BUSINESS PROCESS BASED DATABASE RECOVERY AND EXPERIMENTAL RESULTS

Dr. Pinaki Mitra[1] , Girish Sundaram[2] and Arun Kumar Tripathi[3]

[1]Department of Computer Engineering, Indian Institute of Technology, Guwahati, India
pinaki@iitg.ernet.in
[2]IBM India Software Labs, Pune, India
gisundar@in.ibm.com
[3]Department of Computer Engineering, Indian Institute of Technology, Guwahati, India
t.arun@iitg.ernet.in

ABSTRACT

In this research paper we have explained and presented the results of a novel database recovery method. Taking into account that most data servers today are becoming increasingly autonomic and business process oriented, there is a need to develop a new model where it is possible to do a database recovery based on the occurrence of a business process rather than non intuitive data like time, log file name/location or system change number. The advantage of this recovery model is that the user will no longer be required to remember non intuitive data like timestamp up to which the data has to be restored, log file name/location or system change number. Moreover the database server has built in intelligence to identify transaction behaviour, study the pattern among these batches and then storing recovery information specific to these business process to be used later for recovery. Another advantage is that the down time needed for restoring the database is reduced drastically since the trial and error approach followed by conventional data servers in the absence of accurate recovery information is eliminated. We have also simulated the Idea using JAVA API's and Apache Derby and presented the experimental results for reference..

KEYWORDS

Database Recovery, Business Process based database recovery model

1. Introduction : Database Recovery – A Background and Need for Improvement

Database restoration is an activity of replacing an existing database or creating a new database using a previous version / copy of the backup taken at a older point of time and using transaction logs and archive logs (if existing) to apply the transactions to roll forward the database to a valid and consistent state.

Restoration techniques existing in today's conventional data servers are restricted to the following models:

- Time-based recovery model, also called point-in-time recovery (PITR), which recovers the data up to a specified point in time.

- Transaction log based recovery model where database is rolled forward till the transactions from a specific transaction log file (Archive or unarchived) is applied.
- Change-based recovery model or log sequence recovery model based on the system change number assigned by the data server.

For any of these models to work, the database user or Administrator needs to be aware of the following three things:

- **Timestamp information** , in case of Time-based recovery model
- **Log file** location and name information, in case of Transaction log based recovery model
- **System change number (SCN) or Log sequence number (LSN),** in case of Change-based recovery model or log sequence recovery model.

The drawback with these models is that restoration of database without data loss is not possible unless and until anyone of the above three inputs is known accurately. If any of the above three points are not known accurately then recovery is not possible. For instance, if the user Joe knows that he had performed an EOD banking business operation on the database which has entered invalid data into the data tables on 23-4-2007 between 2 and 2:30 pm. But since the exact timestamp information is missing we need to follow an iterative process using a trial and error method to arrive to the right database state. In today's mission critical 24X7 database installations, the time lost can result in loss of millions of dollars.

Taking into account that most data servers today are becoming increasingly autonomic and business process oriented, there is a need to develop a new model where it is possible to do a database restore based on the occurrence of a business process rather than non intuitive data like time, log file name/location or system change number.

A business process is a set of coordinated tasks and activities, conducted by people and equipment / software, which will lead to accomplishing a specific organizational goal.

In terms of a database server a business process can be defined as a set of individual batches each of which can be further decomposed into a set of individual SQL statements. An example would be a banking **EOD account balance validation batch** process which would ideally consist of the following statements:

1) **Getting the account details from the account table**
   Select cust_id, account_type, acc_balance from acc_table, cust_table into :custid,:account_type,:balance;

2) **Updating EOD table for this customer**
   Update EOD_table E set EOD_balance =: balance where E.cust_id =: custid

3) **Inserting into transaction table the activities carried out by the customer for that day**
   Insert into tran_table (select cust_id, tran_id, date, transaction_activity)

NOTE: This batch is used only for illustration purpose and actual banking transactions may vary.

The core Business Process Based Database Recovery Model can be summarized in the following few steps:

1) An intelligent system which collects information about various transactions executing at various points in time and logs them in a system catalog.
2) Analyses the information collected in step 1 over a period of time to identify the pattern of the transactions executed
3) The pattern that the system strives to identify is number of executions of a specific business batch process, is this number greater than the defined threshold, time of execution, number of users executing the batch process.
4) Once a definite pattern for constantly executed batches is identified these batches are given some system name which can be later modified by the database admin depending on what these batches are doing from the business point of view. For example EOD_batch, HR_Batch etc.
5) The system stores recovery information for each of these identified batches which can then be later used to perform database recovery if needed.

The advantage of this model is that the user will no longer be required to remember non intuitive data like timestamp up to which the data has to be restored, log file name/location or system change number. Moreover the database server has built in intelligence to identify transaction behaviour, study the pattern among these batches and then storing recovery information specific to these business process to be used later for recovery. Another advantage is that the down time needed for restoring the database is reduced drastically since the trial and error approach followed by conventional data servers in the absence of accurate recovery information is eliminated.

2. Business Process Based Database Recovery (BPBDR) – An pproach

2.1. Algorithms

The Implementation can be divided in to two parts the logging & pattern determination phase and the recovery phase. Both the phases are described with the help of algorithms below.

Logging & Pattern determination Module Algorithm:

1. A database transaction T1 starts.
2. Log statements of the transaction in the system log table.
3. If transaction is complete go to next step else continue with step 2
4. Give the logged batch transaction a system generated name
5. Compare the statements inside the batch with previously stored batches
6. If statements are matching add/update information to existing batch, transaction details in the system catalog else create new entries in the system catalog for the batch and add details like number of users who have executed the batch, transaction logs storing the data used in the batch, commit time of the batch, data files storing data of this batch, SQL statement details of the batch.
7. If N (number of times the batch was executed previously) > T (Threshold value defined by the database user after which the user should be prompted to give the batch a user defined name since it has been identified as a frequently executed batch) then prompt the user for a name for the batch process else go to next step
8. Exit

Recovery Module Algorithm:

1. User submits a request to perform business process based recovery.
2. System prompts the user with a list of identified business batch process names whose recovery information is available in the database server along with the timestamp of the occurrences of each of these batches
3. User selects one of the business batch process to be used for database recovery
4. Database system fetches the recovery information from the system catalog pertaining to the selected business batch process
5. Database system verifies whether all the related objects needed for performing a consistent database recovery like transaction logs, data files etc are available, if available go to next step else go to step 7
6. Database performs recovery till the consistent point in time when the selected batch process was executed. Go to next step.
7. Exit

3. JAVA Implementation of BPBDR

Business processes are stored in the form of B+ tree. Records of the B+ tree are Business processes. This helps to store and retrieve business process efficiently. A business process is said to distinguished if it's SQL statements are different from existing one. So SQL statement(s) is chosen as a key of B+ tree record.

3.1. Data Structure's for Business Processes

If (ExeCount - Threshold) is not large then we use data structure explained in Table 3.1.1 and if (ExeCount - Threshold) is very large in the order of thousands we use data structure explained in Table 3.1.2.

Table 3.1.1: Business Process Data Structure – I

FieldName	Data Structure
SQLStatements	String
ExeCount	Integer
Threshold	Integer
UserCount	Integer
LastExeTime	String Array
BusinessProcessDetails	String
RevSQLStatements	String
BP_Name	String
Users_List	Linked List
AboveThreshold_Updates	Linked List

Table 3.1.2: Business Process Data Structure - II

FieldName	Data Structure
SQLStatements	String
ExeCount	Integer
Threshold	Integer
UserCount	Integer
LastExeTime	String Array
BusinessProcessDetails	String
RevSQLStatements	String
BP_Name	String
Users_List	B+ tree
AboveThreshold_Updates	Interval Tree

Table 3.1.3: Above Threshold Updates Field's Data Structure

Fieldname	Data Structure
UserName	String
Args	Linked List
RevArgs	Linked List
Exe_Date_and_Time	Date

Interval Tree Data Structure for AboveThreshold_Updates

This field is a collection of some specific elements shown in Figure 3.1.4. AboveThreshold Updates value is used to store recovery information. This information would be displayed in recovery table. System fetches this information to Display recovery table and perform recovery for a specific execution of specified business process.

Figure: 3.1.4: Recovery Information for Deposit Amount Business Process.

UserName : arun
Args : 2500 0-1-2222-3333-5555
RevArgs : 2500 0-1-2222-3333-5555
Exe_Data_and_Time : 2010/03/09 08:07:33 PM

Recovery Information for Business Process Executed at 2010/03/09 08:07:33 PM

We have a set of business processes Recovery Information those are executed at different time instance. We arranged these Recovery Information in the form of time intervals. We have set of time intervals and recovery Information values correspondingto those intervals.For Example Business processes executed from 2010/03/09 to 2010/04/13 are kept in time interval d.

interval d = {2010/03/09, 2010/04/13}

Values for d = List of Recovery Information for the Business Processes those are executed in interval d.

3.2. Algorithms

"Business Process Based Database Recovery Model"can be divided into two modules the Logging and Pattern Determination Module and the Recovery Module.

Both the modules are described with the help of algorithms below.

Figure 3.2.1 Logging and Pattern Determination Module

Data: *New BP*

Result: Business Processes Database Updated
initialization: Log and Pattern Determination Table;

while *Any BP exists in Log and Pattern Determination Table* do
read *Current BP*;
if *New BP = Current BP* then
if *ExeCount of Current BP _ Threshold of Current BP* then
if *ExeCount of Current BP = Threshold of Current BP* then
Assign User Defined *BP Name* to *Current BP*;
Add new entry in *AboveT hreshold Updates* of *Current BP*;
if *UserName of New BP does not exist in Users List of Current BP*
then
Add UserName in *Users List*;
Increment ExeCount by 1 of *Current BP*;
Update LastExeTime of *Current BP*;
Update Business Processes Database;
Exit;
if *All Business Processes do not match with New BP* then
Assign System generated name to *New BP*;
Add *New BP* in Business Processes Database;

Figure 3.2.2 Recovery Module

Data: BP For Recovery, Exe Date and Time

Result: Database Rollback for BP For Recovery
initialization: Recovery Table;
while *Any BP exists in Recovery Table* do
read Current BP;
if *BP For Recovery = Current BP AND (Exe Date and Time of BP For Recovery = Current BP instance)* then
Fetch Recovery Information for Current BP instance;
if *Recovery Record is OK* then
Set Recovery parameters for recovery;
Perform Business Process Recovery;
if *Database Recovery is done Successfully* then
Update *AboveT hreshold Updates* of *Current BP*;
Display Message, Business Process Recovered Successfully;
else

Database Error, Business Process not recovered;
else
Display Message, Recovery record is Corrupted;
Exit;

if *All Business Processes do not match with BP For Recovery* then
Display Message, Business Process instance does not exist;

3.3. Configuration and Usage

Configuration and usage of Business Process Based Database Recovery Java API is very easy. It does not need any special software or Hardware. We have used Apache Derby database to store data so Apache Derby database server must run in your system. you need to import our Java API and your desired Business Process(s) only. Detailed Configuration and use is explained with the help of Allocate Banker Business Process. Allocate Banker business process is represented in the form of Allocate_Banker.java This Allocate_Banker.java contains SQLStatement, Reverse_SQLStatement, BP_Name,BP_Detail, Threshold_value, Args, and RevArgs fields. ExecuteBP() and ExecuteRevBP() are to primary methods of Allocate_Banker.java used communicate with Apache Derby database.

3.4. Load Business Processes on Buffer

```
Logging BP newLogging BP = new Logging BP();
BTree<String, BusinessProcess> Load BPs
Load BPs = newLogging BP.LoadBP on Buffer(FileName);
```

3.5. Execution and Logging of Business Process

```
Allocate Banker newAllocateBanker = new Allocate Banker();
Vector Args = new Vector(); Vector RevArgs = new Vector();
newAllocateBanker.setThreshold value(1);
Args = newAllocateBanker.getSQLStatement Args(”Clear Water Bay”, ”C8392380567”, ”sta
rosana”);
RevArgs = newAllocateBanker.getRevSQLStatement Args(”Clear Water Bay”, ”C8392380567”, ”sta
rosana”);
Date and Time newDate and Time = new Date and Time();
BP Updates newBP Updates1 = new BP Updates(”arun”, Args, RevArgs, newDate and Time.getDate
and Time());
BusinessProcess newBP1 = new BusinessProcess(newAllocateBanker.getSQLStatement(),
newAllocateBanker.getThreshold value(),
newAllocateBanker.getBP Detail(), newAllocateBanker.getReverse SQLStatement(), newBP
Updates1.getUserName());
Executed = newAllocateBanker.ExecuteBP();
If (Executed)
{
newBP1.setBP Name(”Sys”+newLogging BP.getSystemProcessNo());
newBP1.setLastExeTime(newDate and Time.getDate and Time());
newLogging BP.Log BusinessProcess(Load BPs, newBP1, newBP Updates1);
System.out.println(”AllocateBanker Business Process Executed”);
}
Else
{
System.out.println(”AllocateBanker Business Process Not Executed”);
}
```

3.6. Display Logging and Pattern Determination Table

newLogging BP.Display Log and Pattern Determination Table(Load BPs);

3.7. Display Recovery Table

newLogging BP.Display Recovery Table(Load BPs);

3.8. Display Recovery Table for specific time interval

newLogging BP.Display Recovery Table(Load BPs, StartInterval, EndInterval);

3.9. Perform Recovery

```
RevArgs = newAllocateBanker.getRevSQLStatement Args(”Downtown”, ”C3487327487”, ”sta
carina”);
newBP Updates1 = new BP Updates(”arun”, Args, RevArgs, RecoveryTime);
newBP1 = new BusinessProcess(newAllocateBanker.getSQLStatement(),
newAllocateBanker.getThreshold value(), newAllocateBanker.getBP Detail(),
newAllocateBanker.getReverse SQLStatement(), newBP Updates1.getUserName());
Executed = newAllocateBanker.ExecuteRevBP();
If (Executed)
{
newLogging BP.Perform Recovery(newBP1.getSQLStatements(), newBP Updates1, Load BPs);
System.out.println(”AllocateBanker Business Process Recoverd”);
}
Else
{
System.out.println(”AllocateBanker Business Process Not Recoverd”);
}
```

3.10. Store Business Processes on Stable Storage

newLogging BP.StoreBP on Disk(StoreFileName, Load BPs);

4. Experimental Results

We have derived a formula for calculation of optimal checkpoint interval. The Formula considers input failure rate, checkpoint overhead, redo time and number of log records as input variables. For check point overhead and redo time we have analyzed the effect of disk parameters such as sector size, number of sectors per track, spindle speed and log record entry size. The proposed formula has been simulated in java. The results are presented in a XY graph.

Proposed model assumes that plenty of jobs are ready to execute for a predefined time span. In this whole time span all jobs should be executed. We cannot continue executing all jobs without taking checkpoint because system may crash at any time whose probability of failure is already approximated.
To execute all jobs successfully system should take checkpoint after a certain time. And if failure occurs in interval then redo should be done for not committed jobs. We have derived an optimal checkpoint interval on the basis of all factors that affects the length of checkpoint interval.

Let’s assume

Total Transaction time span in which all jobs would be executed is T ms.

Length of check-pointing interval: t ms.
Redo time for each log record: t_1 ms.
Checkpoint overhead: t_2 ms.
Probability of failure during a single time unit (Interval): p
Average number of log record instructions written in each time unit: c
Calculated Expected Failure time during the transactions life span = T*p

From the above assumption checkpoint overhead for whole time span T would be $(T/t)* t_2$ and failure cost would be $(T*p)* t_1*(c/2)*t$.
Calculated Total cost of transactions execution
$F(t) = (T/t)* t_2 + (T*p)* t_1*(c/2)*t$
Say $k_1 = (T*p)* t_1*(c/2)$
$k_2 = T* t_2$
$F(t) = k_2*(1/t) + k_1*t$ ----------(1)
To find the value of F(t) which minimizes t, we differentiate F(t) with respect to t.
$f(t) = (d/dt)\{F(t)\}$
$f(t) = k_1 - k_2*(1/t^2)$ ----------(2)
For optimal checkpoint $f(t) \approx 0$ and $t = t_{opt}$
$0 = k_1 - k_2*(1/ t_{opt}^2)$
$t_{opt}^2 = k_2/ k_1$
$t_{opt} = \sqrt{(k_2/ k_1)}$
$t_{opt} = \sqrt{\{(T* t_2)/((T*p)* t_1*(c/2))\}}$
$t_{opt} = \sqrt{\{(t_2)/(p* t_1*(c/2))\}}$----------(3)
From Equation (3) it is clear that parameters t_1, t_2, p and c have a major impact on optimal checkpoint interval.

4.1 Simulation Results

From the Equation $t_{opt} = \sqrt{\{(t_2)/(p* t_1*(c/2))\}}$ it is clear that t_2, p, t_1, and c are the parameter that affects the checkpoint interval. Parameters t_1 and t_2 can change on the basis of disk parameters such as spindle speed, number of sectors per track sector size etc.
$t_{opt} \propto \sqrt{t_2}$

Checkpoint interval should increase if t2 is increasing. Value of t2 itself depends on Main Memory buffer size and disk parameters. We have assumed that not whole buffer content is written back into secondary storage but only parts of buffer which is altered is written. For our experiment purpose we have assumed that on an average half of the Main Memory Buffer content is updated per interval. Our experiment in Fig 4.1.1 shows that as we increase buffer size checkpoint interval is increasing. It should happen because if we have sufficient buffer size then only we can keep all updates for an interval otherwise we have to take checkpoint frequently to make updates persistent.

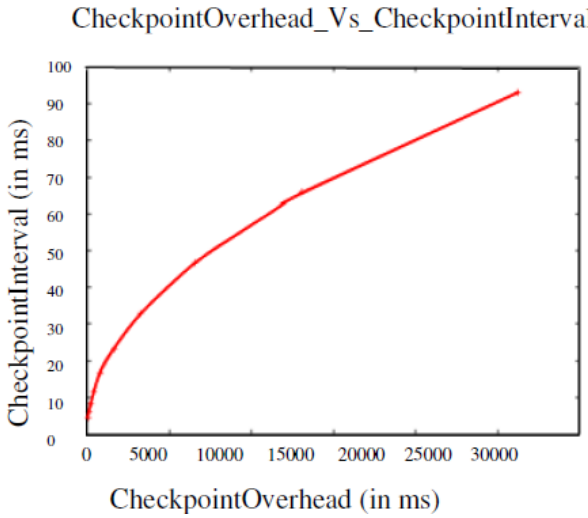

Figure 4.1.1: Checkpoint interval Vs checkpoint overheadt_{opt} α $\sqrt{(1/t_1)}$

Disk parameters, log record size and log processing time are the major parameters which affect the value of redo time for each log record. In case of optimal checkpoint interval if redo time is increasing then restart recovery time would increase, it's an overhead for system. If we keep this redo time minimum we can execute more jobs. To execute more jobs we have to reduce restart recovery time, which can be reduced by decreasing checkpoint interval. The overall conclusion is that as soon as we increase redo time checkpoint interval is decreased. If we decrease redo time then checkpoint interval should increase. That's result of our simulation shown in Fig 4.1.2.

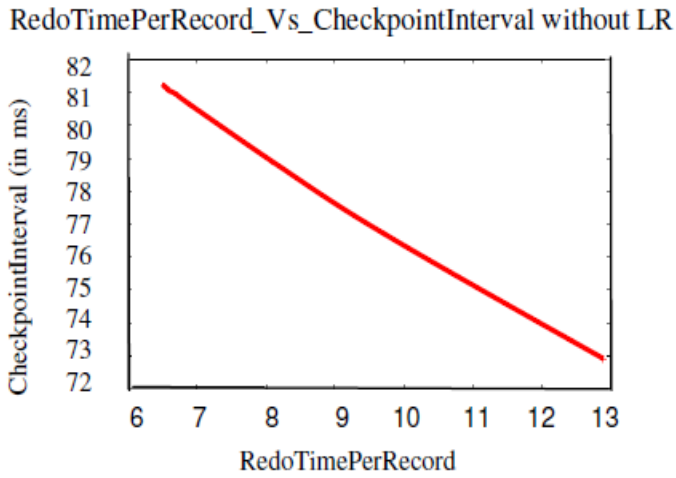

Figure 4.1.2: Checkpoint interval Vs RedoTime for each record

We have also shown in Fig 4.1.3, 4.1.4 and 4.1.5 the results of checkpoint interval affected by disk parameters. Especially if we are increasing number of sectors, disk access time would decrease effect of this, checkpoint overhead would decrease. Result of that checkpoint interval should decrease that is result of our experiment. Similar case is with spindle speed and sector size.

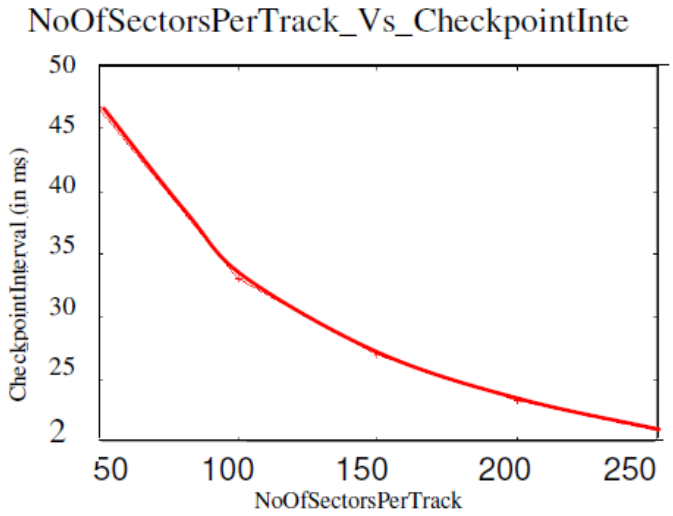

Figure 4.1.3: Checkpoint interval Vs Number Of Sectors Per Track

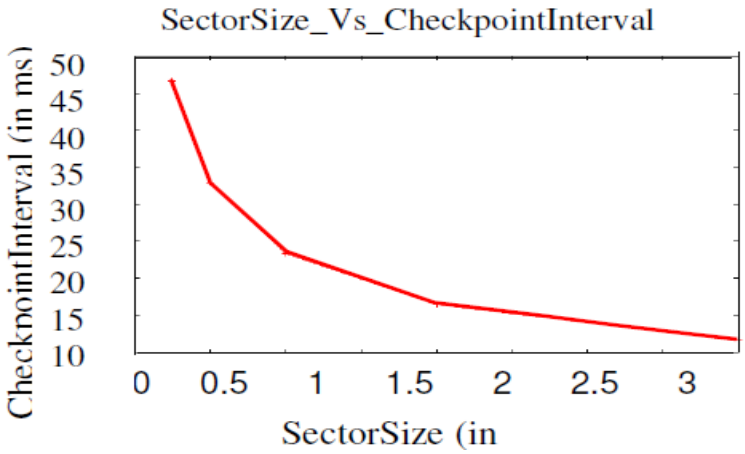

Figure 4.1.4: Checkpoint interval Vs Sector size

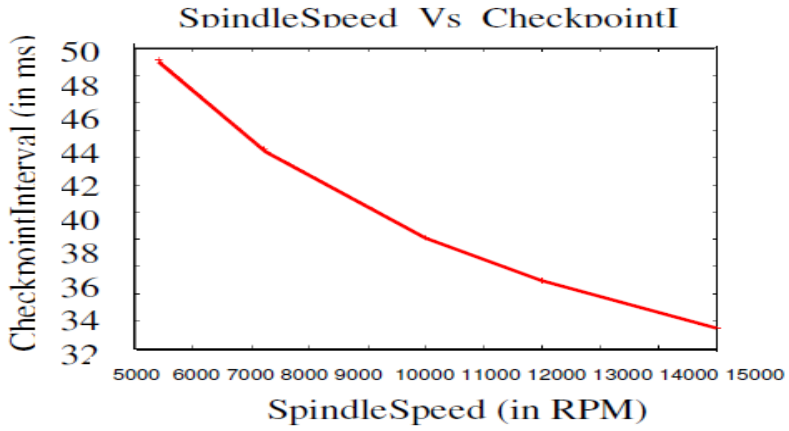

Figure 4.1.5: Checkpoint interval Vs Spindle speedt_{opt} α √(1/c)

C is number of log records written in each interval. We have assumed that in case of failure on an average c/2 log records are written. In case of failure for each log record system has to perform redo operation. So for large number of log records overall redo time would be long and for small number of log records it would be small. That's our experiment shows in Fig 4.1.6 that if we increase number of log records checkpoint interval is decreasing.

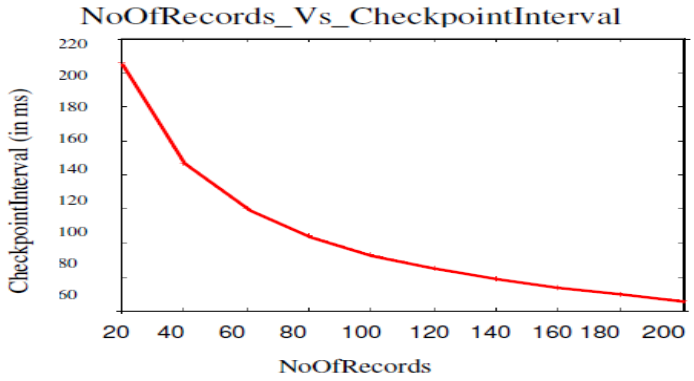

Figure 4.1.6: Checkpoint interval Vs NoOfrecords t_{opt} α √(1/p)

This experiment shows in Fig 4.1.7 that for failure rate checkpoint interval is long and for high failure rates it is small that is desired. If failure rate is very low and we take checkpoint frequently then this is simply wastage of time. We can use checkpoint overhead time to execute job. But if failure rate is high and we keep long checkpoint interval then probability of system crash is high and we have to do redo work again and again.

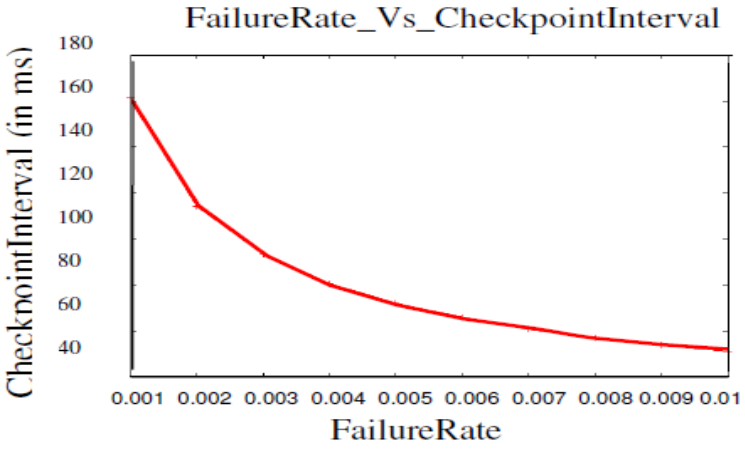

Figure 4.1.7: Checkpoint interval Vs Failure Rate

4.2 Conclusion

Proposed formula for optimal checkpoint interval has shown correct results with respect to the input parameters given to the formula. Verification of the formula is done by resulted graphs.
We have presented a formula for an optimal checkpoint Interval which considers disk parameters along with log record processing time, failure rate, and other factors.

We have developed a Java API for Business Process Based Database Recovery which allows Database user to restore / recover database based on a specific Business Process.The advantage of this model is that the user will no longer be required to remember non intuitive data like timestamp up to which the data has to be restored, log file name/location or system change number. Moreover the database server has built in intelligence to identify transaction behaviour, study the pattern among these batches and then storing recovery information specific to these business process to be used later for recovery. Another advantage is that the down time needed for restoring the database is reduced drastically since the trial and error approach followed by conventional data servers in the absence of accurate recovery information is eliminated.

References

[1] The International Journal of Database Management Systems (IJDMS), Vol.2, No.1, February 2010 Data Guard: A New Approach For Disaster Recovery & Rolling Upgrades

[2] The International Journal of Database Management Systems (IJDMS), Vol.2, No.2, May 2010 Dynamic management of transactions in distributed real-time processing system

[3] The International Journal of Database Management Systems (IJDMS), November 2010, Volume 2, Number 4 A New Optimistic Replication Strategy For Large-Scale Mobile Distributed Database Systems

[4] ISBN: 978-1-60558-102-6 Middleware-based database replication:the gaps between theory and practice

[5] ISBN: 978-1-60558-188-0 Dynamic data recovery for database systems based on fine grained transaction log

[6] SBN: 978-1-59593-686-8 Design of flash-based DBMS: an in-page logging approach

[7] ISBN: 978-1-4244-3422-0 Transaction Support for Log-Based Middleware Server Recovery

[8] Principles of transaction processing By Philip A. Bernstein, Eric Newcomer

[9] CNKI:SUN:SJSJ.0.2008-10-012 Recovery method for main memory database systems based on shadow paging

[10] ISBN: 978-0-7695-3363-6 Fine Grained Transaction Log for Data Recovery in Database Systems

[11] ISBN: 978-1-4244-6347-3 Database design and implementation based on postal enterprise CRM

[12] ISSN: 2150-8097 Database replication: a tale of research across communities

[13] ISBN: 978-1-4244-8959-6 Transactional In-Page Logging for multiversion read consistency and recovery

[14] ISBN: 978-1-4244-8630-4 Recovery and analysis of transaction scope from scattered information in Java Enterprise Applications

[15] International Journal of Information Security Volume 9, Number 1, 51-67, DOI: 10.1007/s10207-009-0095-0

[16] A column dependency-based approach for static and dynamic recovery of databases from malicious transactions

[17] Distributed and Parallel Databases Volume 23, Number 3, 189-205, DOI: 10.1007/s10619-008-7026-3

[18] Movement-based checkpointing and logging for failure recovery of database applications in mobile environments

[19] ISBN: 978-1-4244-4422-9 Dynamic content web applications: Crash, failover, and recovery analysis

[20] ISSN: 2150-8097 Segment-based recovery: write-ahead logging revisited

[21] ISBN: 978-1-60558-563-5 Analytical modeling of lock-based concurrency control with arbitrary transaction data access patterns

Authors

Dr. Pinaki Mitra *is an Assistant Professor at the Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati, INDIA.He did his PhD in Computer Science at the Simon Fraser University, Canada (1994) and M.E. in Computer Science at the Indian Institute of Science, Bangalore.He was a post doctoral fellow at Carleton University, Canada.He was a Research Scientist at Jadavpur University and Assistant Professor at National Institute of Management, Calcutta.*

Girish Sundaram *is a Solution Architect with the IBM India and works closely with various strategic Global IBM Accounts for implementing mission critical business solutions across various domains. He has wide ranging deep product expertise in Data Management , Datawarehousing & ETL development. He is an accomplished Inventor with a number of Filed Patents, Published Disclosures and has a rich experience in Research Publications in renowned International Journals.*

Arun Tripathi *is a M.Tech from Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati, INDIA and currently works for Samsung India as a Researcher in Information Management and Security.*