding optimal maintenance reorganization policies was addressed by several authors. Mendelson and Yechiali [10] assumed records addition to the file follow a renewal process and the number of records added to the file system is the deterioration index. They considered both policies of state-dependent reorganization and reorganization at fixed time intervals. They showed that the optimal state-dependent policy belongs to the class of control-limit policy. They also showed that fixed time interval policy is inferior to the optimal control-limit policy when the record arrival process is Poisson. Heyman [6] also modeled database degradation which results from changes on the initial physical structure of a database. He assumed that transactions are statistically independent and either add, delete, or update data. He cosidered two models. The first model examines the time during which a block of data is filling up. The second model examines the overflows from a block of data, which essentially describes the buildup of disorganization. Analytical results using random-walk model and diffusion process model were obtained for the time until a block of data is initially filled up and the time between subsequent overflows. These results were used to derive an optimal time interval to reorganize a stochastically growing database.

An additional case which illustrates the need for a reorganization and the tradeoff involved is the differential file, which was introduced by Severance and Lohman [12]. The representation of data in terms of difference from a preestablished point of reference is data compaction technique with wide applicability. They described a differential database representation which is shown to be an efficient method for storing a large and changing database. In the representation, all database modifications (updates) are localized into a relatively small storage area, called differential file. This technique can reduce database update costs significantly at a cost of increased access time. When the differential file grows sufficiently large, a reorganization incorporates all the updates into the main database file, and the new differential file is empty and ready to accumulate updates [12]. By utilizing differential files, it is also possible to reduce backup costs, speed the process of database recovery, and even minize the probability of a serious data loss. Aghili and Severance [1] studied their value in reducing backup and recovery costs for on-line databases. For large databases with moderate or naturally concentrated update activity, differential files offer an alternative strategy for reducing backup and recovery costs. A differential file isolates a database from the physical change by directing all new and modified records onto a separate and relatively small file of changes. Since the main file is never changed, it can always be recovered quickly from its dump in the event of a loss. Transaction reprocessing is required only in the event of damage to the differential file. Since this file is small, it can be backed up quickly and frequently to reduce reprocessing costs. It can also be duplexed at reasonable cost as insurance against physical damage to one of the copies [1].

3. Modeling

Grossary of Notation

b	variable backup cost per update
B	fixed backup cost
R	cost of reloading backup file
a	cost of restoring one update from the transaction log during roll forward (recovery)
s	cost of storing one update in the trasaction log per unit time
λ	failure occurrance rate (mean time between failures is $1/\lambda$)
μ	transaction arrival rate (mean time between transactions is $1/\mu$)
σ	inverse of mean number of updates per transaction (mean number of updates per transaction is $1/\sigma$)
X_i	time interval between (i−1)st and ith transaction arrivals
T_n	time until nth transaction arrives ($=\sum_{i=1}^{n} X_i$)
M_n	number of failures occurred during T_n
K_i	number of failures occured during X_i
$N(t)$	number of transactions arrived until time t
$Y(t)$	number of updates accumulated until time t without backup operation
$V(t)$	number of updates accumulated until time t under the control limit policy
B_n	backup cost during a cycle when the control limit is n transactions
S_n	storage cost durig a cycle when the control limit is n transactions
R_n	recovery cost during a cycle when the control limit is n transactions
C_n	total cost during a cycle when the control limit is n transactions
$C(n)$	expected long−run average cost per unit time under the control limit policy when the control limit is n transactions
$S(t)$	storage cost until time t under the fixed time interval policy
$R(t)$	recovery cost until time t under the fixed time interval policy
$C(t)$	expected total cost per unit time when the backup operation is taken in every t time units

We present recovery procedure for the media failure (batch backup) as a base model and treat the rest operations as special cases. Consider a DBS in which transactions arrive stochastically and each transaction has random number of updates. Assume that times between transaction arrivals and the numbers of updates in each transaction are independent and identically distributed. Let $(T_1, T_2, \cdots)$ be the sequence of time intervals between transaction arrivals and $(Z_1, Z_2, \cdots)$ be the sequence of number of updates in each transaction. Then, the number of updates made in the DBS until time t can be expressed as

$$Y(t) = \sum_{i=1}^{N(t)} Z_i$$

where $N(t)$ is the number of transactions occurred until time t. T_i's and Z_i's are independent and identically distributed random variables with means of $1/\mu$ and $1/\sigma$, respectively, and Z_i's are also independent of T_i's. Failures occur according to a Poisson process having rate λ, which means that the times between failures are independent and identical exponential distributions with expected value of $1/\lambda$. Failure process is assumed to be independent of transaction arrival process, $\{Y(t), t \geq 0\}$.

When a failure occurs the latest backup is first reloaded to the destoryed database area, then all the updates made since the last backup and prior to the failure is restored from either the transaction log or the after-image log. We assuem that costs of a recovery process are sum of costs of reloading the backup file and costs of restoring updates made since the last backup. Costs of reloading the backup file are assumed to be fixed R, and costs of restoreing updates during a recovery process are assumed to be a linear function of the number of updates to be restored, i. e. ax, where a is a positive constant which is costs of restoring each update and x is the number of updates to be restored. Storing (Recording) each update in transaction log or after-image log is assumed to cost fixed s per unit time. We also assume that each backup (dump) operation costs a fixed plus linear variable cost, i.e. $B + bn$, where b is a positive constant and n is the number of updates made since the last backup.

We assume that times required to process update transactions, to perform a backup operation, to reload backup file following a failure, and to restore each update during roll forward are negligible compared to the time between batch backup operations, which is true for most of practical cases. Although these times may not be negligible in some special cases, we could consider that these times are converted to the costs and included in the costs of these works. This assumption is also useful when we compare two control policies.

Although we talked only about the base model of batch backup operation so far, which is the most general case, the other operations can easily be represented by this model by slightly changing and redefining parameters and operations. For checkpointing operation, the rollback and recovery procedure periodically saves the pages in main memory on disks, which is called checkpointing, and records all activities on a reliable log tape (audit trail). When the main memory fails, the system recovers by using the log tape and the copy saved at the most recent checkpoint to bring the system to the correct state that preceded the failure. Therefore, checkpointing operation which is a control operation for system failures can be handled by our base model, except that the two relevant levels in the memory hierarchy are main memory and disk rather than disk and archival storage.

For DBS reorganization operation the number of updates could be considered as the number of records added to the database file. Records are added to the database file time to time whenever a transaction occurs. The system state at time t, $Y(t)$, is now the number of records added to the database file until time t. While the system state is k, records may also be retrived, updated or logically deleted. Logically deleted records are not removed from the file until the system is reorganized. It follows that under this practice of no space recovery, deletions do not change the number of stored records and hence do not improve DBS efficiency. In some cases, the number of records added to the database file is not an appropriate deterioration index. In fact, several file systems dynamically recover the space which was occupied by deleted records, and attempt to reuse it when possible. In this case, the role of a reorganization is to compact the file and make this space contiguously reusable. Obviously, the model does not change if there is some other deterioration index, for example the number of updates in differential file, or the number of records in an overflow area. The increase in file occupancy results in performance deteration and increased processing cost. A reorganization enables the database administrator to restore processing efficiency and reduce the subsequent operating costs. However, a reorganization requires extra expenditures to unload and reload the files, for system unavailability during the reoganization time, etc. The problem, therefore, is to establish the optimal tradeoff between efficient system operation and reorganization costs. Failures and recovery process in the base model are irrelavant in the database reorganization problem and the storage cost, S_n, can be interpreted as operating costs of DBS (costs associated with performance deterioration and increased processing costs due to the increase in database file occupancy) while in state n. The backup operation costs, B_n, is replaced by reorganization costs.

In batch updating, updates or changes in the database file are accumulated in a relatively small storage area called a differential file over a period of time and are posted en masse when creating a new database edition (generation). It is more expensive, as measured in terms of storage costs, maintenance time, and overall system complexity, to directly modify the database with each update transaction. As a compromise, a differencial file can be used like an errata list for a book to identify and collect pending record changes. Consulting the differential file as a first step in data retrieval effectively yields an up-to-date database. At a cost of increased access time, database update costs may be reduced. When the differential file grows sufficiently large, a reorganization (batch update operation) incorporates all changes into a new generation of the database, and the now empty differential file can begin accumulating changes anew. The problem is how often to implement batch update operation to balance the costs of an update operation and the costs associated with increased access time and/or other operating costs. Again, failures and recovery process are irrelavant and the storage costs and the costs of backup operation are reinterpreted similarly as in reorganization problem.

To complete our model we need to decide the type of control policy we are going to apply. Two policies are considered, i.e. the control limit policy and the fixed time interval policy. The control limit policy is a control policy that a control action is triggered whenever the system state reaches or exceeds a fixed level. In general, the control limit is expressed in terms of the system state, which is the number of updates accumulated in a DBS for our model. However, the number of accumulated updates in our model increases only when a transaction arrives and it reaches or exceeds a certain level at the time point when a transaction arrives. Therefore, the control limit could be expressed in terms of the number of transactions arrived in stead of the number of accumulated updates. Of course, if the number of updates in each transaction is just one, the two (no. of transactions and no. of updates) are identical. In addition, counting the number of transactions arrived is easier than counting the number of accumulated updates and the computation of the expected long-run average cost per unit time could be simpler when the control limit is expressed in terms of the number of transactions arrived. In this paper, we assume a control limit policy that takes a control action (for example, backup operation) whenever the number of transactions arrived since the last control action becomes a certain fixed level. On the other hand, in the fixed time interval policy control actions are taken at fixed time intervals regardless of the actual system state. While the implementation of the control limit policy requires a continuous monitor of the system state (counting transactions in our model), the implementation of the fixed time interval policy needs just to monitor a real-time clock. In our case, both works can easily be done by softwares such as DBMS or operating system.

Under the control limit policy, the times between backup operations form a renewal sequence and the updates accumulation process under the control-limit policy, $\{V(t),\ t \geq 0\}$, is a regenerative process with respect to this renewal sequence. Therefore, the expected long-run average cost per unit time is the expected cost of a cycle divided by the expected length of a cycle by the Renewal Reward Theorem [see, for example, [11], p.236-237]. Since the backup operation is taken whenever the number of transactions arrived since the last backup is a certain fixed level, n, a cycle in our model is the time until the nth transaction arrives. In the next Section, the expected long-run average cost per unit time is derived and the optimal value of n minimizing it is computed.

Under the fixed time interval policy, the times at which control actions are taken are not, in general, regeneration points since the time between transaction arrivals is assumed to be a general distribution. Therefore, in general, the expected long-run average cost per unit time cannot be computed as the case when the control limit policy is used. Of course, if the time between transaction arrivals is a exponential distribution, the times at which the control (backup) operations are taken are regeneration points. The expected total cost per unit time is computed as a function of time t and the optimal time interval minimizing it is derived in the next Section.

4. Optimal Policies

In this Section, we present the expected cost per unit time for each of two control policies and derive the optimal policies that minimize the expected cost per unit time. Two policies are compared and the difference is also obtained.

4.1. Optimal control limit policy

The objective is to find the value of n, the number of transactions, that minimizes the expected long−run average cost per unit time for the control limit policy. The costs considered here are the backup, recovery, and updates storage (in the transaction log or after−image log) costs. For the model described in the previous Section, the expected long−run average cost per unit time c.n be expressed as the expected cost of a cycle divided by the expected length of a cycle, $E[C_n]/E[T_n]$, by the Renewal Reward Theorem. T_n is just the time until the nth transaction arrives and $E[T_n] = n/\lambda$. Computing $E[C_n]$ is more complicated. $E[C_n]$ is the sum of the expected updates storage costs, the expected recovery costs, and the expected backup costs when the backup operation is taken just after the nth transaction arrives. Each of these can be expressed as follows.

Expected backup cost $= E[B_n] = B + (n/\sigma)b$

Expected updates storage cost $= E[S_n] = s \cdot E[\sum_{i=2}^{n} X_i \sum_{j=1}^{i-1} Z_j]$

$$= s \sum_{i=2}^{n} EX_i \sum_{j=1}^{i-1} EZ_j$$

$$= s \sum_{i=2}^{n} 1/\mu \sum_{j=1}^{i-1} 1/\sigma$$

$$= s\,(1/\mu\sigma) \sum_{i=2}^{n} (i-1)$$

$$= sn(n-1)/2\mu\sigma$$

We may express the expected recovery cost as

$$E[R_n] = EE[R_n | M_n = m]$$
$$= \sum_{m=0}^{\infty} E[R_n | M_n = m]\ P\{M_n = m\}.$$

Additionally, we know from the assumptions in Section 3 that

$$P\{M_n = m\} = \int_0^\infty P\{M_n = m | T_n = t\}\ dF_r(t)$$
$$= \int_0^\infty ((\lambda t)^m e^{-\lambda t}/m!) dF_r(t),$$

where $F_r(t)$ represents the cumulative distribution of T_n. But

$$E[R_n | M_n = m]$$
$$= mR + a \cdot (0 \cdot E[K_1 | M_n = m] + (1/\sigma) \cdot E[K_2 | M_n = m] + \cdots$$
$$+ (n-1)(1/\sigma) \cdot E[K_n | M_n = m]).$$

Since the failure process is a homogeneous Poisson process, if we know that m failures occurred during a backup cycle, the set of m failure occurrence times $\{F_1, F_2, \cdots, F_m\}$ has the same distribution as a set of m random variables which are independent and uniformly distributed over a backup cycle [11, Theorem 3.2]. It can be recognized that the vector $(K_1, K_2, \cdots, K_n)$ is multinormially distributed, with parameters m, $p_1, \cdots, p_n$, given that

$\sum_{i=1}^{n} K_i = m$. Furthermore, $p_i = E[X_i]/E[T_n]$, assuming $E[X_i] < \infty$.

Consequently, $E[K_i | M_n = m] = m \cdot E[X_i]/E[T_n]$.

Therefore,

$$E[R_n | M_n = m] = mR + (a/\sigma)(m\frac{E[X_2]}{E[T_n]} + 2m\frac{E[X_3]}{E[T_n]} + \cdots + (n-1)m\frac{E[X_n]}{E[T_n]})$$
$$= mR + (a/\sigma)\sum_{i=1}^{n-1} mi\frac{E[X_i]}{E[T_n]}$$
$$= mR + m(a/\sigma)\frac{1/\mu}{n/\mu}\ \frac{n(n-1)}{2}$$
$$= mR + \frac{ma(n-1)}{2\sigma}$$

As the result, the expected recovery cost per cycle could be computed as follows.

$$\begin{aligned}
E[R_n] &= \left(R + \frac{a(n-1)}{2\sigma}\right) \sum_{m=0}^{\infty} m\int_0^{\infty} ((\lambda t)^m e^{-\lambda t}/m!)dF_r(t) \\
&= \left(R + \frac{a(n-1)}{2\sigma}\right) \int_0^{\infty} \sum_{m=0}^{\infty} m((\lambda t)^m e^{-\lambda t}/m!)dF_r(t) \\
&= \left(R + \frac{a(n-1)}{2\sigma}\right) \lambda\int_0^{\infty} t dF_r(t) \\
&= \left(R + \frac{a(n-1)}{2\sigma}\right) \lambda E[T_n] \\
&= \left(R + \frac{a(n-1)}{2\sigma}\right) \lambda(n/\mu) \\
&= \frac{R\lambda}{\mu} n + \frac{a\lambda}{2\sigma\mu} n(n-1)
\end{aligned}$$

The operations of integration and summation may be interchanged since the integral is over a product of two functions, each continuous in t and bounded uniformly by 1. Observe that $E[R_n]$ does not depend on the form of the distribution of T_n, $F_r(t)$, but only the mean of the distribution. Consequently $E[C_n]$ also depends on $E[T_n]$ and not on $F_r(t)$. The importance of this observation is that the X_i can have any distribution as far as the mean of the distribution is known.

Therefore, the expected total cost for a backup cycle is give by

$$E[C_n] = B + \frac{b}{\mu} n + \frac{s}{2\sigma\mu} n(n-1) + \frac{R\lambda}{\mu} n + \frac{a\lambda}{2\sigma\mu} n(n-1).$$

The expected long-run average cost per unit time is computed as follows.

$$\begin{aligned}
C(n) &= \frac{E[C_n]}{E[T_n]} \\
&= \frac{\mu B}{n} + \frac{\mu b}{\sigma} + \frac{s}{2\sigma}(n-1) + \lambda R + \frac{\lambda a}{2\sigma}(n-1) \\
&= \frac{\mu B}{n} + \frac{\lambda a + s}{2\sigma}(n-1) + \lambda R + \frac{\mu b}{\sigma} \qquad (1)
\end{aligned}$$

Taking the derivative of C(n) with respect to n yields

$$\frac{dC(n)}{dn} = C'(n) = \frac{\mu B}{n^2} + \frac{\lambda a + s}{2\sigma} \qquad (2)$$

Since $C''(n) = \frac{2\mu B}{n^3} > 0$ for all μ, B, $n>0$, we can find the optimal value of n which minimizes C(n) by setting (2) equal to zero. This yields the optimal value of n as $n^* = \left[\frac{2\sigma\mu B}{s+\lambda a}\right]^{1/2}$.

If n^* is not an integer, compute $C(n)$ for $n=[n^*]$ and $n=[n^*]+1$, where $[n^*]$ represents the greatest integer less than or equal to n^*. The value of n that yields the smaller value for $C(n)$ is the optimal integer value of n.

4.2. Optimal fixed time interval policy

In this section we compute the expected total cost per unit time as a function of time t and derive the optimal backup operation time t^* which minimizes the expected total cost per unit time.

If the backup operation is taken at a fixed time interval t, the expected total cost per unit time is given by

$$C(t) = \frac{1}{t}\{B + b\cdot E[Y(t) + E(S(t)] + E[R(t)]\}.$$

Since $Y(t) = \sum_{i=1}^{N(t)} Z_i$ and $N(t)$ and Z_i's are independent $E[Y(t)] = E[N(t)]\cdot E[Z_i]$.

To get an explicit expression for $E[Y(t)]$, we need to know the specific form of $E[N(t)]$ which has one-to-one correspondence with the distribution of the time between transaction arrivals (X_i) and is hard to derive as an explicit expression in general cases. However, if the process $\{N(t),\ t\geq 0\}$ is a Poisson process with increasing rate of μ, then the mean-value function is known as linear, i.e. $E[N(t)]= \mu t/\sigma$. Here we assume that the transaction arrival process is a Poisson process to get an explicit expression for $C(n)$. Then,

$$E[S(t)] = s\int_0^{\infty} E[Y(\omega)]\,d\omega = s\int_0^{t} (\mu\omega/\sigma)d\omega = \frac{s\mu}{2\sigma}t^2.$$

The expected recovery cost until time t may be expressed as

$$E[R(t)] = \sum_{m=0}^{\infty} E[R(t)\,|\,M(t) = m]\ P\{M(t) = m\},\ \text{where}\ P\{M(t) = m\} = (\lambda t)^m e^{-\lambda t}/m!.$$

However, $E[R(t)\,|\,M(t) = m]$ can be computed by conditioning the number of transactions arrived until time t.

$$E[R(t)\,|\,M(t) = m]$$

$$\sum_{n=0}^{\infty} E[R(t)\,|\,M(t) = m,\ N(t) = n]\ P\{N(t) = n\},\ \text{where}\ P\{N(t) = n\} = (\mu t)^n e^{-\mu t}/n!$$

since we assume that $\{N(t),\ t\geq 0\}$ is a Possion process with increasing rate of μ. But

$$\begin{aligned}
E[R(t)|M(t) &= m,\ N(t) = n] \\
&= mR + a(0 \cdot E[K_1|M(t)=m] + (1/\sigma)\cdot E[K_2|M(t)=m] + \cdots \\
&\quad + (n-1)(1/\sigma)\cdot E[K_{n-1}|M(t)=m] + (n/\sigma)\cdot E[L_n|M(t)=m]) \\
&= mR + a(1/\sigma)m \cdot (\frac{E[X_2}{t} + 2\frac{E[X_3}{t} + \cdots + (n-1)\frac{E[X_{n-1}}{t} + n\frac{E[t-T_n}{t}) \\
&= mR + (am/\sigma)\,\{(\sum_{i=1}^{n-1} i\frac{E[X_i}{t}) + \frac{E[t-T_n}{t}\} \\
&= mR + (am/\sigma)\,\{(\sum_{i=1}^{n-1} i\frac{1/\mu}{t}) + \frac{t-n/\mu}{t}\} \\
&= mR + (am/\sigma)\,(\frac{n(n-3)}{2\mu t} + 1).
\end{aligned}$$

Here L_n is the number of failures during the time between t and the nth transaction arrival time, $t-T_n$.

Hence,

$$\begin{aligned}
E[R(t)|M(t) &= m] \\
&= \sum_{n=0}^{\infty}\{mR + (am/\sigma)(\frac{n(n-3)}{2\mu t} + 1)\}\,(\mu t)^n e^{-\mu t}/n! \\
&= (am/2\mu\sigma t)\sum_{n=0}^{\infty} n(n-3)(\mu t)^n e^{-\mu t}/n! + (mR + am/\sigma)\sum_{n=0}^{\infty}(\mu t)^n e^{-\mu t}/n! \\
&= (am/2\mu\sigma t)\sum_{n=0}^{\infty} n^2(\mu t)^2 e^{-\mu t}/n! - (3am/\mu\sigma t)\sum_{n=0}^{\infty} n(\mu t)^n e^{-\mu t}/n! + (mR + am/\sigma) \\
&= mR + am/\sigma + am(\mu t-2)/2\sigma \\
&= m\{R + a/\sigma + a(\mu t-2)/2\sigma\}
\end{aligned}$$

Therefore,

$$\begin{aligned}
E[R(t)] &= \sum_{n=0}^{\infty} m\{R + a/\sigma + a(\mu t - 2)/2\sigma\}\,(\lambda t)^m e^{-\lambda t}/m! \\
&= \{R + a/\sigma + a(\mu t - 2)/2\sigma\}\sum_{n=0}^{\infty} m\cdot(\lambda t)^m e^{-\lambda t}/m! \\
&= \{R + a/\sigma + a(\mu t - 2)/2\sigma\}\cdot(\lambda t) \\
&= \lambda Rt + a\mu\lambda t^2/2\sigma
\end{aligned}$$

Finally, the expected total cost per unit time under the fixed time interval policy is given by

$$C(t) = B/t + b\mu/\sigma + \frac{s\mu + a\mu\lambda}{2\sigma}t + \lambda R. \tag{3}$$

Again, taking the derivative of C(t) with respect to t yields

$$\frac{dC(t)}{dt} = C'(t) = -B/t^2 + (s\mu + a\mu\lambda)/2\sigma. \tag{4}$$

Since the second derivative of C(t) with respect to t is $C''(t) = 2B/t^3 > 0$ for all B and $t > 0$, we may get the optimal time interval for the backup operation which minimzes C(t) by setting (4) equal to zero.

This yields the optimal time interval for the backup operation

$$t^* = \left[\frac{2B\sigma}{\mu(s+a\lambda)}\right]^{1/2} \tag{5}$$

4.3. Comparision of two policies

As mentioned earlier, implementing the control limit policy requires continuous monitoring of the system state (number of transactions arrived, in our case) and implementing the fixed time interval policy needs continuous monitoring of a real−time clock. In general, the fixed time interval policy is easier to implement than the control limit policy because observing a real−time clock is easier than continuous monitoring of the system state. Accordingly, in many cases, an implementation of the control limit policy brings with extra cost. However, in our backup operation and the other DBS related operations discussed in Section 2, a continuous monitor of the system state can as easily be done as a continuous monitor of a real−time clock by softwares such as DBMS or operation system. Therefore, we may find the superior policy by directly comparing the expected long−run average cost per unit time under the optimal control limit policy and the expected total cost per unit time under the optimal fixed time interval policy.

By evaluating C(t), expressed in (3), at $t = t^*$, we get the expected total cost per unit time under the optimal fixed time interval policy. Similarly, by evaluating C(n), expressed in (1), at $n = n^*$, we get the expected long−run average (total) cost per unit time under the optimal control limit policy. The difference of these two values is given as follows.

$$C(t^*) - C(n^*) = (s + a\lambda)/2\sigma > 0 \text{ for all s, a, } \lambda\text{, and } \sigma > 0.$$

Therefore, we may conclude that the control limit policy is superior to the fixed time interval policy in our case if the transaction arrival process is a Piosson process. Observe that the difference increases as the mean of the number of updates in each transaction, $1/\sigma$, each update storage cost, s, and/or the failure rate, λ, increase.

Additionally, remember that to get an explicit expression for the expected total cost per unit time under the fixed time interval policy, we need to know the specific form of the transaction arrival process and we assumed that the transaction arrival process is a Poisson process. On the other hand, for the control limit policy we only need to know the mean of the time between transaction arrivals. Therefore, the control limit policy requires less stringent assumption about the transaction arrival process, which is an additional benefit of using the control limit policy.

To illustrate our model and the difference of two policies, let us consider a hypothetical but realistic example. Assume that the primary copy of a database file is stored on a disk and that the logs are maintained on online tape. Suppose further that $B = R = \$1000$, $b = \$1$ per update, $s = \$3.2*10^{-9}$ per update per sececond, $a = \$0.01$ per update, $\mu = 2$ per second, $\sigma = 1/3$ (3 updates per transaction in average), and $\lambda = 4.1*10^{-7}$ per second (approximately one failure per month). Then $n^* = 641061$ transactions or about once every 3.7 days for an expected cost per unit time of $\$6.01/\text{sec}$ and $t^* = 213687$ seconds (approximately 2.47 days) for almost the same expected cost per unit time. The difference in the expected costs per unit time for two optimal policies becomes $1.095*10^{-8}$. The optimal fixed time interval policy costs slightly more than the optimal control limit policy. Although the cost difference looks relatively small, we need to remember that this is the cost difference per unit time. Therefore, if the system is operated for a long time period, the actual cost difference may be considerable. And as s, a, λ, and/or $1/\sigma$ increase, the difference in the expected cost per unit time increase too.

5. Conclusion

For an IS to be successful the continual control of the DBS is very important. In this paper we described DBS related such control operations and decision problem in each of them. Then we modeled a general batch backup situation, derived the optimal timing (frequency) of the control operation (backup operation) for two different kinds of control policies, and compared them. For the control limit policy the Renewal Reward Theorem was used to find the expected long-run average cost (backup, recovery, and update storage cost in the transaction log) per unit time, which was then minimized to find the optimal backup control limit in terms of the number of transactions. For the fixed time interval policy we assumed the transaction arrival process is a Poisson process to obtain an explicit expression for the expected total cost per unit time. We showed that the control limit policy is superior to the fixed time interval policy and the former requires less stringent assumption for the transaction process than the latter does.

Currently, the complexity and variety of implementation strategies and the difficulty of acquiring all the pertinent parameters make this type of analysis often difficult to perform. As the standardization of implementations increases within advanced database management systems, and the size of the files increases, the value and ease of such analysis will improve. The responsibility for estimating the pertinent parameters will reside with the database administrator. Information on the number of transactions, the number of updates (records), the rate of trasaction arrivals, and the rate of failure occurrences may be obtained easily by softeware monitoring during normal production runs. Determining the costs of each control operation, the recovery costs, and the update (records) storage costs are somewhat more involved, but not too difficult. If the overhead from such software measurement is prohibitive, hardware monitors might be used. Alternatively, occasional measurement programs might be run against the database to sample performance. Estimation of the cost of each control operation may be obtained each time a control operation is performed.

The workload of a typical DBS is seldom constant. The operations staff will normally use long-term batch operations such as backups, reorganizations, and batch updates to level the system load during periods of light activity, such as the late night hours or weekends. Thus in practice the above optimal solutions are used only as guidelines to pick among nights or weekends, say, that a particular file should have a backup or maintenance done. Using the expected interval between these operations, it is clearly advantageous to schedule them so that the effort is spread among the inactive periods as evenly as possible to avoid "bottleneck nights".

Finally, it should be remarked that substantial economies may result from the policy of performing the reorganization at the same time that a backup of the database is created. In fact, if the backup is maintained on the same medium as the production database, the nonreorganized database might serve as the backup.

REFERENCES

[1] Aghili, H. and Severance, D. G., "A Practical Guide to the Design of Differential Files for Recovery of On-Line Databases", *ACM Transactions on Database Systems*, Vol.7, No.4 (1982), pp.540-565.

[2] Coffman, E. G., Jr. and Gilbert, E. N., "Optimal Strategies for Scheduling Checkpoints and Preventive Maintenance", *IEEE Transactions on Reliability*, Vol.39, No.1(1990), pp.9-18.

[3] Date, C. J., *An Introduction to Database Systems*, Vol. II, Addison-Wesley, Reading, Mass., 1983.

[4] Duda, A., "Performance Analysis of the Checkpointing-Rollback-Recovery System via Diffusion Approximation", *Proceedings of the International Workshop on Applied Mathematics and Performance Reliability Models of Computer Communication Systems* (Pisa, Italy, Sept. 26-30, 1983), pp.387-399.

[5] Geist, R., Reynolds, R., and Westall, J., "Selection of a Checkpoint Interval in a Critical-Task Environment", *IEEE Transactions on Reliability*, Vol.37, No.4(1988), pp. 395-400.

[6] Heyman, D. P., "Mathematical Models of Database Degradation", *ACM Transactions on Database Systems*, Vol.7, No.4(1982), pp.615-631.

[7] L'ecuyer, P. and Malenfant, J., "Computing Optimal Checkpointing Strategies for Rollback and Recovery Systems", *IEEE Transactions on Computers*, Vol.37, No.4(1988), pp.491-496.

[8] Leung, C. and Choo, Q., "On the Execution of Large Batch Programs in Unreliable Computing Systems", *IEEE Transactions on Software Engineering*, Vol.SE-10 (1984), pp.444-450.

[9] Lohman, G. M. and Muckstadt, J. A., "Optimal Policy for Batch Operations: Backup, Checkpointing, Reorganization, and Updating", *ACM Transactions on Database Systems*, 2, No.3(1977), pp.209-222.

[10] Mendelson, H. and Yechiali, U., "Optimal Policies for Data Base Reorganization", *Operations Research*, Vol.29, No.1(1981), pp.23-36.

[11] Ross, S. M., *Introduction to Probability Models*, Academic Press, New York, 1981.

[12] Severance, D. G. and Lohman, G. M., "Differential Files: Their Application to the Maintenence of Large Database", *ACM Transactions on Database Systems*, Vol.1, No.3(1976), pp.256-267.

[13] Shin, K. G., Lin, T., and Lee, Y., "Optimal Checkpointing of Real-Time Tasks", *IEEE Transactions on Computers*, Vol.36, No.11(1987), pp.1328-1341.

[14] Sockut, G. H. and Goldberg, R. P., "Database Reorganization-Principles and Practice", *Computing Surveys*, Vol.11, No.4(1979), pp.371-395.

[15] Tantawi, A. N. and Ruschitzka, M., "Performance Analysis of Checkpointing Strategies", *ACM Transactions on Computer Systems*, Vol.2, No.2(1984), pp.123-144.