University of Kentucky
UKnowledge

Theses and Dissertations--Computer Science | Computer Science

2011

CHECKPOINTING AND RECOVERY IN DISTRIBUTED AND DATABASE SYSTEMS

Jiang Wu
University of Kentucky, smile_926@hotmail.com

Right click to open a feedback form in a new tab to let us know how this document benefits you.

Recommended Citation
Wu, Jiang, "CHECKPOINTING AND RECOVERY IN DISTRIBUTED AND DATABASE SYSTEMS" (2011). *Theses and Dissertations--Computer Science*. 2.
https://uknowledge.uky.edu/cs_etds/2

This Doctoral Dissertation is brought to you for free and open access by the Computer Science at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Computer Science by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

المنارة للاستشارات

www.manaraa.com

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained and attached hereto needed written permission statements(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine).

I hereby grant to The University of Kentucky and its agents the non-exclusive license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless a preapproved embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's dissertation including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Jiang Wu, Student

Dr. Dakshnamoorthy Manivannan, Major Professor

Dr. Raphael A. Finkel, Director of Graduate Studies

المنارة للاستشارات

www.manaraa.com

CHECKPOINTING AND RECOVERY IN DISTRIBUTED AND DATABASE SYSTEMS

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
Department of Computer Science
at the University of Kentucky

By
Jiang Wu

Lexington, Kentucky

Director: Dr. Dakshnamoorthy Manivannan, Professor of Computer Science

Lexington, Kentucky

2011

Copyright © Jiang Wu 2011

www.manaraa.com

ABSTRACT OF DISSERTATION

CHECKPOINTING AND RECOVERY IN DISTRIBUTED AND DATABASE SYSTEMS

A transaction-consistent global checkpoint of a database records a state of the database which reflects the effect of only completed transactions and not the results of any partially executed transactions. This thesis establishes the necessary and sufficient conditions for a checkpoint of a data item (or the checkpoints of a set of data items) to be part of a transaction-consistent global checkpoint of the database. This result would be useful for constructing transaction-consistent global checkpoints incrementally from the checkpoints of each individual data item of a database. By applying this condition, we can start from any useful checkpoint of any data item and then incrementally add checkpoints of other data items until we get a transaction-consistent global checkpoint of the database. This result can also help in designing non-intrusive checkpointing protocols for database systems. Based on the intuition gained from the development of the necessary and sufficient conditions, we also developed a non-intrusive low-overhead checkpointing protocol for distributed database systems.

Checkpointing and rollback recovery are also established techniques for achieving fault-tolerance in distributed systems. Communication-induced checkpointing algorithms allow processes involved in a distributed computation take checkpoints independently while at the same time force processes to take additional checkpoints to make each checkpoint to be part of a consistent global checkpoint. This thesis develops a low-overhead communication-induced checkpointing protocol and presents a performance evaluation of the protocol.

KEYWORDS: Distributed Systems, Distributed Database Systems, Communication-induced Checkpointing, Consistent Global Checkpoints, Transaction-consistent Global Checkpoints

المنارة للاستشارات

www.manaraa.com

Jiang Wu

01/12/2012

المنارة للاستشارات

www.manaraa.com

CHECKPOINTING AND RECOVERY IN DISTRIBUTED AND DATABASE SYSTEMS

By

Jiang Wu

Dr. Dakshnamoorthy Manivannan

Director of Dissertation

Dr. Raphael A. Finkel

Director of Graduate Studies

01/12/2012

المنارة للاستشارات

www.manaraa.com

ACKNOWLEDGEMENTS

It is a pleasure to thank those who made this thesis possible. I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family.

First of all, I would like to express my deepest gratitude to my advisor, Dr. Dakshnamoorthy Manivannan, for his excellent guidance, patience, caring, and leading/funding me to do research in fault tolerance in distributed systems. Dr. Manivannan has been a great mentor on every account, and his broad knowledge and constructive suggestions to this dissertation are sincerely appreciated.

I would like to thank the committee members Dr. Mukesh Singhal, Dr. Zongming Fei and Dr. YuMing Zhang for their helpful comments on my dissertation.

Also, I would like to thank Yi Luo who helped me finding papers and reviewed the simulation codes.

Thanks also go to all members in the Laboratory for Advanced Networking for their kind help.

Finally I would like to thank my family members. I need to thank my parents for giving their continuous support and encouragement. Most importantly, I would like to thank my wife for her great patience during my study at University of Kentucky.

المنارة للاستشارات
www.manaraa.com

Contents

Acknowledgements **iii**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**
1.1 Checkpointing in Distributed Database Systems 2
1.1.1 Background 2
1.1.2 Motivation 8
1.2 Checkpointing Distributed Computations 10
1.2.1 Background 10
1.2.2 Motivation 19
1.3 Contribution of the Thesis 20
1.4 Organization of the Dissertation 21

2 Necessary and Sufficient Conditions for Transaction-Consistent Global Checkpoints of a Distributed Database System **22**
2.1 Serialization Graphs 23
2.2 Necessary and Sufficient Conditions 29
2.2.1 Applications 44
2.3 Conclusion 45

3 A Non-intrusive Checkpointing Protocol for Distributed Database Systems **47**
3.1 Proposed Protocol 49
3.1.1 Basic Idea Behind the Protocols of Baldoni et al. 49
3.1.2 Proposed Checkpointing Protocol 51
3.2 Performance Analysis 52
3.2.1 Performance Analysis 52
3.2.2 Simulation Results 57
3.3 Conclusion 60

4 An Enhanced Model-based Communication-Induced Checkpointing Protocol for Distributed Systems **64**
4.1 An Example Showing Our Motivation 65
4.2 The Sufficient Condition 67

للاستشارات المنارة

www.manaraa.com

4.3 Relation Between Existing Model-based Checkpointing Protocols . . . 72
4.4 An Enhanced Model-based Checkpointing Protocol 74
4.5 Simulation Results 84
4.6 Conclusion 86

5 Conclusion and Future Work 89
5.1 Research Accomplishments 89
5.2 Future Work 90

Bibliography 96

Vita 101

www.manaraa.com

List of Tables

3.1 Proposed Checkpointing Protocol 61
3.2 Recovery Protocol Based on the Checkpointing Protocol 62
3.3 Two Sample T-test Result 63

للاستشارات المنارة
www.manaraa.com

List of Figures

1.1 Visible doubling . . . 15
1.2 How strong visible doubling helps in reducing forced checkpoints . . . 15
1.3 $Z - cycle$ . . . 17

2.1 Local serialization graph induced by transactions $\{T_9, T_3, T_4\}$ on data item x_1. . . . 26
2.2 Local serialization graph induced by transactions $\{T_2, T_3, T_7, T_8\}$ on data item x_2. . . . 26
2.3 Local serialization graphs induced by the transactions on the data items x_3, x_4, and x_5. . . . 27
2.4 Global serialization graph constructed from local serialization graphs. 27
2.5 Illustration of proof for case 1 . . . 39
2.6 Illustration of proof under case 2 . . . 43

3.1 Recovery cost . . . 54
3.2 Simulation results (a) basic checkpoints (b) forced checkpoints . . . 58
3.3 Simulation results (a) total checkpoints (b) average recovery cost . . . 59

4.1 Example showing unnecessary forced checkpoints induced by the protocol in [34]. . . . 65
4.2 Proof for case A . . . 69
4.3 Illustration of proof for case B.2 . . . 71
4.4 Illustration of proof for case B.3 . . . 71
4.5 Essential components of Baldoni's and Garcia's protocols . . . 74
4.6 Relation between some existing model-based communication-induced checkpointing protocols . . . 74
4.7 Case $send(\gamma'.first) \xrightarrow{hb} receive(m')$ . . . 77
4.8 Case $receive(m') \xrightarrow{hb} send(\gamma'.first)$ . . . 77
4.9 Case $i = j$ . . . 78
4.10 Case $i \neq j$ . . . 78
4.11 Case (3.1) . . . 80
4.12 Case (3.3) . . . 81
4.13 Example showing matrix *causal* . . . 81
4.14 The results of our simulation for 50 random distributed computations each of which consisted of 8 processes ($P = 8$), exchanged 1500 messages and took basic checkpoints with parameter IN=25-30 . . . 86

المنارة للاستشارات
www.manaraa.com

4.15 The results of our simulation for 50 random distributed computations each of which consisted of 15 processes, exchanged 1500 messages and took basic checkpoints with parameter IN=25-30 87
4.16 The results of our simulation for 50 random distributed computations each of which consisted of 20 processes, exchanged 1600 messages and took basic checkpoints with parameter IN=40-40 88
4.17 The results of our simulation for 60 random distributed computations each of which consisted of 25 processes, exchanged 1800 messages and took basic checkpoints with parameter IN=30-50 88

5.1 Relation of all algorithms including our future work 93

المنارة للاستشارات
www.manaraa.com

Chapter 1

Introduction

This thesis establishes the necessary and sufficient conditions for a checkpoint of a data item (or the checkpoints of a set of data items) to be part of a transaction-consistent global checkpoint of the database, and, based on these conditions, develops a checkpointing protocol for distributed database systems. This thesis also develops a communication-induced checkpointing protocol that reduces the forced checkpoints taken compared to some existing checkpointing protocols.

Databases are the backbone of all information systems. Databases have several other applications such as webpage development, electronic commerce and cloud computing. Implementing mechanisms for handling failures in database systems is critical for the success of any establishment. To cope with failures, the state of the database has to be saved in stable storage from time to time. A stable storage means non-volatile disk storage that can survive process or data item failures.

A distributed system consists of a collection of autonomous computers, connected by a communication network. A distributed computation is a set $P = \{P_1, P_2, ...P_n\}$ of processes $P_1, P_2, ...P_n$ running on a set of computers in a distributed systems trying to solve a single problem. If the states of the processes involved in the computation are saved periodically, when a failure occurs, they can be restarted from an intermediate consistent state (saved in stable storage) instead of restarting from its initial state. This reduces the amount of re-computation for long-running, time-critical applications. In this dissertation, we address issues related to fault-tolerance in database

www.manaraa.com

systems as well distributed systems.

Next, we present the necessary background for understanding the dissertation, as well as motivation for our research.

1.1 Checkpointing in Distributed Database Systems

1.1.1 Background

We use a model of the distributed database system similar to the model presented by Pilarski et al. [11]. In this model, a *distributed database* consists of a finite set of data items residing at various sites. A data item is the smallest unit of data accessible to transactions. Sites exchange information via messages transmitted on a communication network, which is assumed to be reliable. Message transmission time is unpredictable but finite. In addition, the data items at each site are controlled by a data manager (DM). We assume each data item x has a data manager DM_x, for simplicity. Each DM_x is responsible for controlling access to the data item x and taking checkpoints of that data item periodically.

A transaction is a sequence of read and write operations on the data items in the database and terminates with a commit or an abort operation. Each transaction is managed by an instance of the transaction manager (TM) that forwards its operations to the scheduler which schedules transactions by using a specific concurrency control protocol. The TM with the help of the scheduler is responsible for the scheduling of the transactions in such a way that the integrity of the database is maintained.

A concurrency control algorithm ensures the scheduling of read and write operations issued by transactions in such a way that the execution of transactions is strict and serializable. The strictness property states that no data items may be read or written until the transaction that currently writes it either commits or aborts. Therefore a transaction actually writes a data item at its commit point atomically. A schedule is serial if the operations belonging to each transaction in the schedule appear

المنارة للاستشارات
www.manaraa.com

together in the schedule [1]. A schedule is serializable if the schedule has the effect equivalent to a schedule produced when transactions are run serially in some order. Ensuring strictness and serialization guarantees the integrity of the database as well as efficient execution of transactions in a distributed database system by promising atomicity, consistency, isolation, and durability, referred to as **ACID** properties [1].

- **A**tomicity: Each transaction is executed in its entirety, or not at all executed.
- **C**onsistency Preservation: Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
- **I**solation: Even though multiple transactions may execute concurrently, the system guarantees that for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.
- **D**urability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

In order to maintain ACID requirements and achieve good performance, a proper schedule of transactions in which the operations of various transactions are interleaved as much as possible needs to be determined. Given a schedule, a directed graph, referred to as precedence graph [2] or serialization graph [1], can be constructed to illustrate the dependency of all the transactions running in the database system. The serialization graph serves as an important tool to analyze transaction processing in distributed database systems.

In our model, each data item is checkpointed by a local transaction periodically. Before a transaction takes a checkpoint of a data item it obtains an exclusive lock on

www.manaraa.com

the data item and hence no other transaction can be accessing that data item while it is checkpointed. The state of a data item changes when a transaction accesses that data item for a write operation. Each checkpoint of a data item is assigned a unique sequence number. We assume that the database consists of a set of n data items $\mathbf{X} = \{x_i \mid 1 \leq i \leq n\}$. In addition, we denote by $C_i^{k_i}$ the checkpoint of x_i with sequence number k_i. The set of all checkpoints of data item x_i is denoted by $\mathbf{C_i} = \{C_i^{k_i} \mid k_i : k_i \geq 0\}$. The initial state of data item x_i is represented by checkpoint C_i^0 and a virtual checkpoint $C_i^{virtual}$ represents the last state reached by x_i after termination of all transactions accessing data item x_i. To minimize checkpointing overhead, a data item is checkpointed only if its state has changed since it was last checkpointed. That is, after a data item is checkpointed, it is not checkpointed again until at least one other transaction has accessed and changed that data item.

Let $\mathbf{T} = \{T_i \mid 1 \leq i \leq m\}$ be the set of all transactions that access the database. In order to make the analysis of the relationship between checkpoints of various data items simple, we assume that each checkpoint of a data item x_i is taken by a special *checkpointing transaction*. We denote by $T_{C_i^{k_i}}$ the checkpointing transaction that takes checkpoint $C_i^{k_i}$ of data item x_i. In order to maintain atomicity of transactions, $T_{C_i^{k_i}}$ is the local transaction which is required to be scheduled to access data item x_i when there are no other transactions accessing the data item, enforced by issuing an exclusive lock. The set of checkpointing transactions that produce the checkpoints $\mathbf{C_i}$ is denoted by $\mathbf{T_{C_i}}$ and the set of all checkpointing transactions of all data items in the database is denoted by $\mathbf{T_C}$.

A global checkpoint of the database is a set $\mathbf{S} = \{C_i^{k_i} \mid 1 \leq i \leq n\}$ of local checkpoints consisting of one checkpoint for each data item in the database. The set of checkpointing transactions that produce the global checkpoint $\mathbf{S}$ is denoted by $\mathbf{S_T} = \{T_{C_i^{k_i}} \mid 1 \leq i \leq n\}$. *We use $C_i^{k_i}$ and $T_{C_i^{k_i}}$ interchangeably. Sometimes, when we say a checkpoint of a data item we mean the checkpointing transaction that takes*

المنارة للاستشارات
www.manaraa.com

that checkpoint.

Each regular transaction is a partially ordered set of read/write operations (operations are partially ordered because two adjacent read operations in a transaction are not comparable and may be interchanged without affecting any result). A checkpointing transaction consists of only one operation (namely the *checkpointing operation*), an operation that requires mutually exclusive access to the data item. Let $R_i(x_j)$ (respectively, $W_i(x_j)$) denote the read (respectively, write) operation of T_i on data item $x_j \in \mathbf{X}$ and $O_{C_j^{k_j}}(x_j)$ denote the checkpointing operation of $T_{C_j^{k_j}}$ on data item x_j. A schedule ε over $\mathbf{T} \bigcup \mathbf{T_C}$ is a family of disjoint sets of partially ordered operations of transactions in $\mathbf{T} \bigcup \mathbf{T_C}$ on the data items (one set for each data item) [11]. Let $\varepsilon(x_j)$ consist of all read, write and checkpointing operations on x_j of transactions in $\mathbf{T} \bigcup \mathbf{T_C}$. We denote by $<_{x_j}$ the partial order induced by all read, write, and checkpointing operations on x_j by the schedule ε over $\mathbf{T} \bigcup \mathbf{T_C}$. Given a schedule ε over $\mathbf{T} \bigcup \mathbf{T_C}$, the relation $<_T$ between transactions in $\mathbf{T} \bigcup \mathbf{T_C}$ with respect to the schedule ε is defined as follows:

1) $T_i <_T T_j \Leftrightarrow (i \neq j) \wedge (\exists x_k \in X : (R_i(x_k) <_{x_k} W_j(x_k)) \vee (W_i(x_k) <_{x_k} W_j(x_k)) \vee (W_i(x_k) <_{x_k} R_j(x_k)))$.

2) $T_i <_T T_{C_j^{k_j}} \Leftrightarrow (W_i(x_j) <_{x_j} O_{C_j^{k_j}}(x_j)) \vee (R_i(x_j) <_{x_j} O_{C_j^{k_j}}(x_j))$.

3) $T_{C_i^{k_i}} <_T T_j \Leftrightarrow (O_{C_i^{k_i}}(x_i) <_{x_i} W_j(x_i)) \vee (O_{C_i^{k_i}}(x_i) <_{x_i} R_j(x_i))$.

The concurrency control protocol ensures that a schedule of transactions running in the distributed database system is serializable. One important kind of serialization, called conflict serializability (CSR) [1, 4] is considered in this context. A schedule $\varepsilon \in CSR$ iff the relation $<_T$ is acyclic [4]. A serialization order of a set of transactions with respect to a schedule ε over $\mathbf{T}$ is defined as a linear ordering of all the transactions such that if $T_i <_T T_j$ (either T_i or T_j could be a checkpointing transaction), then T_i

المنارة للاستشارات
www.manaraa.com

must appear before T_j in the ordering. If $\varepsilon \in CSR$, there must exist a serialization order for ε over $\mathbf{T}$ that is compatible with $<_T$.

In a distributed database system, when a failure occurs, the data items should be restored to a state which does not reflect the partial execution of any transaction, called a *transaction-consistent* global checkpoint. Formal definition of a *transaction-consistent* global checkpoint follows [11]:

Definition 1. *A global checkpoint of a distributed database is said to be* **transaction-consistent** *(tr-consistent or simply consistent, for short) with respect to the execution of a set of transactions* $\mathbf{T}$ *if there exists a serialization order (which is an ordering of transactions in* $\mathbf{T}$*)* $\sigma_1\sigma_2$ *for an execution* $\varepsilon \in CSR$ *of* $\mathbf{T}$ *such that the data item states represented by the global checkpoint is the same as those read by a read-only transaction* T_{CP} *after all transactions in* σ_1 *have finished execution and before any transaction in* σ_2 *has started execution.*

Checkpointing and rollback recovery are well-known techniques for handling failures in distributed systems including distributed database systems. The checkpointing protocols for distributed database systems can be classified as log-oriented and dump-oriented [13]. In the log-oriented approach, a dump of the database is taken periodically and also a marker is saved at appropriate places in the log. When a failure occurs, the latest dump is restored and the operations on the log after the dump was taken are applied to the dump until the next marker is reached to restore the database to a consistent state. In this approach, proper positioning of the marker in the log will result in restoring the database to a transaction-consistent global checkpoint. Protocols belonging to this class include [14] and [15]. In the dump-oriented approach, checkpointing is referred to as the process of saving the state of all data items in the database (or taking a dump of the database) in such a way that it represents a transaction-consistent global checkpoint of the database. The protocols proposed in [26, 27, 28] take this approach. The basic idea behind the protocol in [26]

المنارة للاستشارات
www.manaraa.com

is to divide the transactions into two groups: those transactions that arrived before and those that arrived after the checkpointing process. This protocol is non-intrusive but requires a copy of the database stored temporarily. From the concurrency control point of view, a protocol is intrusive means the protocol may intrude the normal execution of transactions causing re-scheduling or delay. Due to concurrent arrival of transactions, it may not be possible to decide whether a transaction has arrived before or after the checkpointing transaction. Such transactions are allowed to access the temporary copy while the checkpointing is in progress. Pu [27] uses coloring (white and black) to distinguish data items that have started checkpointing from data items that have not started checkpointing. Transactions accessing both white and black data items have to be aborted or delayed in order to maintain consistency, which increases transaction response time. Pilarski et al. [28] consider checkpoints as checkpoint transactions, one for each data item. In addition, a checkpoint number (CPN) is associated with each checkpoint. By comparing the CPN, forced checkpoints on data items are taken in order to make every checkpoint useful. Protocols presented in [26, 27] are coordinated protocols, in which one process initiates and coordinates the checkpointing activity.

Pilarski et al. [11] formally define the dependency relation caused by transactions among states of data items. They also analyze checkpointing in a distributed database system by establishing a correspondence between consistent snapshots of distributed computations in distributed systems and transaction-consistent checkpoints in a distributed database system. Moreover, they establish the sufficient conditions for a set of checkpoints to be part of a transaction-consistent global checkpoint. However, they do not establish the necessary and sufficient conditions for a set of checkpoints to be part of a transaction-consistent global checkpoint. In this dissertation, we establish the necessary and sufficient conditions for the checkpoints of a set of data items to be part of a transaction-consistent global checkpoint of the database.

المنارة للاستشارات
www.manaraa.com

Recently, many researchers have focussed on fuzzy checkpointing protocols [10, 23, 24] that write dirty pages to disk and require transaction logs for reconstructing a transaction-consistent state. Fuzzy checkpointing methods appear to be suitable for in-memory databases (IMDB), which store the data in RAM and back it up on the disk [12]. Fuzzy checkpointing does not obstruct the transaction processing but requires an undo/redo log to bring the state of the database from an inconsistent checkpoint back to a consistent state. Still, regular checkpointing approach is suitable for database systems in which the entire database need not be loaded into memory, and hence we focus on such databases only. The protocol proposed by Baldoni et al. [5] uses a non-coordinated approach, in which no process initiates checkpointing and each data item is checkpointed independently. Like the protocol of Pilarski et al. [28], checkpoint numbers are used to implicitly synchronize checkpointing and forced checkpoints are taken to prevent useless checkpoints. This protocol is fully distributed but may incur a large number of forced checkpoints, depending on the execution pattern of the transactions. Kumar et al. [9] present a performance evaluation of some existing recovery protocols for database systems.

1.1.2 Motivation

As we mentioned earlier, a transaction-consistent global checkpoint records a state of the database which reflects the effect of completed transactions only and not the results of any partially executed transactions. A straightforward way to take a transaction-consistent global checkpoint of a distributed database is to block all newly submitted transactions and wait until all the currently executing transactions finish and then take a checkpoint of the whole database. Such a checkpoint is guaranteed to be transaction-consistent, but this approach is intrusive and incurs large latency. Saving (checkpointing) the state of each data item independently and periodically without blocking any transaction (non-intrusive method), is more desirable.

المنارة للاستشارات
www.manaraa.com

However if each data item is checkpointed independently and periodically, we need an algorithm to determine tr-consistent global checkpoints from the checkpoints of individual data items. Moreover, when data items are checkpointed independently without any coordination, some (or all) checkpoints of the data items may not be part of any tr-consistent global checkpoint and hence such checkpoints are useless.

One of our goals is to develop the necessary and sufficient conditions for a checkpoint of a data item (or a set of checkpoints of a set of data items) to be part of a tr-consistent global checkpoint [59, 57]. The necessary and sufficient conditions can be applied in a straightforward way to determine which checkpoints are useless; further, it can be used to construct a tr-consistent global checkpoint starting from any useful checkpoint of any given data item.

Among the non-intrusive checkpointing protocols for distributed database systems, communication-induced checkpointing protocols are fairly new protocols and have potential to perform better than coordinated checkpointing protocols, since they are non-intrusive and have low-overhead. In communication-induced checkpointing protocols, basic checkpoints are taken independently and incrementally on each data item. In addition, forced checkpoints are taken in order to prevent useless checkpoints when information piggy-backed with commit operations of transactions satisfy certain conditions. Each data item can take checkpoints in a fully-distributed manner without any explicit coordination message. In addition, during recovery, each data item knows exactly to which checkpoint it should roll back since such a protocol guarantees the existence of a tr-consistent global checkpoint with the same sequence number. Transaction response time does not degrade in distributed database systems using such checkpointing protocols. Such checkpointing protocols also scale well since no coordination is involved during checkpointing. Such protocols were first presented in [5].

Like the protocol of Pilarski et al. in [28], protocols in [5] use checkpoint sequence

المنارة للاستشارات
www.manaraa.com

numbers to synchronize checkpointing. These protocols use ideas similar to the one used in the index-based communication-induced checkpointing protocol proposed by Manivannan and Singhal [33] for distributed systems. Due to the existence of transactions that require operations on many data items, Protocol 1 in [5] may incur large checkpointing overhead due to the large number of forced checkpoints. In this dissertation, we address this issue and develop a non-intrusive low-overhead checkpointing and recovery protocol for distributed database systems.

1.2 Checkpointing Distributed Computations

1.2.1 Background

A distributed system is a set of computers connected by a communication network. We model a distributed computation as a finite set P of n processes $P = \{P_1, P_2, ..., P_n\}$ running on a set of computers in a distributed system trying to accomplish a single task. We assume that each pair of processes is connected by a *reliable, asynchronous channel with unpredictable but finite transmission delays.* If a process fails it halts immediately without producing any erroneous result. This dissertation assumes that the failure of processes involved in a distributed computation follow the fail-stop model [16]. i.e., When a process fails it stops executing at the end of the last instruction that it has completed successfully and the contents of volatile storage are lost, but the contents of stable storage are preserved.

A process can have three types of events: internal, send and receive events. An internal event does not involve any communication. When P_i executes the event $send(m)$ to P_j, it sends the message m to P_j. When P_j executes the event $receive(m)$, it simply receives the message m directed to P_j from some process P_i. All the events of a distributed computation can be partially ordered with respect to the Lamport *"happened-before"* [32] relation $\stackrel{hb}{\rightarrow}$, defined next.

المنارة للاستشارات
www.manaraa.com

Definition 2. *The Lamport* **"happened before"** *relation* $\stackrel{hb}{\rightarrow}$ *on the set of events in a distributed computation is the transitive closure of the* $\stackrel{hb}{\rightarrow}$ *relation satisfying the following three conditions:*

1) If a and b are events of the same process and a comes before b, then $a \stackrel{hb}{\rightarrow} b$.

2) If a is the event send(m) and b is the corresponding event receive(m), then $a \stackrel{hb}{\rightarrow} b$.

3) If $a \stackrel{hb}{\rightarrow} b$ *and* $b \stackrel{hb}{\rightarrow} c$*, then* $a \stackrel{hb}{\rightarrow} c$.

Given a distributed computation $\widehat{H}$, its associated Checkpoint and Communication Pattern (CCP), denoted as $(\widehat{H}, C_{\widehat{H}})$, consists of the set of messages and the set of local checkpoints in $\widehat{H}$. Usually we denote by $C_{i,x}$, the checkpoint of process P_i with sequence number (or index number) x. The interval between $C_{i,x}$ and $C_{i,x+1}$, consisting of the sequence of events between these two checkpoints including the checkpointing event $C_{i,x}$ but excluding the checkpointing event $C_{i,x+1}$, is denoted as $I_{i,x}$.

If a process involved in a distributed computation fails and rolls back to a previous checkpoint, other processes may have to roll back to a previous checkpoint to maintain consistency. The set of checkpoints to which the processes involved in a distributed computation can be rolled back after a failure must record states of all the processes that could happen concurrently. A set C of checkpoints, one from each process involved in the distributed computation consisting of n processes $\{P_1, ..., P_n\}$, is globally consistent if and only if there is no message m from P_i to P_j (for any $i \neq j$) whose receiving event has been recorded in the checkpoint $C_{j,y} \in C$ but its corresponding sending event has not been recorded in the checkpoint $C_{i,x} \in C$. The set C itself is called a consistent global checkpoint of the distributed computation. Using Lamport "happened before" relation $\stackrel{hb}{\rightarrow}$, a consistent global checkpoint of a distributed computation can be defined as follows.

المنارة للاستشارات
www.manaraa.com

Definition 3. *A set S of checkpoints, one from each process involved in a distributed computation, is called a* **consistent global checkpoint** *of the distributed computation if for any two checkpoints $A, B \in S$, $A \overset{hb}{\not\rightarrow} B$.*

Rollback-Dependency Trackability

A Checkpoint and Communication Pattern satisfies Rollback-Dependency Trackability (RDT) if all rollback dependencies between local checkpoints are on-line trackable. Specifically, if a process P_j rolls back to a checkpoint $C_{j,y}$, any other process P_i must be able to determine the checkpoint $C_{i,x}$ to which it has to roll back so that the checkpoints to which all the processes roll back form a consistent global checkpoint. RDT was introduced first by Wang [44]. There are a number of checkpointing protocols that ensure RDT property which are characterized and summarized in [41]. An important concept introduced in the study of protocols that ensure RDT property is the concept of *visible doubling*, introduced by Baldoni et al. [35]. RDT property can be stated in terms of *Z-paths* introduced by Netzer and Xu [7] and the idea of *visible doubling* of Z-paths introduced by Baldoni *et al.* [35].

Definition 4. *A* **Z-path** *from $C_{i,x}$ to $C_{j,y}$ is a sequence of messages $([m_1, m_2, ..., m_q] : q \geq 1)$ such that,*

1. *$C_{i,x} \overset{hb}{\rightarrow} send(m_1)$, m_1 is sent by P_i;*
2. *for each i, $1 \leq i \leq q-1$, we have: $receive(m_i) \in I_{k,s} \wedge send(m_{i+1}) \in I_{k,t} \wedge s \leq t$;*
3. *m_q is received by P_j and $receive(m_q) \overset{hb}{\rightarrow} C_{j,y}$;*

A Z-path can also be defined between two checkpoint intervals. A Z-path from $I_{i,x}$ to $I_{j,y}$ exists iff a Z-path exists from $C_{i,x}$ to $C_{j,y+1}$. A Z-path is causal if the receiving event of each message (except for the last one) in the sequence $[m_1, m_2, ..., m_q]$ happens before the sending event of the next message in the sequence. A Z-path is non-causal

المنارة للاستشارات
www.manaraa.com

if it is not causal. A Z-path with only one message is trivially causal. For simplicity, a causal Z-path is also called a causal path. In this dissertation, we use ζ to represent a Z-path $\zeta = [m_1, ..., m_q]$ (also called a **message chain**). We denote by $\zeta.first$ ($\zeta.last$) to represent the first (last) message in the message chain. $|\zeta|$ denotes the number of messages in the message chain ζ. In addition, if a message chain ζ consists of two disjoint sub message chains ζ' and ζ'', ζ can also be represented as $\zeta'' + \zeta'$ and $\zeta - \zeta'$ is to represent ζ''. The following definitions are from Baldoni et al. [34, 35]:

Definition 5. *If a and b are checkpoints or message chains, we say a is* **causally concatenated** *to b, denoted as $a \circ b$, if:*

1) $a = C_{i,x}, b = \zeta$ and $\exists \nu \geq 0 : send(\zeta.first) \in I_{i,x+\nu}$; or

2) $a = \zeta, b = C_{i,x}$ and $\exists \nu > 0 : receive(\zeta.last) \in I_{i,x-\nu}$; or

3) $a = \zeta, b = \zeta'$ and $receive(\zeta.last) \xrightarrow{hb} send(\zeta'.first)$.

Definition 6. *A message chain ζ is* **non-causally concatenated** *to a message chain ζ' in the checkpoint interval $I_{k,y}$, denoted as $\zeta \overset{k,y}{\bullet} \zeta'$, if: $(receive(\zeta.last) \in I_{k,y}) \wedge (send(\zeta'.first) \in I_{k,y}) \wedge (send(\zeta'.first) \xrightarrow{hb} receive(\zeta.last))$.*

Definition 7. *A causal path μ from $I_{i,x}$ to P_j is* **prime** *if for every causal path μ' from $I_{i,x'}$ to P_j with $x \leq x'$, $receive(\mu.last) \xrightarrow{hb} receive(\mu'.last)$.*

Intuitively, a prime causal path from $I_{i,x}$ to P_j is the first casual path to P_j that includes the interval $I_{i,x}$ in P_j's causal past. Throughout this dissertation, we denote by $M(C_{i,x}, P_k)$ [34], the set of causal chains μ starting after $C_{i,x}$ such that the recipient of $\mu.last$ is P_k. As stated above, a causal chain μ is prime in $M(C_{i,x}, P_k)$ iff there does not exist another causal chain $\mu' \in M(C_{i,x}, P_k)$ such that $receive(\mu'.last) \xrightarrow{hb} receive(\mu.last)$.

المنارة للاستشارات
www.manaraa.com

Definition 8. *A* **PCM-path** $\mu \bullet m$ *is a Z-path that is the non-causal concatenation of a causal path* μ *and a single message* m*, where* μ *is prime and* $send(m) \xrightarrow{hb} receive(\mu.last)$ *in the same checkpoint interval [35].*

Definition 9. *A Z-path from* $I_{i,x}$ *to* $I_{j,y}$ *is* **causally doubled** *if* $i = j \wedge x \leq y$ *or if there exists a causal path* μ *from* $I_{i,x'}$ *to* $I_{j,y'}$ *where* $x \leq x'$ *and* $y' \leq y$ *[35].*

Definition 10. *A* PCM*-path* $\mu \bullet m$ *is* **visually doubled** *if and only if it is causally doubled by a causal path* μ' *with* $receive(\mu'.last) \xrightarrow{hb} send(\mu.last)$ *[35].*

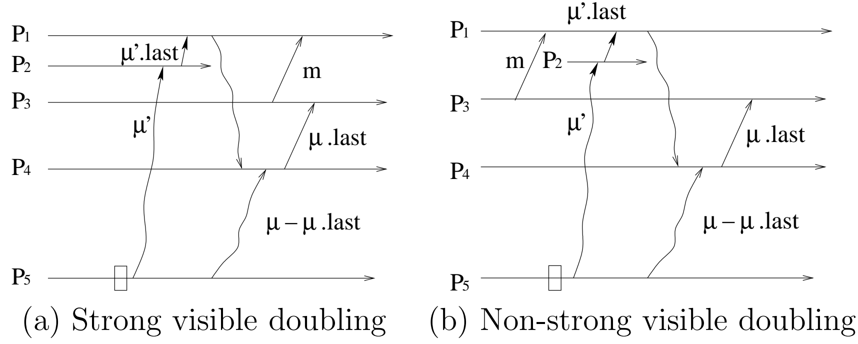
A causal doubling of the PCM-path $\mu \bullet m$ is visible to a process upon receiving $\mu.last$ only if the doubling path μ' belongs to the causal past of the sending event of $\mu.last$. A causally doubled PCM-path is not necessarily visibly doubled, but a non-causally doubled one must be non-visibly doubled. Then a characterization of RDT for online protocols based on causal history is stated in the following Theorem proved in [35].

Theorem 1.2.1. *A Checkpoint and Communication Pattern produced by a protocol based only on causal history satisfies the* RDT *property if and only if all* PCM*-paths are visibly doubled.*

Based on the concept of visible doubling, we developed our model-based checkpointing protocol. Even though visible doubling was introduced in the study of RDT, it helps in tracking communication patterns that can lead to Z-cycles. We are specifically interested in a subset of it, called *Strong Visible Doubling* which is defined as follows.

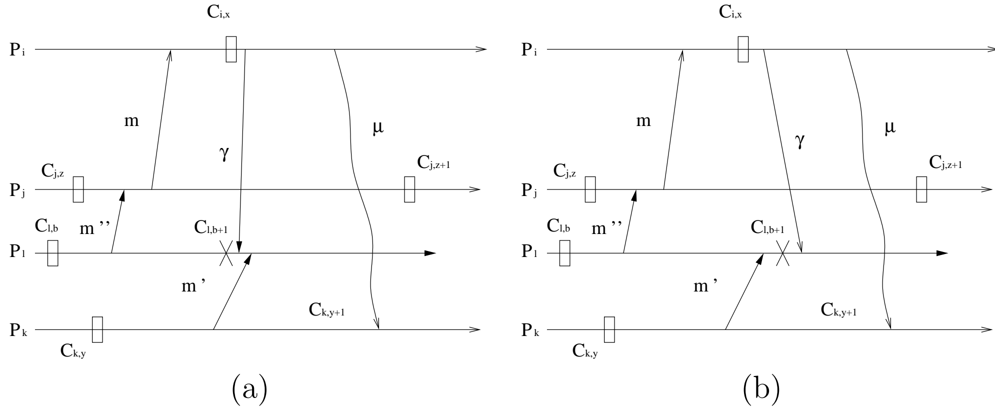
Definition 11. *A* PCM*-path* $\mu \bullet m$ *is* **strongly visibly doubled** *if it is causally doubled by a causal path* μ' *with* $receive(\mu'.last) \xrightarrow{hb} send(\mu.last)$ *and* $receive(\mu'.last) \xrightarrow{hb} receive(m)$*.*

المنارة للاستشارات
www.manaraa.com



(a) Strong visible doubling (b) Non-strong visible doubling

Figure 1.1: Visible doubling



(a) (b)

Figure 1.2: How strong visible doubling helps in reducing forced checkpoints

Figure 1.1(a) shows an example of strong visible doubling of the PCM-path $\mu \bullet m$ and Figure 1.1(b) shows an example of non-strong visible doubling of $\mu \bullet m$. The reason we distinguish these two situations can be explained using Figure 1.2. In Figure 1.2(a), the PCM-path $\mu \bullet m'$ is strongly visibly doubled by message γ. Then if the receiving of γ makes P_l take a forced checkpoint $C_{l,b+1}$, the Z-cycle (a Z-path from a checkpoint to itself) from $C_{i,x}$ to itself consisting of the message chain $\mu \bullet m' \bullet m'' \circ m$ is prevented. On the other hand, in Figure 1.2(b), $\mu \bullet m'$ is non-strongly visibly doubled. Even though the receiving of γ makes P_l take a forced checkpoint $C_{l,b+1}$, the Z-cycle from $C_{i,x}$ to itself due to the message chain $\mu \bullet m' \bullet m'' \circ m$ still exists. This example shows that these two situations make difference in preventing Z-cycles since the existence of $C_{l,b+1}$ prevents the Z-cycle in Figure 1.2(a) but not in Figure 1.2(b).

المنارة للاستشارات
www.manaraa.com

Finally, we describe the transitive dependency tracking mechanism for determining the existence of a causal path between events, in particular, between checkpoints. This is a well-known technique to track causality, which was proposed independently by Fidge [49] and Mattern [50]. Each process maintains and propagates a dependency vector of size n, where n is the number of processes involved in the distributed computation. By comparing the values of these vectors associated with the checkpoints, a process can determine the causal relation between checkpoints and make rollback decisions when a failure occurs. This dependency vector maintained by each process is also called the *vector clock*. Let VC_i be the *current* value of the vector clock of process P_i and $m.VC$ be the current value of the vector clock of the process sending the message m, piggybacked on message m. We denote by $VC(C_{i,x})$ the vector clock associated with the checkpoint $C_{i,x}$. All entries of VC_i are initialized to 0, except the ith entry which is initialized to one. At any time, $VC_i[i]$ represents the *current* checkpoint interval of P_i, and it is incremented by P_i every time a new checkpoint is taken. The current value of the vector clock is piggy-backed with every application message sent by each process. When a process receives a message from some other process, it sets its vector clock to be the componentwise maximum of its current value and the value received in the message. Thus, entry $VC_i[j]$ $(i \neq j)$ represents the highest checkpoint interval of P_j known to P_i. If VC_1 and VC_2 represent two vector clock values, they can be compared as follows [48].

1. **Equal:** $(VC_1 = VC_2) \equiv (\forall k, 1 \leq k \leq n, VC_1[k] = VC_2[k])$;
2. **Less Than or Equal:** $(VC_1 \leq VC_2) \equiv (\forall k, 1 \leq k \leq n, VC_1[k] \leq VC_2[k])$;
3. **Less Than:** $(VC_1 < VC_2) \equiv (VC_1 \leq VC_2) \wedge (VC_1 \neq VC_2)$;
4. **Concurrent:** $(VC_1 || VC_2) \equiv (VC_1 \not< VC_2) \wedge (VC_2 \not< VC_1)$;

It is a well known fact that there is a causal path from $C_{i,x}$ to $C_{j,y}$ if and only if $VC(C_{i,x}) \leq VC(C_{j,y})$. Therefore if the checkpointing and communication pattern

المنارة للاستشارات
www.manaraa.com

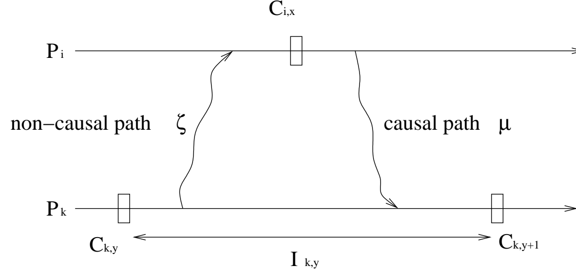


Figure 1.3: $Z - cycle$

is RDT, when process P_i rolls back to $C_{i,x}$, by comparing the vector clock values associated with the checkpoints, process P_j rolls back to its latest checkpoint $C_{j,y}$ such that $VC(C_{i,x})||VC(C_{j,y})$.

Z-cycles

If a Z-path exists from $C_{i,x}$ to $C_{i,x}$, we then have a Z-cycle involving $C_{i,x}$. In other words, a Z-cycle (ZC) [7], is a Checkpoint and Communication Pattern involving a checkpoint $C_{i,x}$ and a message chain $\widehat{\zeta}$ such that $\widehat{\zeta} \circ C_{i,x} \circ \widehat{\zeta}$. However, it is always possible to separate $\widehat{\zeta}$ into two non-empty sub-chains, a causal sub-chain μ and a sub-chain ζ such that $\widehat{\zeta} = \mu \overset{k,y}{\bullet} \zeta$ [34] (Figure 1.3). This observation gives rise to the following definition of Z-cycles [34]:

Definition 12. *A* **Z-cycle** *(ZC) is a Checkpoint and Communication Pattern, denoted by* $ZC(C_{i,x}, \mu \overset{k,y}{\bullet} \zeta)$ *such that* $\zeta \circ C_{i,x} \circ \mu \overset{k,y}{\bullet} \zeta$.

The size of a Z-cycle is the length of ζ (i.e., $|\zeta|$). Now let us introduce the "No-Z-cycle" property.

Definition 13. *A Checkpoint and Communication Pattern of a distributed computation* $(\widehat{H}, C_{\widehat{H}})$ *satisfies the* **"No-Z-cycle"** *property (NZC)* iff *no ZC exists in* $(\widehat{H}, C_{\widehat{H}})$.

Based on Netzer and Xu's result [7], if the CCP of a distributed computation $(\widehat{H}, C_{\widehat{H}})$ satisfies the "No-Z-cycle" property, every checkpoint is useful and hence can

المنارة للاستشارات
www.manaraa.com

be part of a consistent global checkpoint.

A key issue in checkpointing is how to maintain a *consistent global checkpoint* [31], from which a distributed computation can be restarted after a failure without losing much computation. Three checkpointing approaches have been proposed in the current literature, namely, Coordinated checkpointing, Uncoordinated checkpointing and Communication-Induced checkpointing.

Under Coordinated checkpointing, processes synchronize their checkpointing activities so that a consistent global checkpoint is always maintained in the system. The storage requirement of this checkpointing method is minimum since each process needs to keep at most two checkpoints: one committed and one possibly not committed. Major disadvantages include 1) process execution may have to be suspended during the synchronization, resulting in performance degradation and 2) it requires extra message overhead to synchronize the checkpointing activities.

Under Uncoordinated checkpointing, processes take local checkpoints periodically without any coordination with each other. This method allows maximum process autonomy for taking checkpoints and has no coordination message overhead. Under this approach, some or all checkpoints taken may lie on Z-cycles and hence are useless for the purpose of recovery.

Under Communication-Induced checkpointing, processes take checkpoints independently without any coordination while at the same time maintain necessary information to track checkpointing and communication patterns that could lead to the formation of Z-cycles in the future. When such communication patterns are detected, forced checkpoints are taken to prevent the formation of such Z-cycles in the future. Communication-Induced checkpointing protocols have been classified into two subclasses, namely, index-based and model-based protocols. Index-based protocols [38, 29] differ from Model-based protocols [34, 45, 52] in the way in which they capture the checkpoint and communication patterns that may lead to Z-cycles. Note

المنارة للاستشارات
www.manaraa.com

that Communication-Induced checkpointing methods track the checkpoint and communication patterns that may (and not necessarily will) lead to Z-cycles. Therefore, these protocols track communication patterns that are likely to result in the formation of Z-cycles (we call such patterns as *potential Z-cycles*) in the future and prevent them from happening in a pessimistic way. The Index-based protocols track the potential Z-cycles by comparing index numbers of checkpoints while the model-based protocols track a predefined checkpoint and communication patterns that are likely to result in Z-cycles. The Index-based protocols associate each local checkpoint with an index number and try to enforce consistency among local checkpoints with the same sequence number. In the model-based protocols, a communication pattern that could lead to Z-cycles is defined. The protocols then monitor and capture the formation of such communication patterns using some heuristics and take forced checkpoints to prevent the formation of such undesirable checkpoint and communication patterns.

1.2.2 Motivation

Existing model-based checkpointing protocols [34, 45, 52] track checkpoint and communication patterns by gathering information from the messages sent and received. Upon receiving a message from a process, a process can learn from the information piggybacked with the messages and information at hand whether the checkpointing and communication pattern complies with the model. All these three protocols have a common characteristic, namely, each process P_1 monitors the formation of potential Z-cycles based on local information ignoring what may happen at other processes, especially those to which P_1 has sent messages, say P_2, P_3. Sometimes, this unawareness may leave process P_1 having erroneous information about the formation of a potential Z-cycle solely based on local information. Later sections illustrate the situation under which a process P_1 may detect the formation of a potential Z-cycle under the algorithms in [34, 45, 52], even though there is no possibility for Z-cycles to form.

المنارة للاستشارات
www.manaraa.com

The reason for these false detections are due to P_1 not maintaining and using information about processes to which it has sent messages. We show in later sections that if all those processes to which a process has sent messages (P_2 and P_3) have already detected potential Z-cycles, P_1 does not need to take any action even if a potential Z-cycle is detected by itself. Based on this intuition, our goal is to develop an efficient model-based communication-induced checkpointing protocol which makes use of not only the information piggy-backed with messages received from other processes but also information about processes to which a process has sent messages.

1.3 Contribution of the Thesis

Our contribution lies in two areas, namely, (i) checkpointing and recovery in distributed database systems and (ii) checkpointing in distributed systems. We define the notion of zigzag paths[1] between checkpoints of data items in a distributed database system and establish the necessary and sufficient conditions for a set of checkpoints of a set of data items to be part of a tr-consistent global checkpoint. This condition helps in constructing tr-consistent global checkpoint of a distributed database incrementally, starting from a checkpoint of any given data item. Furthermore, this condition can help in designing and evaluating non-intrusive checkpointing protocols for distributed database systems. We also developed a non-intrusive checkpointing protocol for distributed database systems that has low checkpointing overhead and low recovery overhead. The protocol makes use of the concept of logical checkpoints to reduce the number of checkpoints taken.

Our second contribution is the development of an efficient communication-induced checkpointing protocol for distributed systems. As we mentioned earlier, communication-induced checkpointing protocols eliminate useless checkpoints by tracking communication patterns that could result in the formation of Z-cycles and forcing processes to

[1]This is a generalization of the concept of zigzag paths introduced by Netzer and Xu [7] in the context of checkpointing distributed computations.

www.manaraa.com

take checkpoints in addition to the independently taken checkpoints to prevent the formation of Z-cycles. It is well known that Z-cycles are not on-line trackable. So, different protocols use different heuristics to prevent Z-cycles. We introduce the concept of Extended Suspect Z-cycles, a communication pattern that could lead to the formation of Z-cycles in the future. Extended Suspect Z-cycles are on-line trackable and preventable if forced checkpoints are taken at appropriate times. We prove that the system will have no Z-cycles if there are no Extended Suspect Z-cycles. Then, we develop a communication-induced checkpointing protocol which tracks the communication patterns and detects and prevents the formation of Extended Suspect Z-cycles. Our checkpointing protocol has lower checkpointing overhead compared to an existing checkpointing protocol in this category.

1.4 Organization of the Dissertation

The remaining part of the dissertation is organized as follows. In Chapter 2, we generalize the concept of zigzag paths, proposed by Netzer and Xu [7] between checkpoints of processes involved in a distributed computation, to checkpoints of data items of a distributed database system and establish the necessary and sufficient conditions that a checkpoint of a data item (or a set of checkpoints from a set of data items) needs to satisfy in order to be part of a transaction-consistent global checkpoint of the database [59, 57]. In Chapter 3, we present a non-intrusive checkpointing protocol for distributed database systems [58]. In Chapter 4, we present our model-based communication-induced checkpointing protocol for distributed systems [51, 53] which reduces the numbers of forced checkpoints compared to an existing protocol. Chapter 5 summarizes the contribution of the thesis and concludes with a discussion of future research work.

المنارة للاستشارات
www.manaraa.com

Chapter 2

Necessary and Sufficient Conditions for Transaction-Consistent Global Checkpoints of a Distributed Database System

In a distributed system, to minimize the lost computation due to failures, the state of the processes involved in a distributed computation is periodically saved in stable storage (checkpointed). When one or more processes involved in the distributed computation fails, the processes are restarted from the latest consistent global checkpoint. If processes are independently and periodically checkpointed, one or more checkpoints taken may not be part of any consistent global checkpoint and hence such checkpoints are useless [7]. Netzer and Xu [7] introduced the notion of zigzag paths between checkpoints of processes involved in a distributed computation and established the necessary and sufficient conditions for a given checkpoint of a process to be part of a consistent global checkpoint (i.e., useful). They proved that a checkpoint of a process is useful if and only if there is no zigzag path from that checkpoint to itself.

Checkpointing is also an established technique for handling failures in database systems. Many of the checkpointing schemes proposed in the literature for distributed database systems are intrusive to different extent. Non-intrusive checkpointing protocols under which transactions do not have to be blocked when checkpoints are taken are desirable [30]. If each data item in a distributed database is checkpointed by an independent transaction periodically, it is quite possible that none of the checkpoints taken is part of any transaction-consistent global checkpoint of the database. In this

المنارة للاستشارات
www.manaraa.com

chapter, motivated by the work of Netzer and Xu for distributed computation [7], we generalize the concept of zigzag paths to distributed databases and establish the necessary and sufficient conditions for a given checkpoint of a data item (or checkpoints of a set of data items) to be part of a transaction-consistent global checkpoint. Next, we introduce some terminology.

2.1 Serialization Graphs

Following the terminology introduced in Chapter 1, let $\mathbf{T} = \{T_i \mid 1 \leq i \leq m\}$ be a set of transactions that access the database. Let $\mathbf{X} = \{x_i \mid 1 \leq i \leq n\}$ be the set of data items in the distributed database. We assume that each checkpoint of a data item x_i is taken by a special transaction called *checkpointing transaction*. We denote by $T_{C_i^{k_i}}$ the checkpointing transaction that takes checkpoint $C_i^{k_i}$ of data item x_i. The set of all checkpoints of data item x_i is denoted by $\mathbf{C_i} = \{C_i^{k_i} \mid k_i : k_i \geq 0\}$. The set of checkpointing transactions that produce the global checkpoint $\mathbf{S}$ is denoted by $\mathbf{S_T} = \{T_{C_i^{k_i}} \mid 1 \leq i \leq n\}$. We use $C_i^{k_i}$ and $T_{C_i^{k_i}}$ interchangeably. Let T_C be the set of all checkpointing transactions.

If the concurrency control algorithm guarantees an execution $\varepsilon \in CSR$, then the corresponding relation $<_T$ induces a directed acyclic graph (Dag) structure on $\mathbf{T} \bigcup \mathbf{T_C}$ and conversely [4]. We call this graph the *global serialization graph* with respect to the schedule ε of $\mathbf{T} \bigcup \mathbf{T_C}$. For each data item, the transactions accessing that data item induce a subgraph of the global serialization graph. The *local serialization graph* induced by the transactions in $\mathbf{T} \bigcup \mathbf{T_C}$ that access data item x_i is denoted by $G_{x_i}(V_{x_i}, E_{x_i})$: the vertex set $V_{x_i} = \{T_k \cup T_{C_i^{k_i}} \mid T_k \in \mathbf{T}$ has accessed data item x_i; $C_i^{k_i}$ is the k_i^{th} checkpoint of x_i taken by local checkpoint transaction $T_{C_i^{k_i}}\}$ and the edge set $E_{x_i} = \{E_{x_i}^{TT} \cup E_{x_i}^{TC} \cup E_{x_i}^{CT}\}$, where

1: $E_{x_i}^{TT} = \{(T_i, T_j) \mid T_i, T_j \in V_{x_i}; T_i <_T T_j\}$.

المنارة للاستشارات
www.manaraa.com

2: $E_{x_i}^{TC} = \{(T_j, T_{C_i^{k_i}}) \mid T_j, T_{C_i^{k_i}} \in V_{x_i};\ T_j <_T T_{C_i^{k_i}}\}$.

3: $E_{x_i}^{CT} = \{(T_{C_i^{k_i}}, T_j) \mid T_j, T_{C_i^{k_i}} \in V_{x_i};\ T_{C_i^{k_i}} <_T T_j\ \}$.

By merging the local serialization graphs $G_{x_i}(V_{x_i}, E_{x_i})$, we can construct the global serialization graph $G(V, E)$ where

$$V = \bigcup_{x_i \in X} V_{x_i}$$

and

$$E = \bigcup_{x_i \in X} E_{x_i}.$$

Next we illustrate the construction of the global serialization graph with an example. Suppose we have the following nine transactions $\mathbf{T} = \{T_1, \cdots, T_9\}$ accessing a database containing five data items $X = \{x_1, \cdots, x_5\}$.

1. $T_1 : R_1(x_5), W_1(x_5)$
2. $T_2 : W_2(x_2), W_2(x_4)$
3. $T_3 : R_3(x_1), W_3(x_1), W_3(x_2), W_3(x_4)$
4. $T_4 : W_4(x_3), W_4(x_1), W_4(x_4), R_4(x_5)$
5. $T_5 : R_5(x_3), R_5(x_4)$
6. $T_6 : W_6(x_3), W_6(x_4)$
7. $T_7 : W_7(x_2), R_7(x_2)$
8. $T_8 : R_8(x_2), W_8(x_2)$
9. $T_9 : W_9(x_1), W_9(x_5)$

المنارة للاستشارات
www.manaraa.com

Consider the schedule $\varepsilon \in CSR$ over $(\mathbf{T} \bigcup \mathbf{T_C})$ where

$\varepsilon = \{O_{C_1^0}(x_1),\ O_{C_2^0}(x_2),\ O_{C_3^0}(x_3),\ O_{C_4^0}(x_4),\ O_{C_5^0}(x_5),\ W_2(x_2),\ W_9(x_1),\ R_3(x_1), O_{C_2^1}(x_2), W_4(x_3), W_2(x_4), O_{C_4^1}(x_4), W_3(x_1), O_{C_1^1}(x_1), W_4(x_1), W_9(x_5), O_{C_5^1}(x_5), W_4(x_4), O_{C_4^2}(x_4), W_3(x_2), R_5(x_3), R_5(x_4), W_6(x_3), W_6(x_4), W_3(x_4), O_{C_2^2}(x_2), W_7(x_2), R_1(x_5), W_1(x_5), R_8(x_2), R_7(x_2), W_8(x_2), R_4(x_5)\}$.

This schedule induces the following partial order on the operations performed by the transactions on each data item:

1. $\varepsilon(x_1) : O_{C_1^0}(x_1) <_{x_1} W_9(x_1) <_{x_1} R_3(x_1) <_{x_1} W_3(x_1) <_{x_1} O_{C_1^1}(x_1) <_{x_1} W_4(x_1)$
2. $\varepsilon(x_2) : O_{C_2^0}(x_2) <_{x_2} W_2(x_2) <_{x_2} O_{C_2^1}(x_2) <_{x_2} W_3(x_2) <_{x_2} O_{C_2^2}(x_2) <_{x_2} W_7(x_2) <_{x_2} R_8(x_2) <_{x_2} R_7(x_2) <_{x_2} W_8(x_2)$
3. $\varepsilon(x_3) : O_{C_3^0}(x_3) <_{x_3} W_4(x_3) <_{x_3} R_5(x_3) <_{x_3} W_6(x_3)$
4. $\varepsilon(x_4) : O_{C_4^0}(x_4) <_{x_4} W_2(x_4) <_{x_4} O_{C_4^1}(x_4) <_{x_4} W_3(x_4) <_{x_4} W_4(x_4) <_{x_4} O_{C_4^2}(x_4) <_{x_4} R_5(x_4) <_{x_4} W_6(x_4)$
5. $\varepsilon(x_5) : O_{C_5^0}(x_5) <_{x_5} W_9(x_5) <_{x_5} O_{C_5^1}(x_5) <_{x_5} R_1(x_5) <_{x_5} W_1(x_5) <_{x_5} R_4(x_5)$

This schedule induces the following relations among the transactions. These relations correspond to edges in the global serialization graph constructed from the local serialization graphs.

$T_{C_1^0} <_T T_9, T_9 <_T T_3, T_3 <_T T_{C_1^1}, T_{C_1^1} <_T T_4, T_{C_2^0} <_T T_2, T_2 <_T T_{C_2^1}, T_{C_2^1} <_T T_3, T_3 <_T T_{C_2^2}, T_{C_2^2} <_T T_7, T_7 <_T T_8, T_{C_3^0} <_T T_4, T_4 <_T T_5, T_5 <_T T_6, T_{C_4^0} <_T T_2, T_2 <_T T_{C_4^1}, T_{C_4^1} <_T T_3, T_3 <_T T_{C_4^2}, T_{C_4^2} <_T T_4, T_4 <_T T_5, T_5 <_T T_6, T_{C_5^0} <_T T_9, T_9 <_T T_{C_5^1}, T_{C_5^1} <_T T_1, T_1 <_T T_4$.

The local serialization graphs induced by the partial orders $\varepsilon(x_i)$ on the data items are shown in Figures 2.1 to 2.3. The global serialization graph constructed from the local graphs is shown in Figure 2.4. The global serialization graph $G = (V, E)$ is obtained by merging the local serialization graphs where

www.manaraa.com

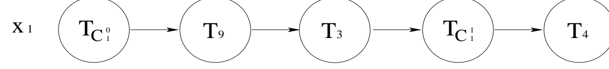


Figure 2.1: Local serialization graph induced by transactions $\{T_9, T_3, T_4\}$ on data item x_1.

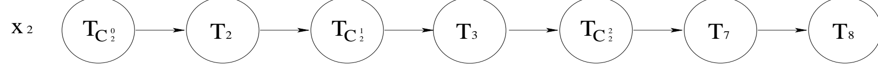


Figure 2.2: Local serialization graph induced by transactions $\{T_2, T_3, T_7, T_8\}$ on data item x_2.

$V = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{C_1^0}, T_{C_2^0}, T_{C_3^0}, T_{C_4^0}, T_{C_5^0},$

$T_{C_1^1}, T_{C_2^1}, T_{C_2^2}, T_{C_4^1}, T_{C_4^2}, T_{C_5^1}\}$

and

$E = \{(T_{C_1^0}, T_9), (T_{C_2^0}, T_2), (T_{C_3^0}, T_4), (T_{C_4^0}, T_2), (T_{C_5^0}, T_9), (T_9, T_3), (T_3, T_{C_1^1}), (T_{C_1^1}, T_4),$

$(T_2, T_{C_2^1}), (T_{C_2^1}, T_3), (T_3, T_{C_2^2}), (T_{C_2^2}, T_7), (T_7, T_8), (T_2, T_{C_4^1}), (T_{C_4^1}, T_3), (T_3, T_{C_4^2}),$

$(T_{C_4^2}, T_4), (T_4, T_5), (T_{C_5^0}, T_9), (T_9, T_{C_5^1}), (T_{C_5^1}, T_1), (T_1, T_4), (T_5, T_6)\}.$

The graph in Figure 2.4 is acyclic and hence the schedule $\in CSR$. Since $\varepsilon \in CSR$, we have the following serialization order that is compatible with $<_T$. This ordering may not be unique because some transactions in this can be reordered without violating $<_T$. For example, T_2 and T_9 can be interchanged in the order.

$T_{C_1^0}, T_{C_2^0}, T_{C_3^0}, T_{C_4^0}, T_{C_5^0}, T_2, T_9, T_{C_2^1}, T_{C_4^1}, T_3, T_{C_5^1}, T_1, T_{C_2^2}, T_{C_1^1}, T_{C_4^2}, T_4, T_5, T_7, T_8, T_6$

We use the following notations in this chapter: $T_i \longrightarrow^+ T_j$ iff there is a path from transaction T_i to T_j in the serialization graph (T_i and/or T_j could be a checkpointing transaction). $T_i \longrightarrow T_j$ iff there is an edge from T_i to T_j (T_i or T_j could be a checkpointing transaction). Let $\sigma_1 \subseteq \mathbf{T}$ and $\sigma_2 \subseteq \mathbf{T}$ be such that $\sigma_1 \bigcap \sigma_2 = \phi$. Then, by $\sigma_1 \mathbf{S_T} \sigma_2$ with respect to the serialization order induced by conflict-serializable execution ε over $\mathbf{T}$, we mean that each checkpointing transaction in $\mathbf{S_T}$ starts executing only after every transaction in σ_1 has been executed and before

المنارة للاستشارات
www.manaraa.com

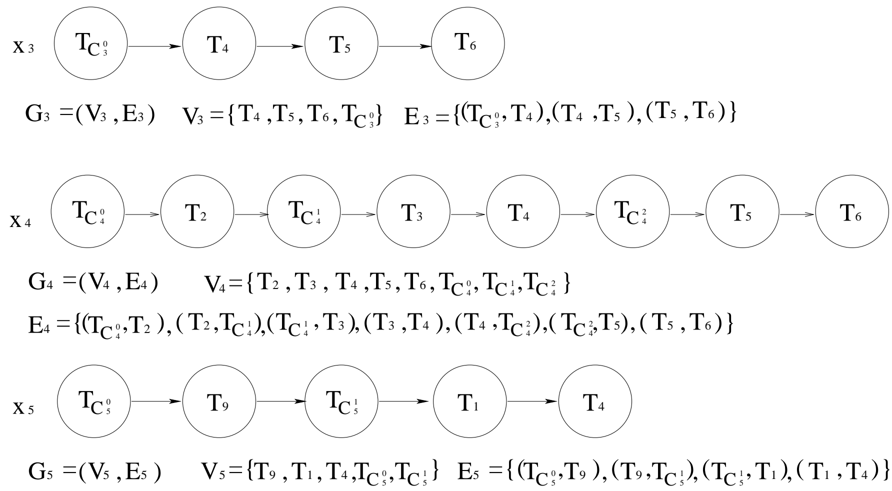


Figure 2.3: Local serialization graphs induced by the transactions on the data items x_3, x_4, and x_5.

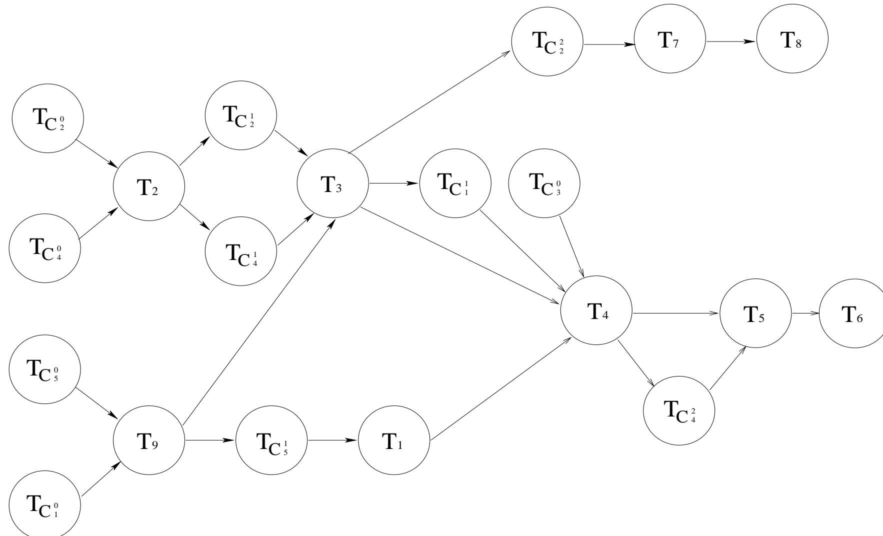


Figure 2.4: Global serialization graph constructed from local serialization graphs.

المنارة للاستشارات
www.manaraa.com

any transaction in σ_2 has started execution. In particular, if $\sigma_1 \bigcup \sigma_2 = \mathbf{T}$, then the set of checkpoints $\mathbf{S}$ taken by $\mathbf{S_T}$ is tr-consistent iff $\sigma_1 \mathbf{S_T} \sigma_2$.

Next, we make the following observations:

Observation 1. *For any checkpointing transaction $T_{C_i^{k_i}}$, since it accesses the data item x_i exclusively, $T_{C_i^{k_i}}$ must have a path in the* local serialization graph *either to or from any transaction T_j that has accessed x_i.*

Observation 2. *For any checkpointing transaction $T_{C_i^{k_i}}$, since it accesses the data item x_i exclusively, if there exist two transactions T_i and T_j that access x_i such that $T_i \longrightarrow^+ T_{C_i^{k_i}}$, $T_{C_i^{k_i}} \longrightarrow^+ T_j$, then in the* local serialization graph *induced by $\mathbf{T} \cup \mathbf{T_C}$ on the data item x_i, any path from T_i to T_j must pass through $T_{C_i^{k_i}}$.*

Observation 3. *In the* local serialization graph *induced by $\mathbf{T} \cup \mathbf{T_C}$ on the data item x_i, for any checkpointing transaction $T_{C_i^{k_i}}$ and two other transactions T_i and T_j that have accessed x_i, the following holds:*

1. *If $T_{C_i^{k_i}} \longrightarrow^+ T_j$ and there exists $T_i \longrightarrow^+ T_j$ without any checkpoint transaction along the path in the local serialization graph, then $T_{C_i^{k_i}} \longrightarrow^+ T_i$.*

2. *Similarly, if $T_i \longrightarrow^+ T_{C_i^{k_i}}$ and there exists a path $T_i \longrightarrow^+ T_j$ from T_i to T_j without any checkpoint transaction along the path in the local serialization graph, then $T_j \longrightarrow^+ T_{C_i^{k_i}}$.*

Observations 1 and 2 are trivial. Observation 3 holds because under case 1, suppose $T_{C_i^{k_i}} \longrightarrow^+ T_i$ is not true, then $T_i \longrightarrow^+ T_{C_i^{k_i}}$ from Observation 1. Since $T_{C_i^{k_i}} \longrightarrow^+ T_j$, from Observation 2, every path in the local serialization graph from T_i to T_j must pass through $T_{C_i^{k_i}}$, which contradicts our assumption that there exists a path $T_i \longrightarrow^+ T_j$ without any checkpointing transactions along the path. A similar argument can be used to prove the correctness of case 2 in Observation 3.

المنارة للاستشارات
www.manaraa.com

We make use of these Observations in the proofs of the theorems in Section 2.2. Notice that the transactions T_i, T_j in the previous observations could be checkpointing transactions as well.

2.2 Necessary and Sufficient Conditions

In distributed database systems, it would be ideal if individual data items could be checkpointed without any coordination and a tr-consistent global checkpoint could be constructed from the checkpoints of the individual data items whenever it is needed for recovery. To construct such a global checkpoint, we need to know what checkpoints could be combined to construct a tr-consistent global checkpoint. In the following theorem, we establish the necessary and sufficient condition for a set of checkpoints, one from *each* data item (i.e., a global checkpoint of the database) to form a tr-consistent global checkpoint of the database with respect to a given schedule of a given set of transactions. We assume that each data item has a virtual checkpoint which represents the final state of the data item after all the transactions finish execution.

Theorem 2.2.1. *Let $\mathbf{T} = \{T_1, \cdots, T_m\}$ be a set of transactions accessing the database consisting of n data items $\mathbf{X} = \{x_1, \cdots, x_n\}$. Assume that each data item is checkpointed by a checkpointing transaction that runs at the site which contains the data item. Let $\mathbf{S} = \{C_i^{k_i} \mid 1 \leq i \leq n\}$ be a set of checkpoints, one for each data item and let $\mathbf{S_T} = \{T_{C_i^{k_i}} \mid 1 \leq i \leq n\}$ be the set of checkpointing transactions that produce $\mathbf{S}$. Let ε be a conflict serializable schedule over $\mathbf{T} \bigcup \mathbf{S_T}$. Then $\mathbf{S}$ is a tr-consistent global checkpoint iff there is no path between any two checkpointing transactions belonging to $\mathbf{S_T}$ in the global serialization graph corresponding to the schedule ε.*

Proof: (If Part) Suppose there is no path between any two checkpointing transactions in $\mathbf{S_T}$ in the global serialization graph. Then we prove that the set $\mathbf{S}$ forms a

المنارة للاستشارات
www.manaraa.com

tr-consistent global checkpoint. It is sufficient to prove that there exist a serialization order $\sigma_1\sigma_2$ of $\mathbf{T}$ with respect to ε such that $\sigma_1\mathbf{S_T}\sigma_2$, i.e., each checkpointing transaction in $\mathbf{S_T}$ is executed only after every transaction in σ_1 has finished execution and before any transaction in σ_2 starts execution.

We say $T_i \longrightarrow^+ \mathbf{S_T}$ if there exists a path from T_i to some checkpointing transaction in $\mathbf{S_T}$. Similarly, we say $\mathbf{S_T} \longrightarrow^+ T_i$ if there exists a path from some checkpointing transaction in $\mathbf{S_T}$ to T_i. Any transaction in $\mathbf{T}$ belongs to at least one of the following three sets.

1) $\sigma_a = \{T_i \in \mathbf{T} \mid T_i \longrightarrow^+ \mathbf{S_T}\}$

2) $\sigma_b = \{T_i \in \mathbf{T} \mid \mathbf{S_T} \longrightarrow^+ T_i\}$

3) $\sigma_c = \{T_i \in \mathbf{T} \mid \text{neither } T_i \longrightarrow^+ \mathbf{S_T} \text{ nor } \mathbf{S_T} \longrightarrow^+ T_i\}$

From Observation 1, we know that $\sigma_c = \phi$ since T_i must access at least one data item. In addition, we have $\sigma_a \bigcup \sigma_b \bigcup \sigma_c = \mathbf{T}$. Since $\sigma_c = \phi$, $\sigma_a \bigcup \sigma_b = \mathbf{T}$

Let $T_v \in \sigma_a$, then $T_v \longrightarrow^+ \mathbf{S_T}$ by definition. In particular $T_v \longrightarrow^+ T_{C_i^{k_i}}$ for some i, which means that T_v has finished accessing x_i before $T_{C_i^{k_i}}$ takes checkpoint on data item x_i.

Claim: For every data item x_j, T_v must have finished accessing x_j before $T_{C_j^{k_j}} \in \mathbf{S_T}$ starts execution.

Proof of claim: The following three cases arise.

1) $T_v \longrightarrow^+ T_{C_j^{k_j}}$. This means T_v has finished accessing x_j before $T_{C_j^{k_j}}$ takes checkpoint on x_j.

2) $T_{C_j^{k_j}} \longrightarrow^+ T_v$. This case cannot arise since $T_v \longrightarrow^+ T_{C_i^{k_i}}$ we have $T_{C_j^{k_j}} \longrightarrow^+ T_{C_i^{k_i}}$, a contradiction to the assumption that there is no path between any two checkpointing transactions in $\mathbf{S_T}$.

المنارة للاستشارات
www.manaraa.com

3) Neither $T_v \longrightarrow^+ T_{C_j^{k_j}}$ nor $T_{C_j^{k_j}} \longrightarrow^+ T_v$. From Observation 1, T_v does not access x_j. In this case we can simply treat T_v as a transaction that has executed before $T_{C_j^{k_j}}$ has started.

Therefore T_v must have finished accessing every data item x_j (that it needs to access) before $T_{C_j^{k_j}}$ starts execution. This proves our claim. So, each transaction in σ_a finishes execution before any of the checkpointing transactions in $\mathbf{S_T}$ has started execution.

Similarly, we can prove that each transaction $T_v \in \sigma_b$ starts accessing any data item x_j only after checkpointing transaction $T_{C_j^{k_j}} \in \mathbf{S_T}$ has finished execution.

Let $\sigma_1 = \sigma_a$, the set of all transactions that have finished execution before none of the checkpointing transactions in $\mathbf{S_T}$ has started execution. Let $\sigma_2 = \sigma_b$, the set of all transactions that have started execution after every checkpointing transaction in $\mathbf{S_T}$ has finished execution. We have $\sigma_1 \bigcup \sigma_2 = \mathbf{T}$ and $\sigma_1 \bigcap \sigma_2 = \sigma_a \bigcap \sigma_b$. Moreover, $(\sigma_a \bigcap \sigma_b) = \phi$, because if $(\sigma_a \bigcap \sigma_b) \neq \phi$, let $T_i \in (\sigma_a \bigcap \sigma_b)$. Then, by definition of σ_a and σ_b, there exists $T_{C_v^{k_v}}, T_{C_w^{k_w}} \in \mathbf{S_T}$, such that $T_i \longrightarrow^+ T_{C_v^{k_v}}$, and $T_{C_w^{k_w}} \longrightarrow^+ T_i$. Hence $T_{C_w^{k_w}} \longrightarrow^+ T_{C_v^{k_v}}$, a contradiction to the assumption that there is no path between any two checkpointing transactions in $\mathbf{S_T}$. Therefore, we have $\sigma_1 \mathbf{S_T} \sigma_2$.

(Only-if Part) Conversely, suppose $\mathbf{S}$ is a tr-consistent global checkpoint, then we prove that no two transactions in $\mathbf{S_T}$ have a path between them in the global serialization graph. Suppose there is a path from $T_{C_i^{k_i}} \in \mathbf{S_T}$ to $T_{C_j^{k_j}} \in \mathbf{S_T}$. Then there exists a transaction $T_{c_1} \in \mathbf{T}$ such that $T_{C_i^{k_i}} \longrightarrow^+ T_{c_1} \longrightarrow T_{C_j^{k_j}}$.

First we show that T_{c_1} starts execution after every checkpointing transaction in $\mathbf{S_T}$ has finished. Because of the path $T_{C_i^{k_i}} \longrightarrow^+ T_{c_1}$, we know that T_{c_1} must start execution after $T_{C_i^{k_i}} \in \mathbf{S_T}$ has finished. Since $T_{C_i^{k_i}} \in \mathbf{S_T}$, where $\mathbf{S_T}$ produces a tr-consistent global checkpoint S, by definition of tr-consistent global checkpoint, besides $T_{C_i^{k_i}}$, T_{c_1} must start execution after every other checkpointing transaction $T_{C_v^{k_v}} \in \mathbf{S_T}$, where $v \neq i$, finishes execution. Therefore T_{c_1} starts execution after every checkpointing transaction in $\mathbf{S_T}$ has finished. On the other hand, on x_j, T_{c_1} has

المنارة للاستشارات
www.manaraa.com

started execution before $T_{C_j^{k_j}} \in \mathbf{S_T}$ has started due to the edge $T_{c_1} \longrightarrow T_{C_j^{k_j}}$. This implies $\mathbf{S_T}$ is not a tr-consistent global checkpoint.

Hence a global checkpoint $\mathbf{S}$ is tr-consistent with respect to a serializable schedule of a set of transactions iff there is no path between any two checkpointing transactions in $\mathbf{S_T}$ in the global serialization graph corresponding to the schedule. $\diamond$

Theorem 2.2.1 is useful for verifying whether a given global checkpoint is tr-consistent. For instance, in Figure 2.4, $\mathbf{S} = \{T_{C_1^0}, T_{C_2^1}, T_{C_3^0}, T_{C_4^1}, T_{C_5^0}\}$ forms a tr-consistent global checkpoint because no two elements in $\mathbf{S}$ have a path between them. However, this theorem does not help in constructing a tr-consistent global checkpoint incrementally. This is because if there is no path between two checkpoints of two different data items, it does not mean that these two checkpoints together can be part of a tr-consistent global checkpoint. For example, in Figure 2.4, there is no path between $T_{C_5^1}$ and $T_{C_2^2}$. However, checkpoints C_5^1 and C_2^2 cannot belong to a tr-consistent global checkpoint because data item x_4 does not have a checkpoint that can be combined with C_5^1 and C_2^2 to extend it to a tr-consistent global checkpoint. For instance, C_4^1 cannot be used because there is a path from $T_{C_4^1}$ to $T_{C_2^2}$ and the remaining checkpoints of x_4 cannot be used for similar reasons. Therefore, additional restrictions need to be added in order to be able to extend a given set of checkpoints to a tr-consistent global checkpoint. As mentioned earlier, our goal is to come up with the necessary and sufficient conditions for a set of checkpoints of a set of data items to be part of a tr-consistent global checkpoint. The next theorem addresses this problem. For that we need to introduce some new terminology.

Next, we introduce some terminology for developing the necessary and sufficient conditions for a set of checkpoints to be part of a tr-consistent global checkpoint. Netzer and Xu [7] introduced the concept of zigzag paths between checkpoints of a distributed computation and used it to establish the necessary and sufficient conditions for a set of checkpoints of a set of processes involved in a distributed computation

المنارة للاستشارات
www.manaraa.com

to be part of a consistent global checkpoint of the computation. We generalize their definition of zigzag paths to checkpoints in distributed database systems and use it for establishing the necessary and sufficient conditions for a set of checkpoints of a set of data items to be part of a tr-consistent global checkpoint of the database.

Definition 14. *Let* $\mathbf{T}$ *be a set of transactions executing on a database. Let* $C_i^{k_i}$ *be a checkpoint taken by the checkpointing transaction* $T_{C_i^{k_i}}$ *on data item* x_i*, and let* $C_j^{k_j}$ *be another checkpoint taken by checkpointing transaction* $T_{C_j^{k_j}}$ *on data item* x_j*. We say a* **zigzag path** *with respect to* $\mathbf{T}$ *exists from* $T_{C_i^{k_i}}$ *to* $T_{C_j^{k_j}}$ *if there exists a set of transactions* $\mathbf{T}' = \{T_{i_1}, T_{i_2}, \cdots, T_{i_v}\} \subseteq \mathbf{T}$ *such that*

a) $T_{i_1} \in \mathbf{T}'$ *is a transaction such that* $T_{C_i^{k_i}} \longrightarrow T_{i_1}$ *in the global serialization graph;*

b) for any $T_{i_k} \in \mathbf{T}'(1 \leq k < v)$, $T_{i_{k+1}} \in \mathbf{T}'(1 < (k+1) \leq v)$ *is a transaction such that*

1: $T_{i_k} \longleftarrow T_{i_{k+1}}$ *(we call such an edge as reverse edge);*

or

2: $T_{i_k} \longrightarrow T_{i_{k+1}}$ *or (*$T_{i_k} \longrightarrow T_{C_w^{kw}}$ *and* $T_{C_w^{kw}} \longrightarrow^+ T_{i_{k+1}}$ *for some* w*);*

c) $T_{i_v} \in T'$ *is a transaction such that* $T_{i_v} \longrightarrow T_{C_j^{k_j}}$*;*

For example, in the global serialization graph shown in Figure 2.4,

1: A zigzag path exists from $T_{C_5^1}$ to $T_{C_4^2}$, the path being $T_{C_5^1} \longrightarrow T_1 \longrightarrow T_4 \longrightarrow T_{C_4^2}$.

2: A zigzag path exists from $T_{C_5^1}$ to $T_{C_2^2}$, the path being $T_{C_5^1} \longrightarrow T_1 \longrightarrow T_4 \longleftarrow T_3 \longrightarrow T_{C_2^2}$.

3: No zigzag path exists between $T_{C_2^1}$ and $T_{C_4^1}$ or between $T_{C_4^2}$ and $T_{C_2^2}$.

Note that any directed path in the global serialization graph is also a zigzag path but not conversely.

المنارة للاستشارات
www.manaraa.com

A checkpoint $C_i^{k_i}$ (or, the corresponding checkpointing transaction $T_{C_i^{k_i}}$) is involved in a zigzag cycle (Z-cycle for short) iff there is a zigzag path from $T_{C_i^{k_i}}$ to itself. Example checkpoints that are involved in Z-cycle in Figure 2.4 include checkpoints $T_{C_5^1}$, the Z-cycle being $T_{C_5^1} \longrightarrow T_1 \longrightarrow T_4 \longleftarrow T_3 \longleftarrow T_9 \longrightarrow T_{C_5^1}$; and $T_{C_1^1}$, the Z-cycle being $T_{C_1^1} \longrightarrow T_4 \longleftarrow T_3 \longrightarrow T_{C_1^1}$. $T_{C_4^2}$ is also on a Z-cycle. Next, we establish the necessary and sufficient condition.

Theorem 2.2.2. *A set* $\mathbf{S}'$ *of checkpoints, each checkpoint of which is from a different data item, can belong to the same tr-consistent global checkpoint with respect to a serializable schedule of a set of transactions iff no checkpoint in* $\mathbf{S}'$ *has a zigzag path to any checkpoint (including itself) in* $\mathbf{S}'$ *in the global serialization graph corresponding to that schedule.*

Proof:

(If-Part:) Suppose no checkpoint in $\mathbf{S}'$ has a zigzag path to any checkpoint (including itself) in $\mathbf{S}'$. We construct a tr-consistent global checkpoint $\mathbf{S}$ that contains the checkpoints in $\mathbf{S}'$ and one checkpoint for each data item not represented in $\mathbf{S}'$ as follows:

- For each data item that has no checkpoint in $\mathbf{S}'$ and that has a checkpoint with a zigzag path to a member of $\mathbf{S}'$, we include in $\mathbf{S}$ its first checkpoint that has no zigzag path to any checkpoint in $\mathbf{S}'$. Such a checkpoint is guaranteed to exist because the virtual checkpoint of a data item, representing the state of the data item after all the transactions in $\mathbf{T}$ have terminated, does not have an outgoing zigzag path.

- For each data item that has no checkpoint in $\mathbf{S}'$ and that has no checkpoint with zigzag path to a member of $\mathbf{S}'$, we include its initial checkpoint. (It is also the first checkpoint that has no zigzag path to any member of $\mathbf{S}'$ and there cannot be a zigzag path from any checkpoint in $\mathbf{S}'$ to this initial checkpoint).

المنارة للاستشارات
www.manaraa.com

We claim that **S** is a tr-consistent global checkpoint. From Theorem 2.2.1, it is sufficient to prove that there is no path between any two checkpoints of **S** in the global serialization graph. Suppose there is a path from a checkpoint $A \in \mathbf{S}$ to a checkpoint $B \in \mathbf{S}$. Assume that the checkpoint A was taken on data item x_i and checkpoint B was taken on data item on x_j.

Case 1: $A, B \in \mathbf{S}'$. This condition implies that a zigzag path from A to B exists, contradicting the assumption that no zigzag path exists between any two checkpoints in $\mathbf{S}'$.

Case 2: $A \in \mathbf{S} - \mathbf{S}'$ and $B \in \mathbf{S}'$. This contradicts the way $\mathbf{S} - \mathbf{S}'$ is constructed (checkpoints in $\mathbf{S} - \mathbf{S}'$ are chosen in such a way that no zigzag path exists to any member of $\mathbf{S}'$ from those checkpoints).

Case 3: $A \in \mathbf{S}'$ and $B \in \mathbf{S} - \mathbf{S}'$. B cannot be an initial checkpoint, since no checkpoint can have a path to an initial checkpoint. Then by the choice of B, B must be the first checkpoint on x_j that has no zigzag path to any member of $\mathbf{S}'$. The checkpoint preceding B on x_j, say D, must have a zigzag path to some member of $\mathbf{S}'$, say E. Since D precedes B on x_j, we have, in the local serialization graph of x_j, $D \longrightarrow^+ B$, which also exists in the global serialization graph.

Let T_u be a transaction (that accessed x_j which results in the creation of the edge $T_u \longrightarrow B$ in the serialization graph) that lies on the zigzag path from A to B. Note that such transaction exists because B and D are checkpoints of data item x_j and we assume that a checkpoint is taken only after the state of the data item has been changed by one or more transactions.

Claim: There exists a zigzag path from A to E in the global serialization graph.

Proof of the claim: Since $D \longrightarrow^+ B$, $T_u \longrightarrow B$, and B is created by a checkpointing transaction, we get $D \longrightarrow^+ T_u$ in the local serialization graph of x_j from Observation 3. Any path in the local serialization graph is also a path in the global serialization graph. Therefore the path $D \longrightarrow^+ T_u$ can be found in the global serial-

المنارة للاستشارات
www.manaraa.com

ization graph. Then in the global serialization graph, the zigzag path from A to T_u, the reverse path $D \longleftarrow^+ T_u$, and the zigzag path from D to E form a zigzag path from A to E, which is a contradiction to the assumption that no zigzag path exists between any two checkpoints in $\mathbf{S}'$. Hence the claim is true.

Case 4: $A \in \mathbf{S} - \mathbf{S}'$ and $B \in \mathbf{S} - \mathbf{S}'$. As in case 3, B must be the first checkpoint of x_j that has no zigzag path to any member of $\mathbf{S}'$ and A must be the first checkpoint of x_i that has no zigzag path to any member of $\mathbf{S}'$. Then the checkpoint that precedes B on data item x_j, say D, must have a zigzag path to some member of $\mathbf{S}'$, say E. Then, as in case 3, there exists a zigzag path from A to E. This contradicts the choice of A that A is the first checkpoint on data item x_i with no zigzag path to any member of $\mathbf{S}'$.

Therefore $\mathbf{S}$, containing $\mathbf{S}'$, is a tr-consistent global checkpoint.

(Only-if Part:) Conversely, suppose there exists a zigzag path between two checkpoints in $\mathbf{S}'$ (not necessarily distinct), then we show that they cannot belong to the same tr-consistent global checkpoint. Assume that a zigzag path exists from A to B (A could be B) and along such a path, the length of consecutive reverse edges is at most w. We use induction on w to show that A and B cannot belong to the same consistent global checkpoint.

Base case ($w = 0$): If the length of consecutive reverse edges is at most zero, the zigzag path from A to B is in fact a path from A to B. Then, from Theorem 2.2.1, A and B cannot belong to the same consistent global checkpoint.

Base case ($w = 1$): Suppose the length of consecutive reverse edges along the zigzag path from A to B is at most one. Let the consecutive reverse edges with length equal to one from A to B be $T_{1,1} \longleftarrow T_{2,1}, \cdots, T_{1,u} \longleftarrow T_{2,u}$, as shown in Figure 2.5(a). Suppose those reverse edges are components of local serialization graph corresponding to data items $x_{1,1}, \cdots, x_{1,u}$ respectively.

المنارة للاستشارات
www.manaraa.com

Claim: Not all $x_{1,1}, \cdots, x_{1,u}$ can be equal to x_i, where A takes place.

Proof of claim: Suppose $x_{1,1}, \cdots, x_{1,u}$ are all equal to x_i. Then A, $T_{1,1}$, $T_{2,1}$, $\cdots$, $T_{1,u}$, $T_{2,u}$ are transactions accessing x_i (recall that we use A for the checkpointing transaction that takes the checkpoint A as well as the checkpoint itself). From Observation 1, the following two cases arise:

1) $A \longrightarrow^+ T_{2,u}$. In this case, a path $A \longrightarrow^+ B$ via $T_{2,u}$ exists and hence A and B cannot be part of a tr-consistent global checkpoint, by Theorem 2.2.1.

2) $T_{2,u} \longrightarrow^+ A$. In this case, since $T_{2,u} \longrightarrow T_{1,u}$, we must have $T_{1,u} \longrightarrow^+ A$ from Observation 3. In this case, when we consider the reverse edge $T_{1,u-1} \longleftarrow T_{2,u-1}$, the following two sub-cases arise:

   2.1) $A \longrightarrow^+ T_{2,u-1}$. In this case, a cycle $T_{1,u} \longrightarrow^+ A \longrightarrow^+ T_{2,u-1} \longrightarrow^+ T_{1,u}$ from $T_{1,u}$ to itself exists. However, a cycle cannot exist if the schedule of $\mathbf{T} \cup \mathbf{T_C} \in CSR$.

   2.2) $T_{2,u-1} \longrightarrow^+ A$. in this case, since $T_{2,u-1} \longrightarrow T_{1,u-1}$, we must have $T_{1,u-1} \longrightarrow^+ A$ from Observation 3. If this is the case, we need to consider the previous reverse edge $T_{1,u-2} \longleftarrow T_{2,u-2}$ in the zigzag path and make a similar argument with that edge. Proceeding like this, we will end up with a path $T_{1,1} \longrightarrow^+ A$; since $A \longrightarrow^+ T_{1,1}$, we have $A \longrightarrow^+ A$, i.e., A is on a cycle which is a contradiction to the assumption that the schedule of $\mathbf{T} \cup \mathbf{T_C} \in CSR$ is serializable.

So, our assumption that $x_{1,1}, \cdots, x_{1,u}$ are all equal to x_i is wrong and hence the proof of the claim. This situation is illustrated in Figure 2.5(b). In this figure, dotted lines indicate the possible paths and the dotted lines with X mark indicate the impossible paths.

Using arguments similar to the one above, we can show that $x_{1,1}, \cdots, x_{1,u}$ cannot all be x_j. Figure 2.5(c) illustrates how we can get a contradiction by showing the

المنارة للاستشارات
www.manaraa.com

existence of a cycle. So far, we have proved that there must exist a data item associated with a reverse edge that is different from both x_i and x_j. Let us assume such a data item is $x_{1,p}$ with associated reverse edge as $T_{1,p} \longleftarrow T_{2,p}$. Next we prove our claim that A and B cannot belong to a tr-consistent global checkpoint. We will use Figure 2.5(d) for understanding the basic idea behind the proof.

On data item $x_{1,1}$ that both $T_{1,1}$ and $T_{2,1}$ have accessed, no checkpoint taken after $T_{1,1}$, say D_1, (refer to Figure 2.5(d)) can be combined with A to form a consistent global checkpoint due to the path $A \longrightarrow^+ D_1$ (by Theorem 2.2.1). Therefore, on $x_{1,1}$ we can only use some checkpoint C_1 taken before $T_{2,1}$ accessed $x_{1,1}$ to construct a tr-consistent global checkpoint containing A. Using a similar argument, on $x_{1,2}$, which both $T_{1,2}$ and $T_{2,2}$ have accessed, no checkpoint taken after $T_{1,2}$ accessed, say D_2, can be combined with C_1 to form a consistent global checkpoint due to the path from $C_1 \longrightarrow^+ D_2$ (refer to Figure 2.5(d)). So we have to use some checkpoint C_2 on $x_{1,2}$, which was taken before $T_{2,2}$ accessed $x_{1,2}$. Similarly, on $x_{1,p}$, which both $T_{1,p}$ and $T_{2,p}$ have accessed, we have to use some checkpoint C_p, which was taken before $T_{2,p}$ to construct a tr-consistent global checkpoint containing A.

On the other hand, on data item $x_{1,u}$ that both $T_{1,u}$ and $T_{2,u}$ have accessed, no checkpoint taken before $T_{2,u}$, say C_u, can be combined with B to construct a tr-consistent global checkpoint due to the path $C_u \longrightarrow^+ B$. Therefore, on $x_{1,u}$, we can only use some checkpoint D_u taken after $T_{1,u}$ accessed $x_{1,u}$ to construct a tr-consistent global checkpoint containing B. Similarly, on $x_{1,u-1}$, which both $T_{1,u-1}$ and $T_{2,u-1}$ have accessed, no checkpoint taken before $T_{2,u-1}$, say C_{u-1} can be combined with D_u to construct a tr-consistent global checkpoint due to the path $C_{u-1} \longrightarrow^+ D_u$. So we have to use some checkpoint D_{u-1} on $x_{1,u-1}$ that was taken after $T_{1,u-1}$ accessed $x_{1,u-1}$. Proceeding like this, on $x_{1,p}$, which both $T_{1,p}$ and $T_{2,p}$ have accessed, we have to use some checkpoint D_p, which was taken after $T_{1,p}$ accessed $x_{1,p}$, to construct a tr-consistent global checkpoint containing B.

www.manaraa.com

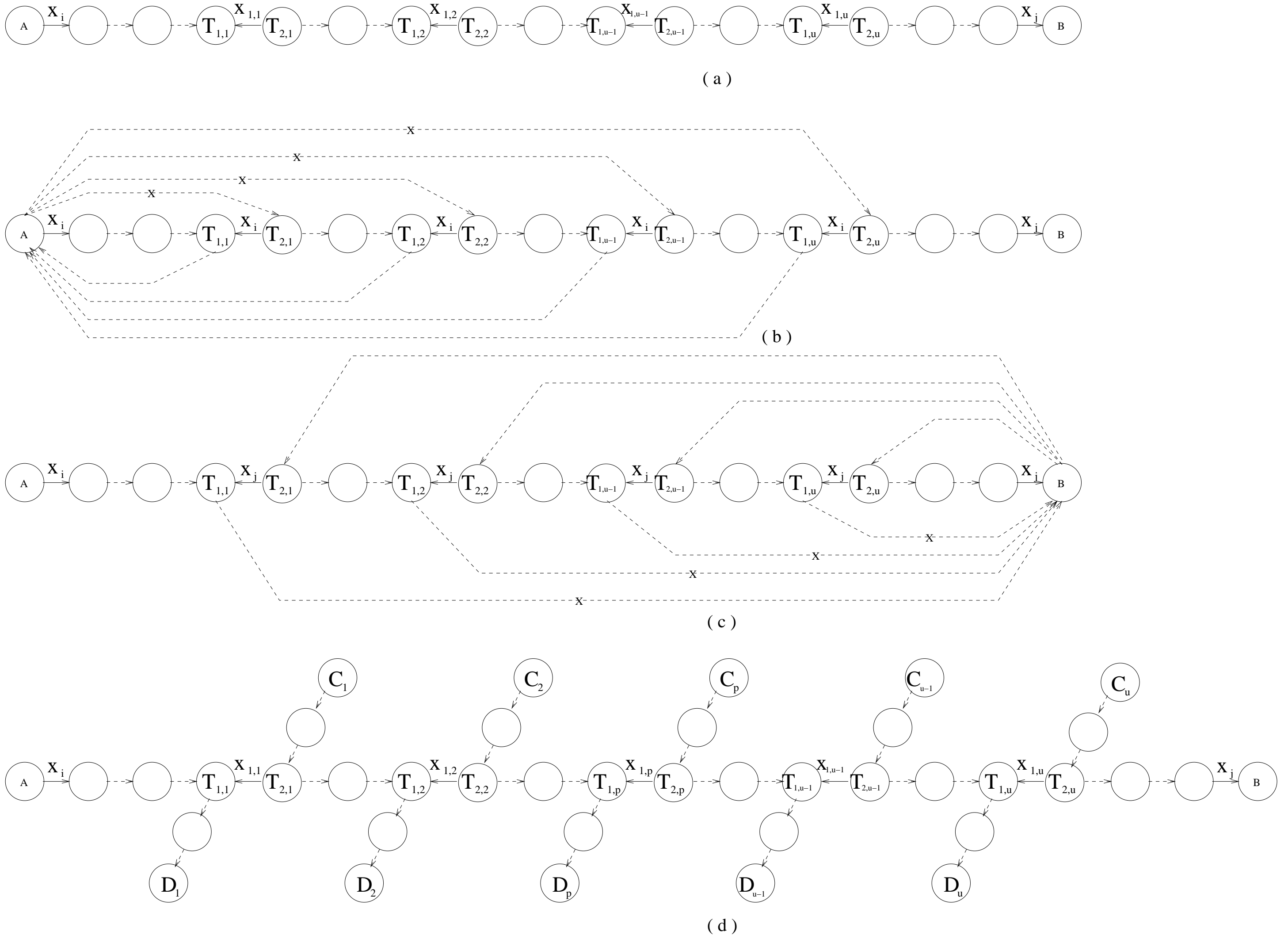


Figure 2.5: Illustration of proof for case 1

Thus, there exists a data item $x_{1,p}$ which is neither x_i nor x_j such that we can only use a checkpoint taken before $T_{1,p}$ and $T_{2,p}$ have accessed $x_{1,p}$ to construct a tr-consistent global checkpoint containing A; on the other hand, we can only use a checkpoint taken after $T_{1,p}$ and $T_{2,p}$ have accessed $x_{1,p}$ to construct a tr-consistent global checkpoint containing B. So, for data item $x_{1,p}$, there is no checkpoint that can be combined with both A and B to construct a tr-consistent global checkpoint. This proves the Theorem in the base case $w = 1$.

Next, assume that if there is a zigzag path from A to B which contains consecutive reverse edges of length at most k, then A and B together cannot belong to a tr-

www.manaraa.com

consistent global checkpoint. We prove that if there exists a zigzag path from A to B which contains consecutive reverse edges of length at most $k+1$, then A and B cannot belong to the same tr-consistent global checkpoint.

Suppose the sequence of consecutive reverse edges along the zigzag path from A to B are $T_{1,1} \longleftarrow \cdots \longleftarrow T_{u_1,1}$ ($u_1 \le k+2$), $T_{1,2} \longleftarrow \cdots \longleftarrow T_{u_2,2}$ ($u_2 \le k+2$), $\cdots$, and $T_{1,v} \longleftarrow \cdots \longleftarrow T_{u_v,v}$ ($u_v \le k+2$). Thus, on the zigzag path from A to B that we consider, we have consecutive reverse edges of lengths $u_1 - 1, \cdots, u_v - 1$, ($u_i \le k+2\ \forall i$). Each of these reverse edges should come from the local serialization graph of a data item. Suppose the reverse edges are edges of local serialization graphs of data items $x_{1,1}, \cdots, x_{u_1-1,1}, \ldots, \ldots, x_{1,v}, \cdots, x_{u_v-1,v}$ respectively. Figure 2.6(a) shows the zigzag path along with the data items from which each of the reverse edges along the path comes. First, we show that at least one of the data items $x_{1,1}, \cdots, x_{u_1-1,1} \ldots, \ldots x_{1,v}, \cdots, x_{u_v-1,v}$ is not equal to x_i (recall that A is a checkpoint of the data item x_i).

Suppose $x_{1,1}, \cdots, x_{u_1-1,1} \ldots, \ldots, x_{1,v}, \cdots, x_{u_v-1,v}$ are all the same as x_i. Then A, $T_{1,1}, \cdots, T_{u_1,1} \ldots, \ldots T_{1,v}, \cdots, T_{u_v,v}$ are transactions accessing x_i. Based on Observation 1, two cases arise:

1) $A \longrightarrow^+ T_{u_v,v}$. In this case, a path $A \longrightarrow B$ via $T_{u_v,v}$ exists, and hence A and B together cannot be part of a tr-consistent global checkpoint by Theorem 2.2.1.

2) $T_{u_v,v} \longrightarrow^+ A$. In this case, because of the sequence of reverse edges $T_{1,v} \longleftarrow \cdots \longleftarrow T_{u_v,v}$ on x_i, from Observation 3, we have $T_{1,v} \longrightarrow^+ A$. Then, when we consider the sequence of reverse edges $T_{1,v-1} \longleftarrow \cdots \longleftarrow T_{u_{v-1},v-1}$, the following two sub-cases arise:

   2.1) $A \longrightarrow^+ T_{u_{v-1},v-1}$. In this case, a cycle (namely, $T_{1,v} \longrightarrow^+ A \longrightarrow^+ T_{u_{v-1},v-1} \longrightarrow^+ T_{1,v}$) from $T_{1,v}$ to itself exists, which is a contradiction to the fact that the schedule of $\mathbf{T} \cup \mathbf{T_C} \in CSR$.

المنارة للاستشارات
www.manaraa.com

2.2) $T_{u_{v-1},v-1} \longrightarrow^{+} A$. In this case, because of the sequence of reverse edges $T_{1,v-1} \longleftarrow \cdots \longleftarrow T_{u_{v-1},v-1}$ on x_i, based on Observation 3, we have $T_{1,v-1} \longrightarrow^{+} A$. In this case, we need to consider the previous sequence of reverse edges $T_{1,v-2} \longleftarrow \cdots \longleftarrow T_{u_{v-2},v-2}$ and repeat the analysis similar to case 2.1.

Continuing this process, we will end up with a cycle in the serialization graph which is a contradiction to the fact that $\mathbf{T} \cup \mathbf{T_C} \in CSR$. This means that our assumption that $x_{1,1}, \cdots, x_{u_1-1,1} \ldots, \ldots x_{1,v}, \cdots, x_{u_v-1,v}$ are all x_i is wrong. In Figure 2.6(b), dotted lines without an X mark show the possible paths and the dotted lines with an X mark show the impossible paths.

Using similar arguments, we can show that not all the data items $x_{1,1}, \cdots, x_{u_1-1,1}$ $\ldots, \ldots x_{1,v}, \cdots, x_{u_v-1,v}$ can be equal to x_j. Figure 2.6(c) illustrates this.

So far we have proved that there must exist a data item associated with at least one reverse edge in the zigzag path from A to B that is different from both x_i and x_j. Suppose such a data item is $x_{g,p}$ and is associated with the reverse edge $T_{g,p} \longleftarrow T_{g+1,p}$ which is one of the reverse edges in the sequence of reverse edges $T_{1,p} \longleftarrow \cdots T_{u_p,p}$. Next, we prove that A and B cannot be part of a tr-consistent global checkpoint. We use Figure 2.6(d) to aid in understanding the proof.

On data item $x_{1,1}$ that both $T_{1,1}$ and $T_{2,1}$ have accessed, no checkpoint D_1, taken after $T_{1,1}$ has accessed $x_{1,1}$, can be combined with A to construct a tr-consistent global checkpoint because there is a path from A to D_1. Therefore we can only use some checkpoint C_1, taken before $T_{2,1}$ on $x_{1,1}$ to construct a consistent global checkpoint containing A. On $x_{1,2}$, which both $T_{1,2}$ and $T_{2,2}$ have accessed, no checkpoint taken after $T_{1,2}$, say D_2, can be combined with C_1 to form a consistent global checkpoint because there is a zigzag path from C_1 to D_2 with consecutive reverse edges of length at most k (by induction hypothesis). So we have to use some checkpoint C_2 on $x_{1,2}$, which was taken before $T_{2,2}$ accessed $x_{1,2}$.

المنارة للاستشارات
www.manaraa.com

Proceeding like this, on data item $x_{g,p}$, which was accessed by the transactions $T_{g,p}$ and $T_{g+1,p}$, no checkpoint D_p taken after both $T_{g,p}$ and $T_{g+1,p}$ have accessed $x_{g,p}$ can be combined with C_{p-1}, to construct a tr-consistent global checkpoint by induction hypothesis (due to the existence of the zigzag path containing consecutive reverse edges of length at most k). So we have to use some checkpoint C_p which was taken before $T_{g,p}$ and $T_{g+1,p}$ have accessed $x_{g,p}$.

On the other hand, on $x_{1,v}$, which both $T_{1,v}$ and $T_{2,v}$ have accessed, no checkpoint C_v that was taken before $T_{2,v}$ accessed $x_{1,v}$ can be combined with B to construct a tr-consistent global checkpoint because C_v has a zigzag path to B with consecutive reverse edges of length at most k. Therefore, on $x_{1,v}$, we have to use some checkpoint D_v, that was taken after $T_{1,v}$ accessed $x_{1,v}$. On $x_{1,v-1}$, which both $T_{1,v-1}$ and $T_{2,v-1}$ have accessed, we cannot use any checkpoint C_{v-1} that was taken before $T_{2,v-1}$ to construct a consistent global checkpoint containing D_v due to the existence of a zigzag path with consecutive reverse edges of length at most k. So we have to use some checkpoint D_{v-1} on $x_{1,v-1}$ that was taken after $T_{1,v-1}$ accessed. Proceeding like this, on $x_{g,p}$, we have to use some checkpoint D_p that was taken after $T_{g,p}$ has accessed to construct a tr-consistent global checkpoint containing D_{p+1}.

Thus, for the data item $x_{g,p}$, which is different from both x_i and x_j, no checkpoint that was taken before $T_{g,p}$ accessed $x_{g,p}$ can be used to construct a tr-consistent global checkpoint containing A and no checkpoint taken after $T_{g+1,p}$ accessed $x_{g,p}$ can be used to construct a tr-consistent global checkpoint containing B. Since no checkpoints exists between $T_{g,p}$ and $T_{g+1,p}$ on $x_{g,p}$, it does not have any checkpoint that can be combined with both A and B to construct a tr-consistent global checkpoint.

Therefore, A and B cannot belong to a tr-consistent global checkpoint. This proves the theorem. $\diamond$

Corollary 1. *A checkpoint of a data item in a distributed database can be part of a tr-consistent global checkpoint of the database iff it does not lie on a zigzag cycle.*

المنارة للاستشارات
www.manaraa.com

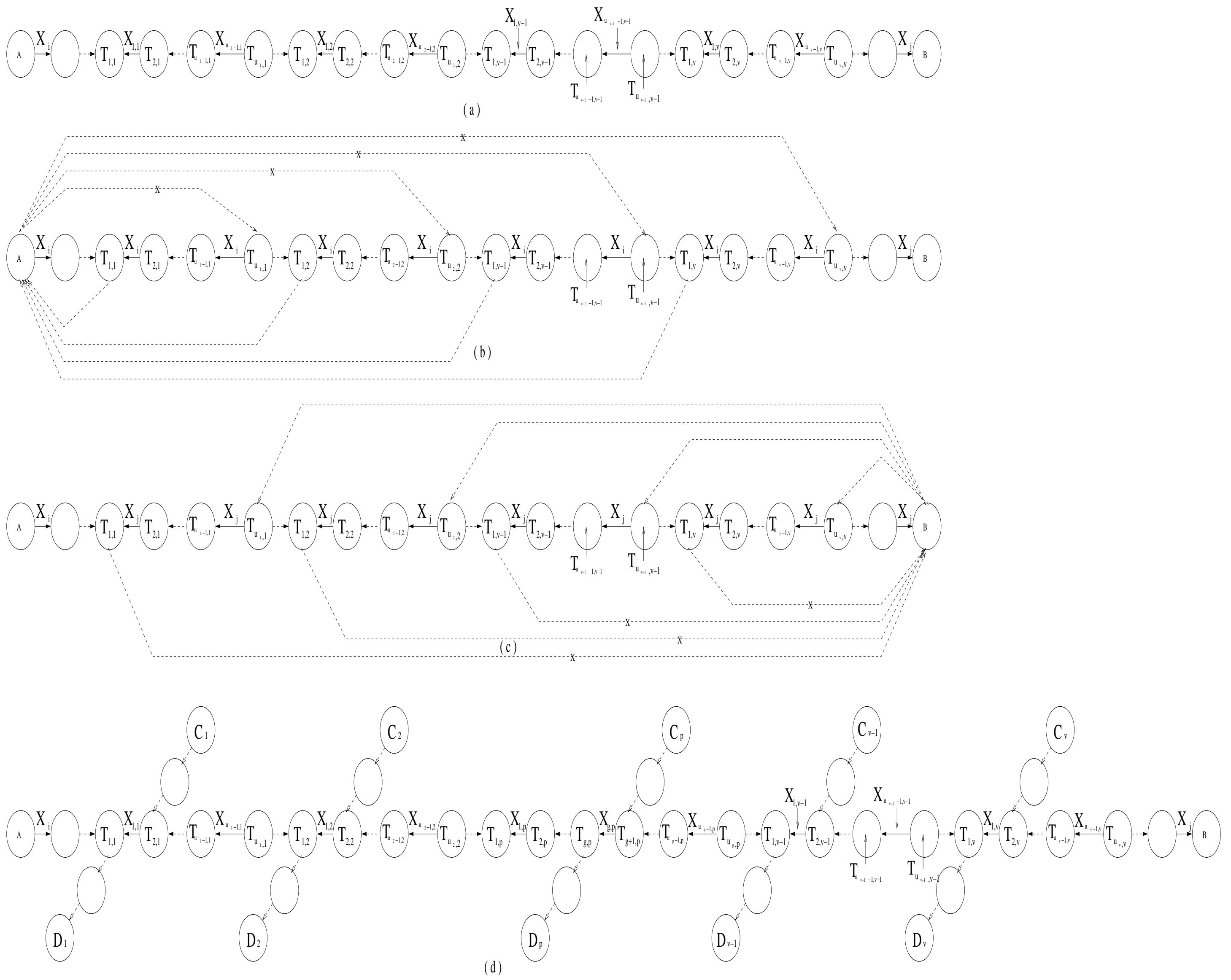


Figure 2.6: Illustration of proof under case 2

المنارة للاستشارات
www.manaraa.com

Proof: Follows from the Theorem 2.2.2 by taking $\mathbf{S}'$ as the singleton set containing the checkpoint. $\diamond$

2.2.1 Applications

Corollary 1 and Theorem 2.2.2 are useful for constructing tr-consistent global checkpoints incrementally. We can start with any checkpoint of any data item that is not on a Z-cycle, and keep adding checkpoints from other data items without violating the conditions in Theorem 2.2.2 until we have finished constructing a tr-consistent global checkpoint of the entire database. This would help in failure recovery, because when a failure occurs the database needs to be restored to a tr-consistent global checkpoint. When data items are checkpointed independently, some of the checkpoints of some of the data items may be useless because they cannot be part of any tr-consistent global checkpoint, as illustrated in Corollary 1. So, Theorem 2.2.2 can throw light on designing non-intrusive checkpointing protocols that allow each of the data items to be checkpointed independently while at the same time ensuring all checkpoints to be useful.

A federated database system (FDBS) is a collection of cooperating database systems [8, 22, 17, 18, 20]. Kleewein [21] discusses practical issues with commercial implementation of federated databases. The individual database systems in a FDBS could be heterogeneous and distributed across several geographically separated sites. In such a system, the individual databases are somewhat autonomous and hence almost all transactions updating a database will be local transactions. Thus, the individual databases can be checkpointed independently in a non-intrusive manner. However, when a failure occurs, all the component databases should be restored to a transaction-consistent global checkpoint. So, constructing a tr-consistent global checkpoint would be useful in such systems. Federated database systems are likely to play an important role in the future, especially in integrating medical databases. Even

المنارة للاستشارات
www.manaraa.com

though the concept of federated databases have been proposed in the early 90's, it has not been widely implemented. FDBSs are suitable for integrating complex data. For example, as Muilu et al. [19] point out, large-scale biobank-based post-genome era research projects like GenomEUtwin (an international collaboration between eight Twin Registries) require extensive amounts of genotype and phenotype data combined from different data sources located in different countries. Building a solid infrastructure for accessing such data requires using the model of federated databases. Muilu et al. [19] also describe how they constructed a federated database infrastructure for genotype and phenotype information collected in seven European countries and Australia and connected this database setting via a network called TwinNET.

2.3 Conclusion

Checkpointing has been traditionally used for handling failures in distributed database systems. An efficient checkpointing protocol should be non-intrusive in the sense that it should not block the normal transactions while checkpoints are taken. A simple approach would be to run a read only transaction which would read the entire database and store it in stable storage. The underlying concurrency control algorithm would ensure that the saved state is tr-consistent. This approach would be very inefficient, especially in the presence of long-living transactions. If each data item is independently checkpointed, not all the checkpoints taken may be useful for constructing a tr-consistent global checkpoint of the entire database. We have presented the necessary and sufficient condition for a set of checkpoints of a set of data items in the database to be part of a tr-consistent global checkpoint of the database. This theory helps in determining which checkpoints are useful for constructing tr-consistent global checkpoints and which are not. It also helps in constructing tr-consistent global checkpoints of the database incrementally starting from an useful checkpoint of any data item. Moreover, the necessary and sufficient conditions established can throw

المنارة للاستشارات
www.manaraa.com

light on designing non-intrusive checkpointing methods which allow data items to be checkpointed independently while at the same time ensure each checkpoint taken is part of a tr-consistent global checkpoint.

للاستشارات
www.manaraa.com

Chapter 3

A Non-intrusive Checkpointing Protocol for Distributed Database Systems

Non-intrusive checkpointing protocols for distributed database systems are preferable to intrusive checkpointing protocols since they do not interfere with normal system activities. For example, executing transactions are not aborted, delayed, or quiesced in order to save a tr-consistent global checkpoint.

Communication-induced checkpointing approaches designed for distributed computations running on distributed systems are non-intrusive checkpointing protocols. Baldoni et al. [5] presented two non-intrusive checkpointing protocols for database systems which are adaptations of communication-induced checkpointing protocols for distributed computations.

Even though the notion of data items in database systems corresponds to processes in distributed computations, communication-induced checkpointing protocols designed for distributed computations cannot be directly applied to distributed database systems. Checkpointing and recovery in a distributed database system differs from checkpointing and recovery for distributed computations in the following aspects: (i) When a failure occurs, the processes involved in a distributed computation need to be restarted from a consistent global checkpoint; however, in a distributed database system, when a failure occurs, the states of the data items of the database need to be restored to a state which represents a tr-consistent global checkpoint. (ii) Dependencies among the states of the processes involved in a distributed computation

المنارة للاستشارات
www.manaraa.com

are caused by messages exchanged between processes; in distributed database systems dependencies among the states of the data items exist because of the concurrent interleaved execution of transactions. (iii) For distributed computations, states of processes are saved in stable storage whereas in database systems, states of data items are saved in stable storage.

When restoring a database from checkpoints, it is important that the checkpoints of the data items to which the database is restored form a transaction-consistent global checkpoint. In coordinated checkpointing protocols, first the arriving transactions are blocked, then the state of the data items of the database are saved after the currently executing transactions finish and then the blocked transactions are allowed to execute. On the other hand, communication-induced checkpointing protocols such as the one proposed in [5], allow the data items to be checkpointed independently and some forced checkpoints are also taken if the dependency relation among checkpoints of data items satisfies some condition. Forced checkpoints are taken in such a way that every checkpoint of every data item is part of a transaction-consistent global checkpoint.

One problem with the protocols in [5] is that they induce a large number of forced checkpoints. We designed a non-intrusive checkpointing protocol which reduces the checkpointing overhead. Our protocol is motivated by the concept of *logical checkpoints* introduced by Vaidhya [54], and the fact that most distributed database systems are deterministic, where transaction logging mechanisms are built-in components for recovery and safety purposes.

The DM on a data item may take two types of checkpoints, namely, physical checkpoints and logical checkpoints [54]. We say a physical checkpoint of a data item has been taken at time t_1 (local time) if the state of the data item has been stored in the stable storage at time t_1. We say a logical checkpoint of a data item has been taken at time t_1 if adequate information has been saved in stable storage to allow

المنارة للاستشارات
www.manaraa.com

the state of the data item at time t_1 to be reconstructed. A physical checkpoint is trivially a logical checkpoint, however, the converse is not true.

One approach to take logical checkpoint of a data item at time t_1 is to take a physical checkpoint at some time $t_0 < t_1$ and log all operations performed on that data item between time t_0 and t_1 into stable storage. This approach can be summarized as [54]:

physical checkpoint + operation log = logical checkpoint

This approach for taking a logical checkpoint may only be used in deterministic systems, because it implements a logical checkpoint using a physical checkpoint and the operation log, which require the system to be deterministic.

3.1 Proposed Protocol

The design of our protocol has been motivated by the protocols of Baldoni et al. [5]. Next, we briefly review the basic idea behind the protocols of Baldoni et al. [5].

3.1.1 Basic Idea Behind the Protocols of Baldoni et al.

Baldoni et al. [5] assume that each data manager DM_x managing data item x maintains a variable ts_x, which stores the timestamp of the last checkpoint of x. Another variable i_x is used to maintain the index of the last checkpoint of x. There is a one-to-one correspondence between the values taken by i_x and ts_x; ts_x may be incremented unevenly while i_x is incremented by one each time. Therefore each checkpoint of x taken at time t_x, denoted by $C_x^{i_x}$ is also denoted by $C_x^{ts_x}$. Moreover, data managers can take basic checkpoints of data items independently and periodically. To facilitate this function, a timer is associated with each data manager and when the timer expires, a checkpoint is taken and the timer is reset. Checkpoints taken this way are called basic checkpoints. In addition, whenever a condition that needs to be prevented

المنارة للاستشارات
www.manaraa.com

is detected by means of comparing timestamps, data managers are directed to take additional checkpoints (referred to as forced checkpoints) in order to ensure that each local checkpoint belongs to a transaction-consistent global checkpoint. The decision to take forced checkpoints is made based on the control information (timestamps) piggybacked with commit messages of transactions. Next, we discuss briefly how the control information is maintained and communicated.

Let R_{T_i}/W_{T_i} be the set of read/write operations issued by a transaction T_i, which is under the control of transaction manager TM_i. Each time an operation of T_i is issued by TM_i to a data manager DM_x on data item x, besides the execution of the operation, DM_x returns the pair (identity of the data item x, value of its current timestamp ts_x) to TM_i. TM_i stores in $MAX_TS_{T_i}$ the maximum value among the timestamps collected from all the data items that are read and/or written by T_i. When the transaction T_i is about to commit, the transaction manager TM_i sends a COMMIT message to each data manager DM_y involved in R_{T_i}/W_{T_i}. The COMMIT messages are piggybacked with $MAX_TS_{T_i}$. Whenever a COMMIT message is received by DM_y, if $MAX_TS_{T_i}$ piggybacked in COMMIT is larger than the value of the local variable ts_y, a forced checkpoint is taken on data item y and ts_y is updated to the value of $MAX_TS_{T_i}$. In addition, whenever a basic checkpoint is taken (when timer expires), the local ts_x is incremented by 1. Based on how often the forced checkpoints are taken, two protocols are introduced in [5]. In the first protocol (we call it as Protocol 1), a forced checkpoint is taken whenever the condition $MAX_TS_{T_i} > ts_x$ holds.

When a site in a distributed database system fails, the states of all the data items at that site have to be restored from checkpoints in the stable storage during the recovery process. If a data item x is restored to a checkpoint with timestamp ts_x, the state of the other data items in the database are restored to their checkpoints with the smallest timestamp that is greater than or equal to ts_x. Such a state is

www.manaraa.com

guaranteed to be transaction-consistent as proved in [5]. If a data item only has local checkpoints with timestamps less than ts_x, it remains in its current state.

Let us call the above mentioned protocol of Baldoni et al. [5] as Protocol 1. To reduce the checkpointing overhead of Protocol 1, they [5] also proposed another protocol (let us call it as Protocol 2) to reduce the number of forced checkpoints based on the idea of lazy checkpointing introduced by Wang [44] for distributed systems. This protocol introduces a system parameter $Z \geq 1$ known to all data managers. This protocol ensures that $\forall x$ (x is a data item) if there exists a checkpoint timestamped $a \times Z$ (where $a \geq 0$ is an integer), then a tr-consistent global checkpoint containing that checkpoint exists. In addition to the control variables used in Protocol 1, each data manager DM_x in Protocol 2 has an additional **variable** V_x, which is incremented by Z (where Z is a predetermined positive integer) each time a forced checkpoint on x, timestamped $a \times Z$, is taken. Protocol 2 does not take forced checkpoints as often as Protocol 1, because it skips taking some forced checkpoints that would have been taken under Protocol 1. It takes forced checkpoint only when $MAX_TS_{T_i} > V_x$ becomes true.

3.1.2 Proposed Checkpointing Protocol

Our protocol reduces the checkpointing overhead further using a different approach. The basic idea behind our protocol is that whenever a forced checkpoint needs to be taken, our protocol tries to avoid it by only placing a marker into the log. Our protocol takes a forced checkpoint only when it can not be avoided. We use the concept of logical checkpoints to reconstruct a transaction-consistent global checkpoint by combining physical checkpoints and the information maintained in the log appropriately. During recovery, physical checkpoints together with logs are used to restore the items of the database to a transaction-consistent global checkpoint.

Similar to the approach in [5], each data manager DM_x has a variable ts_x. ts_x

المنارة للاستشارات
www.manaraa.com

stores the timestamp of the last checkpoint (a physical checkpoint or just a marker in the log) of data item x. i_x denotes the index value of the last physical checkpoint of data item x. In addition, we have a variable $ts_{physical}$ that stores the timestamp of the last physical checkpoint. In the log, a physical checkpoint is represented by $Checkpoint(ts_x, C_x^{ts})$, where C_x^{ts} is a link to the location of this physical checkpoint in the stable storage. This log entry means a physical checkpoint C_x^{ts} with timestamp ts_x has been taken. We denote by $Checkpoint.ts_x$ the first element in this tuple. A marker, on the other hand, uses a pair Marker($ts_{physical}$, ts_x). $ts_{physical}$ is the timestamp of the previous physical checkpoint that the marker depends on for recovery and ts_x is the timestamp of the marker. We use $Marker.ts_{physical}$ and $Marker.ts_x$ to refer to the first and second element in this tuple. We also use a variable called m_{num} to control the size of the consecutive markers within a reasonable range, called m_{max} and initialized as Z. If the number of consecutive markers exceeds $m_{max} = Z$, we will have to take a forced checkpoint rather than continue placing a marker into the log in order to reduce the recovery time when a failure occurs. The proposed checkpointing protocol is given in Table 3.1. The corresponding recovery protocol under this checkpointing scheme is given in Table 3.2.

3.2 Performance Analysis

3.2.1 Performance Analysis

Hereafter, we call Protocol 1, Protocol 2 and our protocol Protocol A, B, and C respectively. Both protocols B and C are derived from Protocol A and they differ in the way in which they suppress forced checkpoints. Firstly, let us analyze the number of basic checkpoints taken under protocols A, B and C. Protocols A and C reset timer at the same point for the same set of transactions. i.e., if there is a basic checkpoint taken by Protocol A, there must be a corresponding basic checkpoint taken under Protocol C, and vice versa. Therefore, both protocols take the same number of

المنارة للاستشارات
www.manaraa.com

basic checkpoints. Protocol B is likely to take more basic checkpoints than Protocol A since it does not reset timer when suppressing a forced checkpoint, which means it does not reset timer as often as protocol A and C, and hence takes more basic checkpoints.

Both protocol B and C take less forced checkpoints than A since they suppress some forced checkpoints. Moreover, Protocol C takes less forced checkpoints than Protocol B. The parameter Z in Protocol C functions similar to the parameter Z in Protocol B. Suppose they have the same value. Let us assume Protocol A is used in a database system for a specific transaction pattern and we also assume that on data item x, there are a total of m forced checkpoints taken between two consecutive basic checkpoints (i.e., basic checkpoint interval). If we use Protocol A in the same system with the same transaction pattern, the number of forced checkpoints taken will be $\lfloor \frac{m}{Z} \rfloor$. Protocol C is not sensitive to $MAX_TS_{T_i}$ in the sense that a data item places a marker in the log instead of taking forced checkpoints until the number of consecutive markers placed in the log reaches Z. On the other hand, under Protocol B, on average, $\lfloor \frac{2 \times m}{Z} \rfloor$ forced checkpoints will be taken between any two consecutive basic checkpoints, the lower on the number of forced checkpoints in this case being $\lfloor \frac{m}{Z} \rfloor$ which occurs when the local timestamp is increased by one each time a $COMMIT(MAX_TS_{T_i})$ arrives; the upper bound is m and occurs if the value of $MAX_TS_{T_i}$ carried by $COMMIT(MAX_TS_{T_i})$ is always larger than V_x. This means Protocol B is sensitive to $MAX_TS_{T_i}$ and it takes more forced checkpoints than Protocol C. On average, our protocol (Protocol C) saves $\lfloor \frac{2 \times m}{Z} \rfloor - \lfloor \frac{m}{Z} \rfloor \leq \lceil \frac{m}{Z} \rceil$ checkpoints on each data item in each basic checkpoint interval.

Since our protocol avoids taking forced checkpoints by placing markers in the log, during recovery, our protocol may have to require data items to roll back to an earlier physical checkpoint than Protocol B in order to reconstruct the most recent consistent state. In other words, under Protocol C, compared to under Protocol B,

المنارة للاستشارات
www.manaraa.com

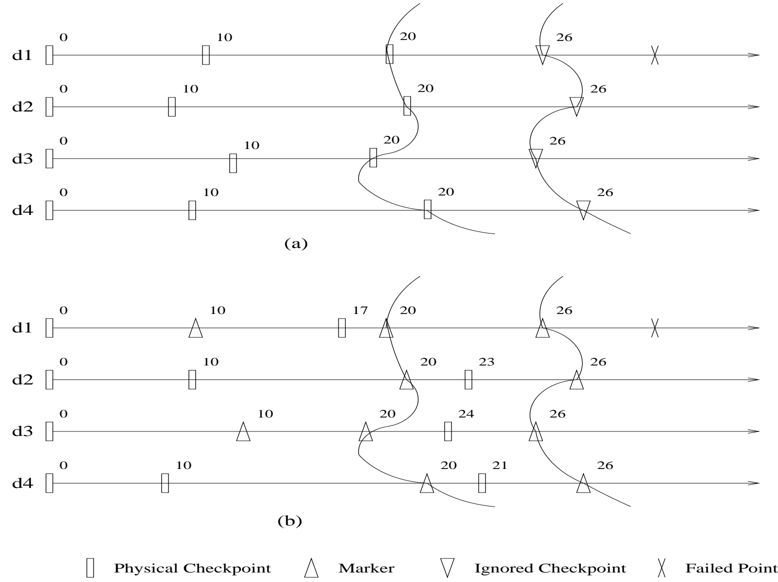


Figure 3.1: Recovery cost

more time may be needed to restore the database to a consistent state when a failure occurs. But statistical analysis and our simulations show that the expected recovery cost on a data item is the same for both Protocols B and C. We define the recovery cost on a data item as the total number of forced checkpoints that are suppressed and the physical checkpoints that are not used during recovery (i.e., checkpoints that lie between the most recent checkpoint on the data item before failure and the actual physical checkpoint to which the data item is restored to during the recovery). These checkpoints are the cost we must pay in order to decrease the number of checkpoints taken. We illustrate this using Figure 3.1.

In Figure 3.1(a), suppose data item d_1 initiates recovery at the point indicated by the cross mark after checkpoint with timestamp 26 has been taken if Protocol A is used. Under Protocol B, suppose such a checkpoint is ignored (suppressed) to reduce the number of checkpoints taken. Then, under Protocol B, d_1 has to roll back to the checkpoint with timestamp $\lfloor \frac{26}{10} \rfloor \times 10 = 20$, assuming $Z = 10$. The state represented by the checkpoint with timestamp 26 is the ideal state to which we would like d_1 to be restored even though it does not exist. The checkpoint with timestamp 20 is the

المنارة للاستشارات
www.manaraa.com

physical checkpoint to which the data item is restored during the recovery. Therefore, the recovery cost is the number of forced checkpoints that are supposed to be taken in Protocol A but suppressed by Protocol B as well as possible basic checkpoints that are never used during recovery. Since the timestamp difference is 6, we can derive that the upper bound of the recovery cost is 6 for data item d_1. Similarly, under Protocol B, the states of all other data items d_i (d_2, d_3 and d_4 in Figure 3.1(a)) have to be restored to checkpoint 20; then the log needs to be replayed until their state reaches the state represented by the logical checkpoint with timestamp 26, since the logical checkpoints with timestamp 26 forms the latest transaction-consistent global checkpoint with respect to this failure. All of them have an upper bound 6 of the recovery cost. In a distributed database system where transactions access all the data items with the same probability, the recovery cost for each data item should be similar in value under Protocol B. Therefore if Protocol B is used, we can use one random variable for a data item to predict the total recovery cost for the whole system.

Checkpoints taken for the execution of the same set of transactions under our protocol is shown in Figure 3.1(b). In Figure 3.1(b), failure occurs after data item d_1 takes checkpoint 26 has been taken if Protocol A is used. Under our protocol, checkpoint 26 is not taken and only a marker is inserted in the log. When d_1 recovers, it has to restore its state to checkpoint 17 since checkpoint 17 is the nearest physical checkpoint. Recall that the number of suppressed forced checkpoints between checkpoint 26 and checkpoint 17 can not exceed 10 ($Z = 10$). So, the recovery costs in this example are 9 for d_1, 3 for d_2, 2 for d_3 and 5 for d_4. Under our protocol, each data item has no longer similar recovery cost, but the recovery cost of each data item is $\leq Z$. Therefore for analyzing the recovery cost of our protocol, we used a random variable for predicting the recovery cost of each data item.

A protocol that results in low recovery cost as well as low checkpointing overhead is ideal. The recovery cost can also be interpreted as the penalty we pay during

المنارة للاستشارات
www.manaraa.com

recovery if we suppress some forced checkpoints. Based on this useful parameter, we compare our protocol (Protocol C) with Protocol B and see how our protocol trades longer recovery time for less checkpointing overhead in the system, compared to Protocol B.

Under Protocol B, each data item may have recovery cost ranging from 0 to $Z-1$ with equal probability. This is the only random variable, denoted as R, we use for evaluating the recovery cost. In addition, all data items should have similar recovery cost and therefore we can make reasonable assumption that they are equal in value (our simulation also shows the similarity of the recovery cost among all the data items). For example, suppose $Z = 5$ and we have two data items x_1 and x_2. Then, when a failure occurs, the expected total recovery cost due to that failure under Protocol B is

$2 \times (0 \times \frac{1}{5} + 1 \times \frac{1}{5} + 2 \times \frac{1}{5} + 3 \times \frac{1}{5} + 4 \times \frac{1}{5}) = 4.$

In general, the expected total recovery cost can be expressed as $n \times E(R) = n \times \frac{1}{2} \times (Z-1)$, where n is the number of data items and $E(R)$ is the expected value of random variable R.

On the other hand, under our protocol, the recovery cost for each data item x is a random variable R_x taking values from 0 to $Z-1$. For example, if we assume $Z = 5$ and the database has two data items (x_1 and x_2), we calculate the *joint probability* ΣR_x where $x \in \{x_1, x_2\}$. Possible values for $R_{x_1} + R_{x_2}$ lies in $\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$. If $R_{x_1} + R_{x_2} = 0$, there is only one possibility $(0 + 0 = 0)$. $R_{x_1} + R_{x_2} = 1$, there are two possibilities $(0 + 1 = 1$ or $1 + 0 = 1)$. Similarly, if $R_{x_1} + R_{x_2} = 2$, there are three possibilities; if $R_{x_1} + R_{x_2} = 3$, there are four possibilities; if $R_{x_1} + R_{x_2} == 4$, there are five possibilities; if $R_{x_1} + R_{x_2} = 5$, $R_{x_1} + R_{x_2} = 6$, $R_{x_1} + R_{x_2} = 7$, and $R_{x_1} + R_{x_2} = 8$, there are four, three, two and one possibilities respectively. Therefore the expected total recovery cost is

$0 \times \frac{1}{25} + 1 \times \frac{2}{25} + 2 \times \frac{3}{25} + 3 \times \frac{4}{25} + 4 \times \frac{5}{25} + 5 \times \frac{4}{25} + 6 \times \frac{3}{25} + 7 \times \frac{2}{25} + 8 \times \frac{1}{25} = 4.$

المنارة للاستشارات
www.manaraa.com

In general, the expected total recovery cost can be expressed as $E(\sum_{x=1}^{n} R_x)$.

This simple example shows that Protocol B of Baldoni et al. has the same recovery cost as our protocol. This is also true for any Z and for any amount of data items consisting of the distributed database system, based on the following reasoning. Since $\forall x$ where $1 \leq x \leq n$, R_x is independent of each other, then

$$E(\sum_{x=1}^{n} R_x) = \sum_{x=1}^{n} E(R_x) = n \times \frac{1}{2} \times (Z - 1).$$

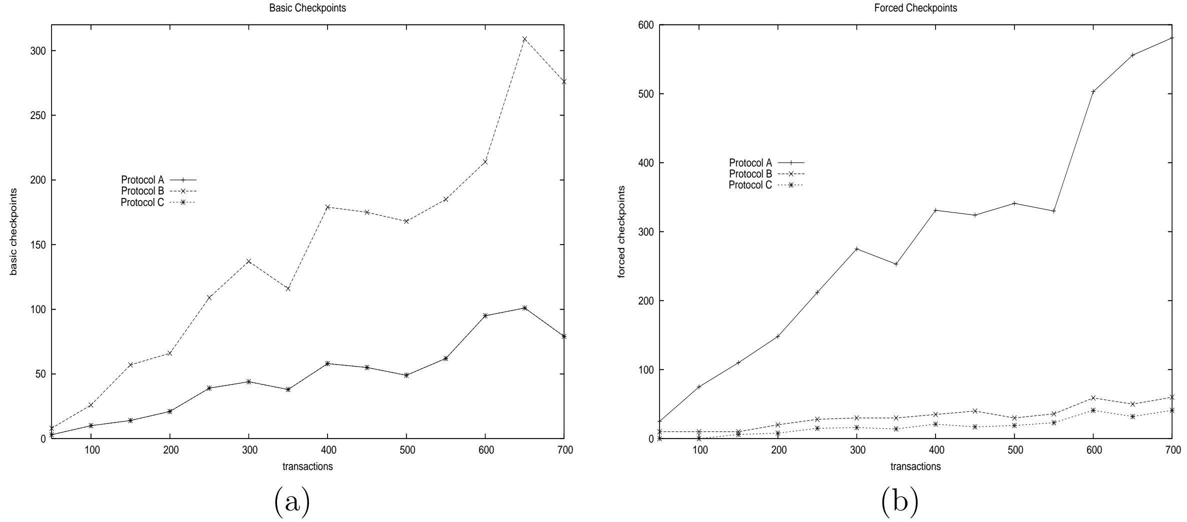
Since the expected total recovery cost are the same for a failure under both Protocol B and C, we know that on each data item, the expected recovery cost should also be the same (close to $\frac{1}{2} \times (Z - 1)$) because we have a fixed number of data items in the system.

From the brief analysis above, we realize that our protocol reduces the checkpointing overhead by reducing the numbers of forced checkpoints taken without introducing additional cost for recovery. In addition, our protocol is easy to implement since the cost of placing markers into the log is low and the log is a built-in component of most database systems. In the next subsection, we show the superiority of our protocol over protocols in [5] through our simulation results.

3.2.2 Simulation Results

We simulated a distributed database system with 10 data items and the transactions can access all the data items with equal probability. In the simulation, a random data item initiates recovery and the total recovery cost for the recovery process is calculated under Protocols B and C. Then the average recovery cost is calculated for each data item under both protocols. In the simulation, under both Protocols B and C, we counted the number of suppressed forced checkpoints, as the recovery cost. When counting the recovery cost under Protocol B, we also counted the basic checkpoints that lie between the physical checkpoints restored to and the most recent

المنارة للاستشارات
www.manaraa.com



(a) (b)

Figure 3.2: Simulation results (a) basic checkpoints (b) forced checkpoints

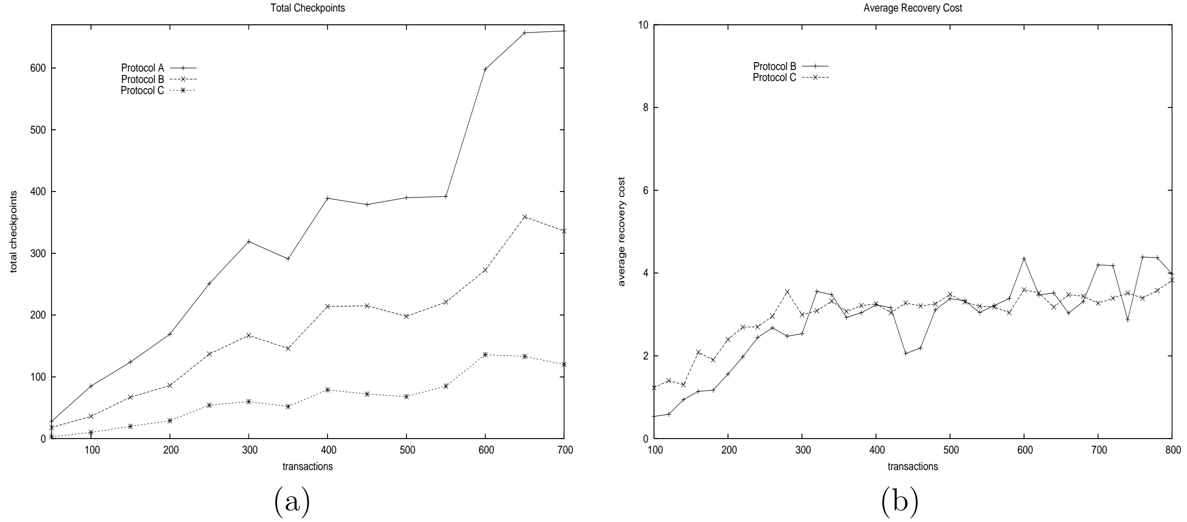
suppressed forced checkpoint. Under Protocol C, no such basic checkpoints exist and we only count suppressed forced checkpoints.

Figure 3.2(a) shows the number of the basic checkpoints taken as the number of transactions processed varies. As we expected, as the number of transactions increases, the basic checkpoints in the system also increase. In addition, Protocol A and C take more or less the same number of basic checkpoints but Protocol B takes much more.

Figure 3.2(b) shows the number of the forced checkpoints taken as the number of transactions processed varies. As predicted in our theoretical analysis, Protocol B takes more forced checkpoints than C. They both take much less forced checkpoints than Protocol A. Figure 3.3(a) shows the total number of checkpoints (basic and forced) in the system for Protocol A, B and C.

Figure 3.3(b) shows the difference in the average(expected) recovery cost for one data item under Protocol B and C for varied number of transactions. As we can see from Figure 3.3(b), the difference in the average recovery cost for a data item among Protocols B and C for the same transaction pattern is small. This similarity does not depend on the number of transactions. In order to show the similarity of

المنارة للاستشارات
www.manaraa.com



(a) (b)

Figure 3.3: Simulation results (a) total checkpoints (b) average recovery cost

the average recovery cost among Protocols B and C mathematically, we conducted the two-sample *t-test* for the data in Figure 3.3. The two-sample t-test is used to determine if two population means are equal. A common application of this method is to test if a new process or treatment is superior to a current process or treatment. The two-sample t-test for two sets of data is defined as:

H_0 (*null hypothesis*): $\mu_1 = \mu_2$

H_a (*alternative hypothesis*): $\mu_1 \neq \mu_2$

Test Statistics: $T = \frac{\overline{Y_1} - \overline{Y_2}}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$ where N_1 and N_2 are the sample sizes, $\overline{Y_1}$ and $\overline{Y_2}$ are the sample means, and s_1^2 and s_2^2 are the sample variances.

Significance Level: α

Critical Region: Reject the null hypothesis that the two means are equal if $T < -t_{(\frac{\alpha}{2},\upsilon)}$ or $T > t_{(\frac{\alpha}{2},\upsilon)}$, where $t(\frac{\alpha}{2},\upsilon)$ is the critical value of the t *distribution* with υ degrees of freedom where $\upsilon = \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{(\frac{s_1^2}{N_1})^2}{(N_1-1)} + \frac{(\frac{s_2^2}{N_2})^2}{(N_2-1)}}$

Table 3.3 shows the result from the Minitab program. The fourth line of the table shows that the sample data for Protocol B have 36 samples, the mean is 2.86, the

المنارة للاستشارات
www.manaraa.com

standard deviation is 1.05 and the standard deviation of mean is 0.18. The fifth line shows the data for Protocol C have also 36 sample with mean 3.010, standard deviation 0.655, and standard deviation of mean 0.11. The T value of this test is -0.74 shown in the ninth line. In order to reject the null hypothesis H_0 that the two means are equal, the value of T must satisfy $T < -t_{(\frac{\alpha}{2},\upsilon)}$ or $T > t_{(\frac{\alpha}{2},\upsilon)}$. Let us choose the Significance Level criteria as $\alpha = 0.05 = 5\%$. Then we have $t_{(\frac{\alpha}{2},\upsilon)} = -2.00172$ which is shown in the 13th line, where $\upsilon = 58$ (degree of freedom υ is given by DF in the ninth line). It is obviously false and therefore the null hypothesis H_0, which is $\mu_1 = \mu_2$ must stand with $1 - 5\% = 95\%$ confidence. We can finally conclude that there is no significant difference between population B and C with 95% confidence.

3.3 Conclusion

Checkpointing has been traditionally used for handling failures in distributed database systems. An efficient checkpointing protocol should be non-intrusive in the sense that it should not block the normal transactions while checkpoints are taken. In this chapter, we presented a non-intrusive checkpointing protocol for distributed database systems which has low checkpointing overhead compared to some existing checkpointing protocols while at the same time does not increase the recovery cost.

المنارة للاستشارات
www.manaraa.com

Table 3.1: Proposed Checkpointing Protocol

- Initialization:
  $ts_x = 0$;
  $i_x = 0$;
  $ts_{checkpoint} = 0$;
  $m_{num} = 0$;
  $m_{max} = Z$;

- Taking basic checkpoints, when the time expires:
  $i_x \leftarrow i_x + 1$;
  $ts_x \leftarrow ts_x + 1$;
  Take checkpoint $C_x^{ts_x}$;
  $ts_{physical} \leftarrow ts_x$;
  Write $Checkpoint(ts_x, C_x^{ts})$ into the log;
  $m_{num} \leftarrow 0$;
  Reset the local timer.

- Taking forced checkpoints or placing markers in the log, when DM_x receives $COMMIT(MAX_TS_{T_i})$ from TM_i:
  if $ts_x < MAX_TS_{T_i}$ and $m_{num} \leq m_{max}$ then

    $i_x \leftarrow i_x + 1$;

    $ts_x \leftarrow MAX_TS_{T_i}$;

    Place the marker $Marker(ts_{physical}, ts_x)$ into the log;

    $m_{num} \leftarrow m_{num} + 1$;

    Reset the local timer;

  else if $ts_x < MAX_TS_{T_i}$ and $m_{num} > m_{max}$ then

    $i_x \leftarrow i_x + 1$;

    $ts_x \leftarrow MAX_TS_{T_i}$;

    $ts_{checkpoint} \leftarrow ts_x$;

    Take forced checkpoint C_x^{ts};

    Write $Checkpoint(ts_x, C_x^{ts})$ into the log;

    $m_{num} \leftarrow 0$;

    Reset the local timer;

  endif
  process the COMMIT message.

المنارة للاستشارات
www.manaraa.com

Table 3.2: Recovery Protocol Based on the Checkpointing Protocol

- Whenever a data item x needs to roll back and initiate recovery:
  Locate the latest checkpoint information in the log;
  if it is a checkpoint in the form of $Checkpoint(ts_x, C_x^{ts})$

  - Track through the link C_x^{ts} in $Checkpoint(ts_x, C_x^{ts})$ in the stable storage and restore the state of the data item to that checkpoint;
  - Broadcast $Recovery(ts_x)$ message to all the data items in the distributed database system where $ts_x = Checkpoint.ts_x$

  else if it is a marker in the log of the form of $Marker(ts_{checkpoint}, ts_x)$

  - Continue searching backward in the log until reaching the entry $Checkpoint(ts_x, C_x^{ts})$ where $Checkpoint.ts_x = Marker.ts_{checkpoint}$;
  - Track through the link C_x^{ts} in the $Checkpoint(ts_x, C_x^{ts})$ to the stable storage and restore the state of the data item;
  - Redo all the operations in the log between $Checkpoint(ts_x, C_x^{ts})$ and $Marker(ts_{checkpoint}, ts_x)$;
  - Broadcast $Recovery(ts_x)$ message to all the data items in the distributed database system where $ts_x = Marker.ts_x$;

  endif
  Resume normal database processing.

- Whenever a data item y receives $Recovery(ts_x)$ message:
  Track backward through all the checkpoint or marker entries in the log and locate the one with the smallest ts_y such that $ts_y \geq ts_x$;
  if it is a checkpoint in the form of $Checkpoint(ts_y, C_y^{ts})$

  - Track through the link C_y^{ts} in the $Checkpoint(ts_y, C_y^{ts})$ to the stable storage and restore the state of the data item to that checkpoint;

  else if it is a marker in the log of the form $Marker(ts_{physical}, ts_y)$

  - Continue searching backward in the log until reaching a $Checkpoint(ts_y, C_y^{ts})$ entry where $Checkpoint.ts_y = Marker.ts_{physical}$;
  - Track through the link C_y^{ts} in the $Checkpoint(ts_y, C_y^{ts})$ to the stable storage and restore the state of the data item;
  - Redo all the operations in the log between $Checkpoint(ts_y, C_y^{ts})$ and $Marker(ts_{checkpoint}, ts_y)$;

  endif
  Resume normal database processing.

المنارة للاستشارات
www.manaraa.com

Table 3.3: Two Sample T-test Result

1 Two-Sample T-Test and CI: B, C

2 Two-sample T for B vs C

3 N Mean StDev SE Mean

4 B 36 2.86 1.05 0.18

5 C 36 3.010 0.655 0.11

6 Difference = mu (B) - mu (C)

7 Estimate for difference: -0.153315

8 95% CI for difference: $(-0.567235, 0.260605)$

9 T-Test of difference = 0 (vs not =): T-Value = -0.74 P-Value = 0.461 DF = 58

10 Inverse Cumulative Distribution Function

11 Student's t distribution with 58 DF

12 $P(X \leq x)$ x

13 0.025 -2.00172

المنارة للاستشارات

www.manaraa.com

Chapter 4

An Enhanced Model-based Communication-Induced Checkpointing Protocol for Distributed Systems

For distributed computations running in distributed systems, communication-induced checkpointing protocols have recently received lot of attention due to their following attractive features: (i) Each process involved in the distributed computation can take checkpoints independently without any coordination with other processes in the computation, called basic checkpoints. (ii) Each basic checkpoint taken is ensured to be part of a consistent global checkpoint of the computation; this is achieved by forcing processes to take some additional checkpoints, called forced checkpoints. It is known that designing a communication-induced checkpointing protocol with optimal number of forced checkpoints is not possible. So, one of the main goals of a communication-induced checkpointing protocol is to reduce the number of forced checkpoints while making all checkpoints useful.

As we noted in Chapter 1, communication-Induced checkpointing protocols have been classified into two subclasses, namely, index-based and model-based protocols. In the model-based protocols, a communication pattern that could result in Z-cycles is identified. The protocols then monitor and detect the formation of such communication patterns and take forced checkpoints to prevent such undesirable checkpoint and communication patterns from occurring, thereby make all checkpoints useful.

In this chapter, we present a model-based communication-induced checkpointing protocol. Next we present the motivation behind designing this checkpointing proto-

www.manaraa.com

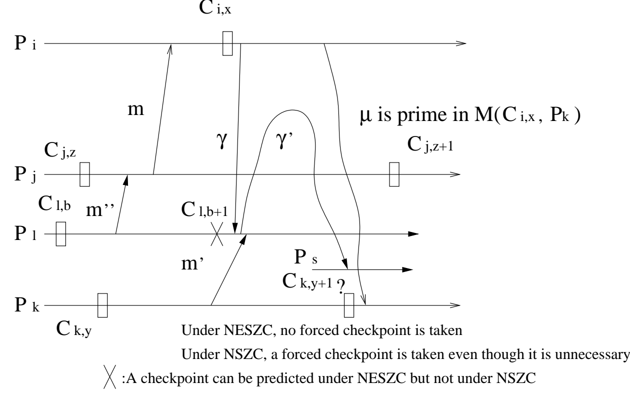


Figure 4.1: Example showing unnecessary forced checkpoints induced by the protocol in [34].

col.

4.1 An Example Showing Our Motivation

The model-based checkpointing protocol presented by Baldoni et al. [34] defines Suspect Z-cycle (SZC), a pattern that may lead to Z-cycles. A SZC is a Checkpoint and Communication Pattern $SZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ such that: $\exists m, m' : C_{j,z} \circ m \circ C_{i,x} \circ \mu \overset{k,y}{\bullet} m'$ with

$$\begin{cases} i & send(m) \in I_{j,z} \\ ii & \mu \text{ is prime in } M(C_{i,x}, P_k) \\ iii & \nexists e \in I_{j,z+1} : e \overset{hb}{\rightarrow} receive(\mu.last). \end{cases} \quad (4.1)$$

The authors in [34] proved that if there is a Z-cycle then there also exists a SZC. Therefore, eliminating all SZCs eliminates all Z-cycles. Besides, they designed a model-based checkpointing protocol that tracks SZCs and forces processes to take additional checkpoints to eliminate all SZCs, and hence all Z-cycles.

However, their protocol may take some unnecessary forced checkpoints while trying to eliminate SZCs. For example, in the CCP illustrated in Figure 4.1, upon the reception of $\mu.last$, the protocol in [34] will detect the $SZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$, since the CCP satisfies all the three conditions for the existence of an extended SZC, and

المنارة للاستشارات
www.manaraa.com

direct P_k to take the forced checkpoint $C_{k,y+1}$ to prevent the $SZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ detected. In this example, let us denote the $SZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$, which leads to the $ZC(C_{i,x}, \mu \overset{k,y}{\bullet} \zeta)$ where $\zeta = m' \bullet m'' \circ m$, by $\mathbf{Z_1}$. $C_{k,y+1}$ is however unnecessary since there exists another $SZC(I_{j,z}, C_{i,x}, \gamma, I_{l,b})$ in this CCP, which leads to the $ZC(C_{i,x}, \gamma \overset{k,y}{\bullet} \zeta')$ where $\zeta' = m'' \circ m$. Let us denote this SZC by $\mathbf{Z_2}$. Clearly $\mathbf{Z_2}$ satisfies all the following three conditions:

1. $send(m) \in I_{j,z}$
2. γ is prime in $M(C_{i,x}, P_l)$
3. $\nexists e \in I_{j,z+1} : e \overset{hb}{\rightarrow} receive(\gamma)$.

So, the protocol in [34] would have directed P_l to take a forced checkpoint $C_{l,b+1}$ before receiving γ to break $\mathbf{Z_2}$. As a result of P_l taking the checkpoint $C_{l,b+1}$, the Z-cycle, $ZC(C_{i,x}, \mu \overset{k,y}{\bullet} \zeta)$ where $\zeta = m' \bullet m'' \circ m$, has been broken, which makes the forced checkpoint $C_{k,y+1}$ unnecessary. So, upon receiving the message $\mu.last$, P_k does not have to take the forced checkpoint $C_{k,y+1}$. But the protocol proposed in [34] would still force P_k to take the forced checkpoint $C_{k,y+1}$. The reason for such an unnecessary forced checkpoint taken by the protocol in [34] is that the existence of $C_{l,b+1}$ is not captured by P_k upon the reception of $\mu.last$. This unawareness makes P_k to take the unnecessary forced checkpoint $C_{k,y+1}$. In order to know the existence of $C_{l,b+1}$, the following three types of information will suffice.

1. To whom and when P_k has sent messages, i.e., the destination of m'.
2. The existence of the causal path γ from P_i to P_l (in this example, γ is a single message), which forms the inner $SZC(I_{j,z}, C_{i,x}, \gamma, I_{l,b})$. γ can be tracked by P_k only if it can be included in the causal past of P_k. i.e., there exists a causal path γ' that can bring the information about the existence of γ to P_s before sending $\mu.last$.

المنارة للاستشارات
www.manaraa.com

3. The receiving of the message γ must occur before the receiving of m'.

Item 1 contains information about the causal future since m' is sent by P_k. On the other hand, items 2 and 3 can be tracked by P_k from the information about the causal past upon receiving $\mu.last$. i.e., P_k checks weather there exists γ that strongly visibly doubles $\mu \bullet m'$. Our goal is to design a protocol that captures this information and makes informed decision to take forced checkpoints.

4.2 The Sufficient Condition

To help track the formation of potential Z-cycles, we make the following definition.

Definition 15. *An* **Extended SZC** *(ESZC) is a Checkpoint and Communication Pattern* $ESZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ *such that:* $\exists m, m' : C_{j,z} \circ m \circ C_{i,x} \circ \mu \overset{k,y}{\bullet} m'$ *satisfying*

$$\begin{cases} (i) & send(m) \in I_{j,z} \\ (ii) & \mu \text{ is prime in } M(C_{i,x}, P_k) \\ (iii) & \nexists e \in I_{j,z+1} : e \overset{hb}{\to} receive(\mu.last) \\ (iv) & \mu \bullet m' \text{ is not strongly visibly doubled.} \end{cases} \tag{4.2}$$

Next, we prove that if there is a Z-cycle there must exist an $ESZC$. Much of the proof of this result is similar to the proof given in [34].

Lemma 4.2.1. *If there exists* $ZC(C_{i,x}, \mu \overset{k,y}{\bullet} \zeta)$ *such that* $|\zeta| = 1$ *then there exists an* $ESZC(I_{k,y}, C_{i,x}, \mu', I_{k,y})$.

Proof: Suppose a Z-cycle $ZC(C_{i,x}, \mu \overset{k,y}{\bullet} \zeta)$ with $|\zeta| = 1$ (i.e., $\zeta = m$). We suppose the prime causal chain in $M(C_{i,x}, P_k)$ is μ' (this chain exists as the set $M(C_{i,x}, P_k)$ contains at least μ). Since both $send(m)$ and $receive(\mu'.last)$ occur in the process P_k, either $receive(\mu'.last) \overset{hb}{\to} send(m)$ or $send(m) \overset{hb}{\to} receive(\mu'.last)$.

المنارة للاستشارات
www.manaraa.com

A. Suppose $receive(\mu'.last) \stackrel{hb}{\rightarrow} send(m)$. This is impossible because this implies $send(m) \stackrel{hb}{\rightarrow} receive(m) \stackrel{hb}{\rightarrow} C_{i,x} \stackrel{hb}{\rightarrow} send(\mu'.first) \stackrel{hb}{\rightarrow} receive(\mu'.last)$, a contradiction to the assumption $receive(\mu'.last) \stackrel{hb}{\rightarrow} send(m)$.

B. Suppose $send(m) \stackrel{hb}{\rightarrow} receive(\mu'.last)$. It has the following properties.

B.1 $\mu' \bullet m$ is not visibly doubled, and hence is not strongly visibly doubled, not violating condition (iv) of Definition 15. This is because $\mu' \bullet m$ is a Z-path from $I_{i,x}$ to $I_{i,x-1}$ (Definition 9).

B.2 Condition (iii) of Definition 15 is not violated. This is because if it is violated, then there exists an event $e \in I_{k,y+1}$ such that $e \stackrel{hb}{\rightarrow} receive(\mu'.last)$. Then, since $m \circ C_{i,x} \circ \mu' \stackrel{k,y}{\bullet} m$, $send(m) \in I_{k,y}$ and μ' is prime in $M(C_{i,x}, P_k)$, we know that $receive(\mu'.last) \in I_{k,y}$ and thus $receive(\mu'.last) \stackrel{hb}{\rightarrow} C_{k,y+1} \stackrel{hb}{\rightarrow} e$, which is a contradiction to the assumption $e \stackrel{hb}{\rightarrow} receive(\mu'.last)$.

Thus we showed in case B, conditions (iii) and (iv) of the Definition 15 are satisfied. Trivially, conditions (i) and (ii) are also satisfied.

Hence we conclude the existence of an $ESZC$ and the lemma follows. □

Next, we prove this result in the general case.

Theorem 4.2.2. *If there exists $ZC(C_{i,x}, \mu \stackrel{k,y}{\bullet} \zeta)$, then there exists an $ESZC$.*

Proof: Suppose a Z-cycle $ZC(C_{i,x}, \mu \stackrel{k,y}{\bullet} \zeta)$ exists. We use induction on the length of ζ. If $|\zeta| = 1$ then the theorem follows from Lemma 4.2.1. Next we assume that if there exists $ZC(C_{i,x}, \mu \stackrel{k,y}{\bullet} \zeta)$ where $|\zeta| < K$ $(K \geq 2)$, then there exists an $ESZC$ and prove that if there exists $ZC(C_{i,x}, \mu \stackrel{k,y}{\bullet} \zeta)$ where $|\zeta| = K$, then there exists an $ESZC$.

Let us consider $I_{j,z}$ such that $send(\zeta.last) \in I_{j,z}$ and a causal chain μ' is prime in $M(C_{i,x}, P_k)$ (this chain exists as the set $M(C_{i,x}, P_k)$ contains at least μ). Since events

www.manaraa.com

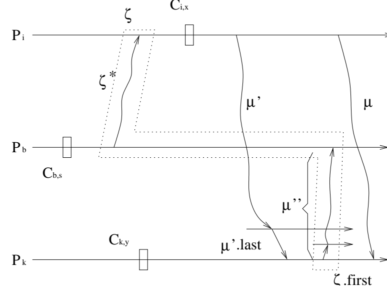


Figure 4.2: Proof for case A

$receive(\mu'.last)$ and $send(\zeta.first)$ occur in P_k, either $receive(\mu'.last) \stackrel{hb}{\rightarrow} send(\zeta.first)$ or $send(\zeta.first) \stackrel{hb}{\rightarrow} receive(\mu'.last)$:

A. If $receive(\mu'.last) \stackrel{hb}{\rightarrow} send(\zeta.first)$, we claim the existence of $ZC(C_{i,x}, [\mu' \circ \mu''] \stackrel{b,s}{\bullet} \zeta^*)$ where μ'' is a causal path and $\mu'' \stackrel{b,s}{\bullet} \zeta^* = \zeta$ for some b and s (see Figure 4.2), hence $|\zeta^*| < |\zeta|$. This is because the checkpoint interval $I_{b,s}$ must exist, otherwise the message chain ζ would be causal. If chain ζ is causal, we have $send(\zeta.first) \stackrel{hb}{\rightarrow} receive(\zeta.last) \stackrel{hb}{\rightarrow} C_{i,x} \stackrel{hb}{\rightarrow} send(\mu'.first) \stackrel{hb}{\rightarrow} receive(\mu'.last)$, a contradiction to the assumption $receive(\mu'.last) \stackrel{hb}{\rightarrow} send(\zeta.first)$.

B. If $send(\zeta.first) \stackrel{hb}{\rightarrow} receive(\mu'.last)$, we get the Z-cycle $ZC(C_{i,x}, \mu' \stackrel{k,y}{\bullet} \zeta)$. We have the following three sub-cases. Note that the three subcases cover all the possibilities, since $(iii \cap iv) \cup \overline{(iii) \cap (iv)} = (iii \cap iv) \cup \overline{(iii)} \cup \overline{(iv)}$.

B.1 $(iii \cap iv)$: i.e., $\nexists e \in I_{j,z+1} : e \stackrel{hb}{\rightarrow} receive(\mu'.last)$ and $\exists m'$: $\mu \bullet m'$ is not strongly visibly doubled. By definition, we get an $ESZC(I_{j,z}, C_{i,x}, \mu', I_{k,y})$ and the claim follows.

B.2 $\overline{(iii)}$: i.e., $\exists e \in I_{j,z+1} : e \stackrel{hb}{\rightarrow} receive(\mu'.last)$. This implies that there exists at least one causal message chain μ'' that starts after $C_{j,z+1}$ such that $\mu''.last$ is received by P_k before $\mu'.last$ or $receive(\mu''.last) = receive(\mu'.last)$. Since events $send(\zeta.first)$ and $receive(\mu''.last)$ occur in P_k, either $send(\zeta.first) \stackrel{hb}{\rightarrow} receive(\mu''.last)$ or $receive(\mu''.last) \stackrel{hb}{\rightarrow} send(\zeta.first)$. Hence the following two subcases arise:

المنارة للاستشارات
www.manaraa.com

(a) Suppose $send(\zeta.first) \xrightarrow{hb} receive(\mu''.last)$. Then we get $ZC(C_{j,z+1}, \mu'' \overset{k,y}{\bullet} \zeta^*)$ where $\zeta^* = \zeta - \zeta.last$ (see Figure 4.3.(a)), hence $|\zeta^*| < |\zeta|$.

(b) Suppose $receive(\mu''.last) \xrightarrow{hb} send(\zeta.first)$. We claim the existence of $ZC(C_{j,z+1}, [\mu'' \circ \mu'''] \overset{b,s}{\bullet} \zeta^*)$ where μ''' is a causal path and $\mu''' \overset{b,s}{\bullet} \zeta^* = \zeta - \zeta.last$ for some b and s (see Figure 4.3.(b)), hence $|\zeta^*| < |\zeta|$. This is because the checkpoint interval $I_{b,s}$ must exist, otherwise the message chain $\zeta - \zeta.last$ is causal. If $\zeta - \zeta.last$ is causal, we have $send(\zeta.first) \xrightarrow{hb} receive(\zeta^*.last) \xrightarrow{hb} C_{j,z+1} \xrightarrow{hb} send(\mu''.first) \xrightarrow{hb} receive(\mu''.last)$, a contradiction to the assumption $receive(\mu''.last) \xrightarrow{hb} send(\zeta.first)$.

B.3 $\overline{(iv)}$: i.e., $\forall m'$: $\mu \bullet m'$ is strongly visibly doubled by causal chain γ. Suppose m' is sent to P_c. Since events $send((\zeta - m').first)$ and $receive(\gamma.last)$ occur in P_c, either $send((\zeta - m').first) \xrightarrow{hb} receive(\gamma.last)$ or $receive(\gamma.last) \xrightarrow{hb} send((\zeta - m').first)$:

(a) If $send((\zeta - m').first) \xrightarrow{hb} receive(\gamma.last)$. Then, there is a $ZC(C_{i,x}, \gamma \overset{c,t}{\bullet} (\zeta - m'))$. Let $\zeta^* = \zeta - m'$ (see Figure 4.4.(a)), hence $|\zeta^*| < |\zeta|$.

(b) If $receive(\gamma.last) \xrightarrow{hb} send((\zeta - m').first)$, we claim the existence of $ZC(C_{i,x}, [\gamma \circ \mu''] \overset{b,s}{\bullet} \zeta^*)$, where μ'' is a causal path and $\mu'' \overset{b,s}{\bullet} \zeta^* = \zeta - m'$ for some b and s (see Figure 4.4.(b)), hence $|\zeta^*| < |\zeta|$. The checkpoint interval $I_{b,s}$ must exist, otherwise the message chain $\zeta - m'$ is causal. If $\zeta - m'$ is causal, we have $send((\zeta - m').first) \xrightarrow{hb} receive((\zeta - m').last) \xrightarrow{hb} C_{i,x} \xrightarrow{hb} send(\gamma.first) \xrightarrow{hb} receive(\gamma.last)$, a contradiction to the assumption $receive(\gamma.last) \xrightarrow{hb} send((\zeta - m').first)$.

Thus we have proved that if there exists a $ZC(C_{i,x}, \mu \overset{k,y}{\bullet} \zeta)$ where $|\zeta| = K$, it is either already an $ESZC(I_{j,z}, C_{i,x}, \mu', I_{k,y})$ or it contains a Z-cycle with $|\zeta^*| < |\zeta| = K$. So, by the induction hypothesis, the theorem is true.□

المنارة للاستشارات
www.manaraa.com

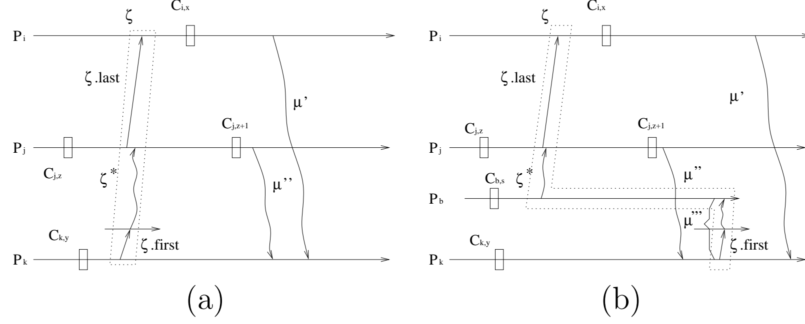


Figure 4.3: Illustration of proof for case B.2

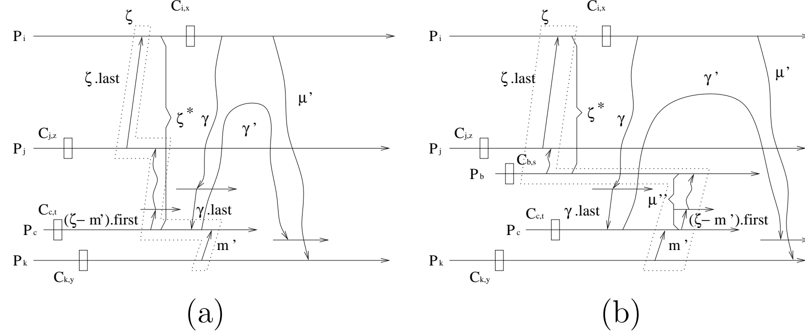


Figure 4.4: Illustration of proof for case B.3

Theorem 4.2.3. *If a checkpoint and communication pattern $(\widehat{H}, C_{\widehat{H}})$ of a distributed computation satisfies the No-Extended-Suspect-Z-cycle ($NESZC$) property, (i.e., no $ESZC$ exists in $(\widehat{H}, C_{\widehat{H}})$), then it satisfies the NZC property.*

Proof: By Theorem 4.2.2, if a ZC exists then an $ESZC$ exists in $(\widehat{H}, C_{\widehat{H}})$. i.e., $ZC \Rightarrow ESZC$. Hence $NESZC \Rightarrow NZC$.□

The following theorem shows the relation between the protocol in [34] ensuring No-Suspect-Z-cycle ($NSZC$) and our protocol ensuring $NESZC$.

Theorem 4.2.4. *If our protocol induces a forced checkpoint for ensuring $NESZC$, the protocol in [34] also induces a forced checkpoint for ensuring $NSZC$.*

Proof: Since an $ESZC$ is also a SZC, the theorem follows from Theorem 4.2.3.□

The converse may not be true. For example in Figure 4.1, there exists a $SZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ but no $ESZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ exists. Our protocol uses a stronger condition than the one used by Baldoni et al. [34] to induce forced checkpoints. So, our protocol

المنارة للاستشارات
www.manaraa.com

is likely to induce less forced checkpoints, in general.

4.3 Relation Between Existing Model-based Checkpointing Protocols

Theorem 4.2.2 shows if there exists a Z-cycle, then there exists an $ESZC$ and the converse may not be true. Therefore the set of $ESZC$s contains the set of Z-cycles. Similarly, from the proof of Theorem 4.2.4, if there exists an $ESZC$, then there exists a SZC and the converse may not be true. Hence, the set of SZCs contains the set of $ESZC$s. The concept of Core-Z-cycle (CZC) introduced in [52] is essentially another term of SZC. Trivially, if there exists an SZC, then there exists a Z-path. But the converse in not true in general. Hence the set of Z-paths contains the set of SZCs. Note that RDT protocols prevent the formation of Z-paths.

Protocol in [45] interprets the concept of SZC from a different perspective, the obsoleteness of a checkpoint. A checkpoint C_1 is said to be obsolete to the most recent checkpoint C_2 on P_i if $C_1 \stackrel{hb}{\rightarrow} C_2$. By checking the obsoleteness of checkpoints, protocol in [45] prevents the formation of potential Z-cycles. An essential component of SZC is that it checks if $\exists e \in I_{j,z+1} : e \stackrel{hb}{\rightarrow} receive(\mu)$, which is same as checking if checkpoint $C_{j,z}$ is obsolete with respect to $C_{k,y+1}$ since $C_{j,z} \stackrel{hb}{\rightarrow} C_{k,y+1}$ (Figure 4.5.(a)). However the converse may not be true. In Figure 4.5.(b), there is no $\exists e \in I_{j,z+1} : e \stackrel{hb}{\rightarrow} receive(\mu)$ but $C_{j,z}$ is still obsolete since $C_{j,z} \stackrel{hb}{\rightarrow} C_{k,y+1}$ through P_b. Therefore, if there exists a pattern satisfying $\exists e \in I_{j,z+1} : e \stackrel{hb}{\rightarrow} receive(\mu)$, there exists the obsoleteness of $C_{j,z}$ but the converse may not be true. Since the definition of SZC and the protocol in [45] contains the negation of $\exists e \in I_{j,z+1} : e \stackrel{hb}{\rightarrow} receive(\mu)$ and obsoleteness respectively, we derive the following: the communication pattern detected and prevented by protocol in [45], contains an SZC; but if there is an SZC, it may not contain a communication pattern detected by the protocol in [45].

Now we will discuss the relationship between of our protocol and protocols in [34, 45, 52]. Under all these three protocols [34, 45, 52], a process, based on local infor-

للاستشارات

www.manaraa.com

mation and information received in messages from other processes, tries detect if a Z-cycle is likely to form. For example, in the communication pattern illustrated in Figure 4.1, upon receiving $\mu.last$, protocols in [34, 45, 52] will make P_k only to check if a potential Z-cycle will be formed based on the information it has and it does not take into consideration the fact that P_l (the process to which P_k has sent a message) has also detected a formation of the same potential Z-cycle based on the information it has. This potential Z-cycle detected by P_l makes the detection of the potential Z-cycle by P_k redundant. Therefore, in order for P_k to make more informed decision regarding the detection of potential Z-cycles, our protocol uses additional information (i.e., the condition *iv* in Definition 15) to check if other processes, especially those it has sent messages to, have detected potential Z-cycles. This is the essence of our proposed protocol. Since our protocol gathers Z-cycle information from both itself and some other processes, it makes a more informed decision regarding the formation of potential Z-cycles. In other words, if there exists a $ESZC$, there must exist a communication pattern detected by protocols in [34, 45, 52] to prevent Z-cycles. However, the converse is not true in general. Hence, each of the communication patterns detected and prevented by the algorithms in [34, 45, 52] contain $ESZC$.

In other words, based on these discussions, we can conclude the following. Existence of ZC implies existence of $ESZC$; existence of $ESZC$ implies existence of communication pattern detected by the protocol in [45]; existence of communication pattern detected by protocol in [45] implies existence of SZC; existence of SZC implies the existence of $Z - path$. Thus, the set of communication patterns detected and prevented by our protocol is the smallest of the sets of communication patterns detected and prevented by all these other protocols. This relationship between our protocol and the set of communication patterns detected and prevented by protocols in [34, 45, 52] is given in Figure 4.6. In the next section, we present our protocol which tracks and prevents the formation of $ESZCs$.

www.manaraa.com

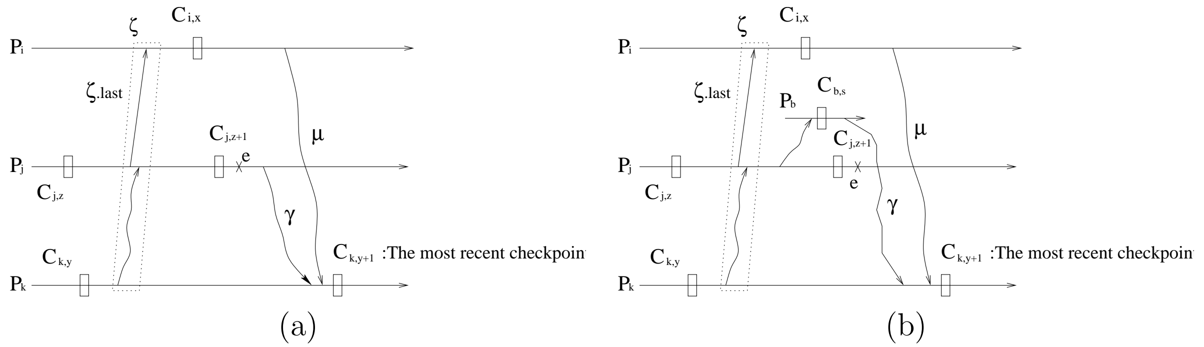


Figure 4.5: Essential components of Baldoni's and Garcia's protocols

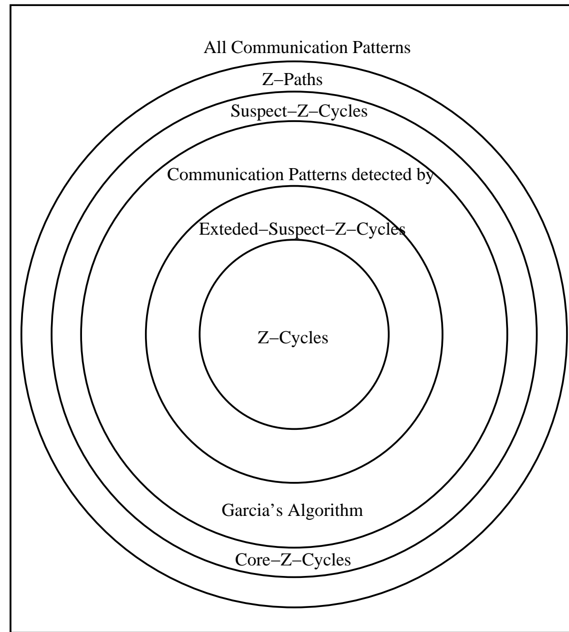


Figure 4.6: Relation between some existing model-based communication-induced checkpointing protocols

4.4 An Enhanced Model-based Checkpointing Protocol

The protocol presented in this section ensures all checkpoints to be useful by preventing the formation of $ESZC$s on-the-fly. In order to track the formation of $ESZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$, whenever a process receives a message $m = \mu.last$ from some other process, it checks if the conditions for the existence of $ESZC$ are satisfied. If so, then it takes a forced checkpoint before processing the message m. We first introduce the data structures and the predicates for tracking $ESZCs$. Some of the data

www.manaraa.com

structures used are similar to the ones used in [34] so we can compare easily.

How to track $\mu \overset{k,y}{\bullet} m'$ where μ is prime in $M(C_{i,x}, P_k)$

To detect if μ is prime in $M(C_{i,x}, P_k)$, we have to use the vector VC, described in Section 1.2.1. When a process P_k receives a message m, if the predicate $\exists i$: $(m.VC[i] > VC_k[i])$ hold, then P_k can conclude that the causal message chain μ where $m = \mu.last$ is prime in $M(C_{i,x}, P_k)$. To detect if a non-causal concatenation in the interval $I_{k,y}$, involving the prime causal message chain μ and a message m' exists, a boolean variable $after_first_send_k$ is maintained by P_k. $after_first_send_k$ is set to TRUE when the first send event occurs in a checkpoint interval of P_k. It is set to FALSE each time a local checkpoint is taken. Hence, upon receiving a message m (with $m = \mu.last$), P_k detects that $\mu \overset{k,y}{\bullet} m'$ where μ is prime in $M(C_{i,x}, P_k)$ if the following predicate holds: $(after_first_send_k \wedge (\exists i : m.VC[i] > VC_k[i]))$.

How to track $C_{j,z} \circ m \circ C_{i,x}$

Each process P_k maintains a vector of integers Imm_Pred_k of size n and a matrix of integers $Pred_k$, of size $n \times n$ [34]. $Imm_Pred_k[\ell]$ represents the latest checkpoint interval from which process P_ℓ sent a message m to P_k which has been delivered to P_k in its current checkpoint interval, say $I_{k,y-1}$ (in other words, $C_{\ell,Imm_Pred[\ell]}$ is an immediate predecessor of checkpoint $C_{k,y}$, once $C_{k,y}$ is taken). All the entries of this vector are set to -1 each time a checkpoint is taken by P_k. $Pred_k[i,j]$ represents, to the knowledge of P_k, the highest checkpoint interval from which process P_j sent a message m which has been delivered by P_i in a checkpoint interval $I_{i,x-1}$ with $x \leq VC_k[i]$ (note that $x \leq VC_k[i]$ simply means that the information about $C_{i,x}$ has been brought to P_k through a message). Similarly all the entries of the matrix $Pred_k$ are initialized to -1, and its content is piggybacked on each message m sent by P_k (denoted as $m.Pred$). The rule to update its entries are the following as given in [34].

المنارة للاستشارات
www.manaraa.com

1. whenever a checkpoint is taken by P_k, $\forall j$: $Pred_k[k,j] = max(Pred_k[k,j], Imm_Pred_k[j])$;

2. upon the arrival of a message m at P_k, $\forall \ell, t$: $Pred_k[\ell][t] = max(Pred_k[\ell,t], m.Pred[\ell,t])$.

By examining the matrix $Pred_k$, P_k can determine the existence of $C_{j,z} \circ m \circ C_{i,x}$ if the entry $Pred_k[i,j]$ is equal to z. Since we don't worry about the value of z and only worry about the existence of the pattern $C_{j,z} \circ m \circ C_{i,x}$, as long as $Pred_k[i,j]$ is not -1, we know the pattern $C_{j,z} \circ m \circ C_{i,x}$ exists.

How to determine $\nexists e \in I_{j,z+1} : e \xrightarrow{hb} receive(\mu.last)$

Upon the arrival of a message m included in a prime causal chain (i.e., $\exists i : (m.VC[i] > VC_k[i])$), in order to track the above condition, we need to know if there exists a j (i.e., P_j) such that $m.Pred[i,j] + 1$ does not belong to the causal past of the receive event of m. There are two possibilities for $m.Pred[i,j] + 1$ to belong to the causal past of the receive event of m. The information about $m.Pred[i,j] + 1$ could have been brought to P_k by m or can be bought to P_k by some other causal message chain before receiving m. The first possibility is captured in $m.VC[j]$, and the second is captured in $VC_k[j]$. Hence the predicate becomes: $(\exists j : m.Pred[i,j]+1 > m.VC[j] \wedge m.Pred[i,j]+1 > VC_k[j])$, or simply $(\exists j : m.Pred[i,j]+1 > max(m.VC[j], VC_k[j]))$. Since $m.VC[j]$ and $VC_k[j]$ can not be less than 0, $m.Pred[i,j] + 1$ can not be less than 1 and thus $m.Pred[i,j]$ can not be -1. Therefore the predicate which tracks $\nexists e \in I_{j,z+1} : e \xrightarrow{hb} receive(\mu.last)$ also tracks $C_{j,z} \circ m \circ C_{i,x}$.

How to check if $\mu \bullet m'$ is not strongly visibly doubled.

Each process P_i can use single matrix $causal_i$ to check if $\mu \bullet m'$ is not strongly visibly doubled. However, this simple data structure can only track a subset of all the

المنارة للاستشارات
www.manaraa.com

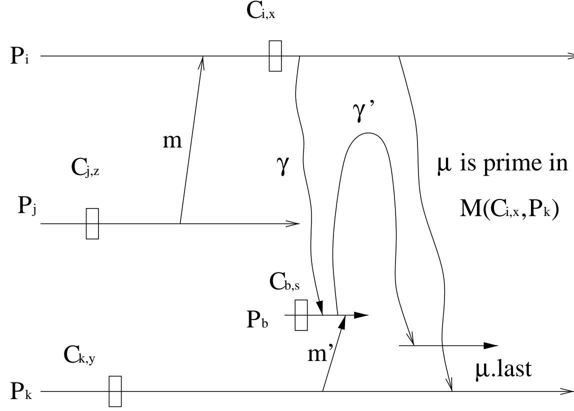


Figure 4.7: Case $send(\gamma'.first) \xrightarrow{hb} receive(m')$

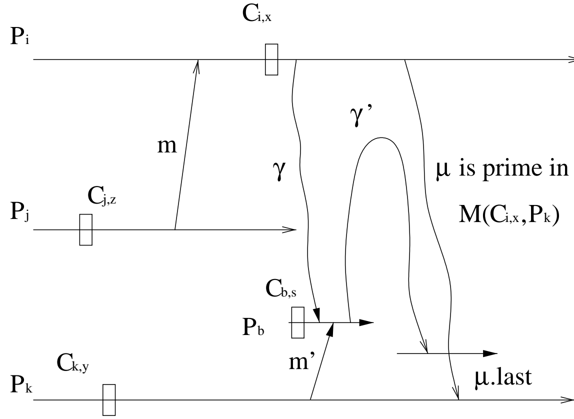


Figure 4.8: Case $receive(m') \xrightarrow{hb} send(\gamma'.first)$

strongly visible doublings. For example, in Figure 4.7, the causal path γ' that brings the strong visible doubling information to P_k happens before receiving message m' (i.e., $send(\gamma'.first) \xrightarrow{hb} receive(m')$).

In order to improve the tracking of strong visible doubling, we need to design more sophisticated data structures to track the situations such as the one illustrated in Figure 4.8 as well, wherein the causal path γ' that brings the strong visible doubling information to P_k happens after the receive of message m' (i.e., $receive(m') \xrightarrow{hb} send(\gamma'.first)$). Besides the matrix *causal* that can tell us the visibly doubling information, we need extra data structure to track if $receive(\gamma.last) \xrightarrow{hb} receive(m')$ to satisfy the definition of strongly visible doubling.

المنارة للاستشارات
www.manaraa.com

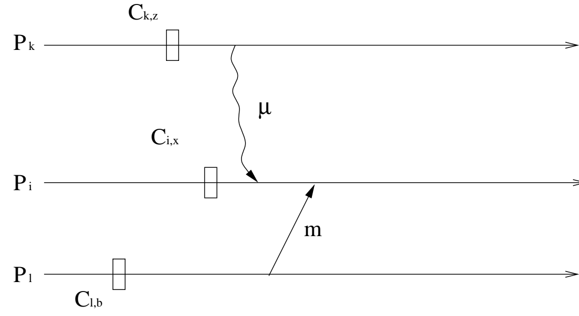


Figure 4.9: Case $i = j$

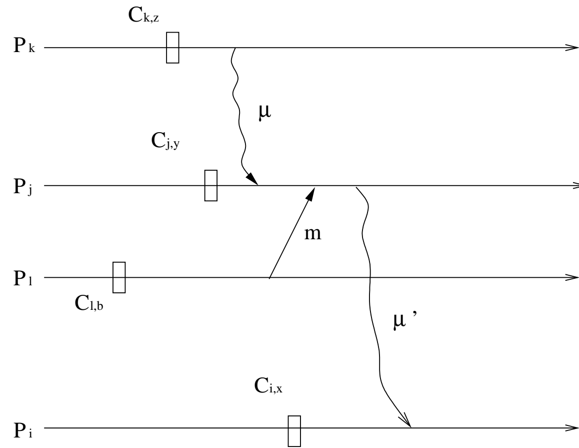


Figure 4.10: Case $i \neq j$

So, for tracking strong visible doubling, we require that each process P_i maintains n boolean matrices of size $n \times n$, $Pre_i^j : 1 \leq j \leq n$. If $i = j$, i.e., $Pre_i^i[k][l] : 1 \leq k, l \leq n$ is TRUE (see Figure 4.9) if in the current checkpoint interval of P_i, (i) P_i has received the last message $\mu.last$ of a causal path μ that originated from P_k, (ii) P_i received the first message m from P_l in its current checkpoint interval; (iii) $receive(\mu.last) \stackrel{hb}{\rightarrow} receive(m)$. Similarly, if $i \neq j$, $Pre_i^j[k][l] : 1 \leq k, l \leq n$ is TRUE (see Figure 4.10) if P_i, in its current checkpoint interval, has received a message m' which is the last message of a causal path mu' (i.e., $mu'.last = m'$ that originated from P_j such that (i) P_j has the knowledge of P_k through causal path μ, (ii) P_j receives the first message m from P_l a nd (iii) this knowledge about P_k happens before the receiving of the first message from P_l.

Process P_i initializes Pre_i as follows: $Pre_i^i[k, l] = TRUE,\ if\ i = k = l$; the rest

المنارة للاستشارات
www.manaraa.com

of the entries of $Pre_i^i[k,l]$ are set to $FALSE$. Similarly, $Pre_i^j[k,l] = TRUE\ if\ i \neq j \wedge (k = l = j)$; the rest of the entries of this matrix are set to $FALSE$. Whenever a new checkpoint is taken by P_i, the matrices $Pre_i^j : 1 \leq j \leq n$ are re-initialized.

When P_s sends a message m, the ℓ matrices $Pre_s^\ell : 1 \leq \ell \leq n$ are piggybacked with m, denoted as $m.Pre^\ell : 1 \leq \ell \leq n$. When a message m, sent by P_s is delivered to P_i, $Pre_i^j[k][l] : 1 \leq j,k,l \leq n$ are updated as follows:

1. For each h such that $m.VC[h] > VC_i[h](h \neq i)$: for every $\ell, \jmath$, $Pre_i^h[\ell, \jmath]$ is set to $m.Pre^h[\ell, \jmath]$.

2. For each h such that $m.VC[h] = VC_i[h](h \neq i)$: for every $\ell, \jmath$, $Pre_i^h[\ell, \jmath]$ is updated to $Pre_i^h[\ell, \jmath] \vee m.Pre^h[\ell, \jmath]$.

3. Pre_i^i is updated as follows: If m is the first message received from P_s in its current checkpoint interval, Pre_i^i is updated using the following rules; otherwise, Pre_i^i is not updated. In order to track if a messages is the first message from a process in its current checkpoint interval, P_i maintains a Boolean array $first_receive_i$ of size n. $first_receive_i[j]$ is set to TRUE when P_i receives the first message from P_j in its current checkpoint interval; it is reset to false when a new checkpoint is taken. Therefore, Pre_i^i is updated, only if upon receiving m $first_receive_i[s] = FALSE$.

   1) Since any causal path that has already been recorded in Pre_i^i must happen before the receiving of message m, this information need to be updated. We introduce a new array of boolean variables $ExistCausal_i^i[\ell] : 1 \leq \ell \leq n$, and $(ExistCausal_i^i[\ell] = TRUE)\ iff\ (\exists \jmath : Pre_i^i[\ell][\jmath] = TRUE)$. If $ExistCausal_i^i[\ell] = TRUE$, it means there exists a causal path from a checkpoint in process P_ℓ to a checkpoint in P_i. Therefore, we set $Pre_i^i[\ell][s] = TRUE$ if $ExistCausal_i^i[\ell] = TRUE$. This procedure can be illustrated using Figure 4.11. In Figure 4.11, when P_i receives m from

المنارة للاستشارات
www.manaraa.com

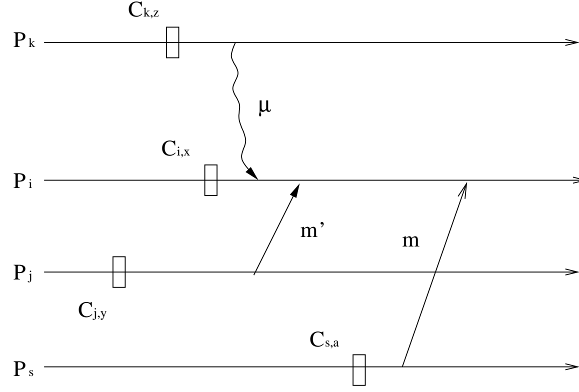


Figure 4.11: Case (3.1)

P_s, it knows it already has a causal path to its current checkpoint interval from P_k since $Pre_i^i[k][j] = TRUE$. Then, $ExistCausal_i^i[k] = TRUE$ and hence we set $Pre_i^i[k][s] = TRUE$. This basically means, $\forall \ell : Pre_i^i[\ell][s] = Pre_i^i[\ell][s] \vee ExistCausal_i^i[\ell]$ where $(ExistCausal_i^i[\ell] = TRUE)\ iff\ (\exists \jmath : Pre_i^i[\ell][\jmath] = TRUE)$.

2) Set $Pre_i^i[s][s] = TRUE$, since there is a causal path from P_s to the current checkpoint interval of P_i.

3) We also have to take into account the transitive dependency caused by the receiving of m. For example, in Figure 4.12, to the knowledge of P_i, P_s has the knowledge of P_k $(k \neq i)$ which precedes the receiving of m' from P_l. i.e., $Pre_i^s[k][l] = TRUE$. So, we set $Pre_i^i[k][s] = TRUE$, due to the causal dependency captured by the causal path $\mu \bullet m$ from P_k to P_i. In other words, $\forall \ell (\ell \neq i) : Pre_i^i[\ell][s] = Pre_i^i[\ell][s] \vee ExistCausal_i^s[\ell]$ where $(ExistCausal_i^s[\ell] = TRUE)\ iff\ (\exists \jmath : Pre_i^s[\ell][\jmath] = TRUE)$.

Each process P_i also needs to maintain a vector $send_list_i$ to track the tuple: to whom and when P_i has sent the first message in its current checkpoint interval. If P_i sends the *first message m in its current checkpoint interval* to P_j and the current value of VC when sending m is VC_i^m, $send_list_i$ is updated as follows: $send_list_i = send_list_i \cup (P_j, VC_i^m)$.

المنارة للاستشارات
www.manaraa.com

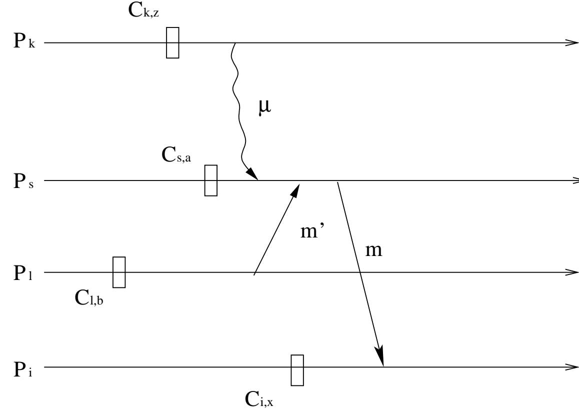


Figure 4.12: Case (3.3)

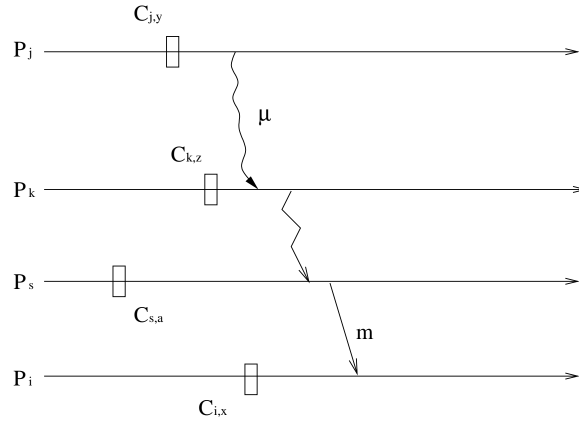


Figure 4.13: Example showing matrix *causal*

With the help of the data structure *send_list* and Pre, we can track the strongly visibly doubling better. For example, in Figure 4.7, if $VC_i^{m'} \not\leq m.VC$, when P_k receives $m = \mu.last$, P_k can track whether PCM path $\mu \bullet m'$ is strongly visibly doubled by simply checking if $\exists \jmath : m.Pre^j[i][\jmath] = TRUE$. In Figure 4.8, for example, if $VC_i^{m'} \leq m.VC$ when P_k receives $m = \mu.last$, P_k can check if PCM path $\mu \bullet m'$ is strongly visibly doubled by checking if $m.Pre^j[i][k] = TRUE$. In summary, when P_k receives $\mu.last$, if $\neg((VC_i^{m'} \not\leq m.VC) \wedge (\exists \jmath : m.Pre^j[i][\jmath] = TRUE)) \vee (VC_i^{m'} \leq m.VC \wedge m.Pre^j[i][k]))$, P_k knows that PCM path $\mu \bullet m'$ is not strongly visibly doubled. Therefore a forced checkpoint needs to be taken.

المنارة للاستشارات
www.manaraa.com

How to prevent $ESZC$

Our goal is to design a Communication-Induced Checkpointing protocol that eliminates $ESZC$s. Based on the discussion above, if a process receives a message and detects that all the predicates mentioned above are satisfied, it detects the existence of an $ESZC$ and takes a forced checkpoint to eliminate the $ESZC$. From the discussions in Section 4.4 to 4.4, we arrive at the following condition for preventing $ESZC$s. Upon the arrival of a message m (i.e., $\mu.last = m$) at process P_k in the checkpoint interval $I_{k,y}$, if the predicate $(after_first_send_k \wedge (\exists i : m.VC[i] > VC_k[i]) \wedge (\exists j_1 : m.Pred[i, j_1] + 1 > max(m.VC[j_1], VC_k[j_1])) \wedge (\exists j_2 : (P_{j_2}, VC_k^{m'}) \in send_list_k \wedge \neg((VC_i^{m'} \nleq m.VC \wedge (\exists j : m.Pre^{j_2}[i][j] = TRUE)) \vee (VC_i^{m'} \leq m.VC) \wedge m.Pre^{j_2}[i][k]))))$ holds, P_k detects that at least one $ESZC(I_{j_1,Pred_k[i,j_1]}, C_{i,x}, \mu, I_{k,y})$ is being formed. In this case, P_k takes a forced checkpoint $C_{k,y+1}$ before processing m. Following is the formal description of our protocol that prevents the formation of $ESZC$ and hence Z-cycles.

Initialization at process P_k:

Take an initial checkpoint;

$after_first_send_k := FALSE$;

$\forall i : i \neq k, VC_k[i] := 0; VC_k[k] := 1$;

$\forall i, j : k = i = j, Pre_k^k[i, j] := TRUE \wedge \forall i, j, l : i \neq k \wedge i = j = l, Pre_k^i[j, l] := TRUE$

else $Pre_k^i[j, l] := FALSE$;

$\forall i, j : Pred_k[i, j] := -1$;

$\forall h : Imm_Pred_k[h] := -1$;

$\forall h : k = h, first_receive_k[h] := TRUE; k \neq h, first_receive_k[h] := FALSE$;

$send_list_k = \emptyset$;

When a message m arrives at P_k from P_s:

المنارة للاستشارات
www.manaraa.com

if $(after_first_send_k \wedge (\exists i : m.VC[i] > VC_k[i]) \wedge (\exists j_1 : m.Pred[i, j_1] + 1 > max(m.VC[j_1], VC_k[j_1])) \wedge (\exists j_2 : (P_{j_2}, VC_k^{m'}) \in send_list_k \wedge \neg((VC_k^{m'} \not\leq m.VC \wedge (\exists \jmath : m.Pre^{j_2}[i][\jmath] = TRUE)) \vee (VC_k^{m'} \leq m.VC \wedge m.Pre^{j_2}[i][k]))))$

then take_ckpt(P_k); %take a forced checkpoint%

$\forall h$: {

if $(m.VC[h] > VC_k[h])$

then $\{VC_k[h] := m.VC[h];$

$\forall \ell, \jmath,\ Pre_k^h[\ell, \jmath] := m.Pre^h[\ell, \jmath];\}$

if $(m.VC[h] = VC_k[h])$

then $\forall \ell, \jmath,\ Pre_k^h[\ell, \jmath] := Pre_k^h[\ell, \jmath] \vee m.Pre^h[\ell, \jmath];$

}

if $(\neg first_receive_k[s])$ then $\{first_receive_k[s] := TRUE;$

$\forall \ell : Pre_k^k[\ell][s] = Pre_k^k[\ell][s] \vee (\exists \jmath : Pre_k^k[\ell][\jmath] = TRUE);$

$Pre_k^k[s][s] = TRUE;$

$\forall \ell(\ell \neq k) : Pre_k^k[\ell][s] = Pre_k^k[\ell][s] \vee (\exists \jmath : Pre_k^s[\ell][\jmath] = TRUE);\}$

$\forall i, j : Pred_k[i][j] := max(m.Pred[i, j], Pred_k[i, j]);$

$Imm_Pred_k[s] := max(Imm_Pred_k[s], m.VC[s]);$

$deliver(m);$

Procedure send(m, P_k, P_j): %send message m from P_k to P_j%

$m.content := data;$

$m.VC := VC_k;$

$m.Pred := Pred_k;$

$m.Pre := Pre_k;$

المنارة للاستشارات
www.manaraa.com

if (m is the first message sent to P_j in P_k's current checkpoint interval)

then $send_list_k := send_list_k \cup (P_j, VC_k^m)$; %$VC_k^m$ denotes the value of VC_k while sending m%

send m from P_k to P_j;

$after_first_send_k := TRUE$;

When a basic checkpoint is scheduled for P_k:

take_ckpt(P_k); %take a basic checkpoint%

Procedure take_ckpt(P_k):

Take a checkpoint;

$\forall h : Pred_k[k,h] := max(Pred_k[k,h], Imm_Pred_k[h])$;

$\forall h : Imm_Pred_k[h] := -1$;

$VC_k[k] := VC_k[k] + 1$;

$after_first_send_k := FALSE$;

$\forall i,j : k = i = j, Pre_k^k[i,j] := TRUE \wedge \forall i,j,l : i \neq k \wedge i = j = l, Pre_k^i[j,l] := TRUE$

else $Pre_k^i[j,l] := FALSE$;

$\forall h : k = h, first_receive_k[h] := TRUE; k \neq h, first_receive_k[h] := FALSE$;

$send_list_k = \emptyset$;

4.5 Simulation Results

We compared the performance of our protocol with the protocol of Baldoni et al. [34] through simulation. For the evaluation, we simulated up to 50 distributed computations, the number of processes involved in each computation varying from 8 to 25. Our goal was to study the effect of communication pattern on the number of forced

www.manaraa.com

checkpoints taken. In each computation, a fixed number of messages were exchanged. The number of messages exchanged varied from 1500 to 1800. A process takes a basic checkpoint after receiving a certain number of messages. This number was varied from 25 to 50 for various scenarios. The number of forced checkpoints taken was recorded under both the protocols for each scenario. For example, Figure 4.14 gives the simulation result for which 50 distributed computations were simulated (i.e., $SN = 50$). Each distributed computation consisted of 8 processes and each computation exchanged 1500 ($M = 1500$) messages. Along the x-axis, we listed the 50 distributed computations. Corresponding to each distributed computation, we plotted the number of forced checkpoints taken by each of these computations along the y-axis (to save space, the y-axis does not start at 0). A process takes a basic checkpoint after receiving a random number of messages (the random number being chosen between 20 and 25) since it took the last checkpoint (forced or basic). This interval for the number of messages received before taking a basic checkpoint is indicated in the figure as $IN = 25 - 35$.

We use a centralized method to simulate a distributed computation to simplify our programming. Instead of letting an individual process to determine when, how and to whom to send a message, messages are generated one-by-one by a central generator. Secondly, we randomize the origin and destination of each message created by the generator to make all the messages evenly distributed in the system.

Simulation results show that our protocol has a significant improvement over the protocol of Baldoni et al. [34] in terms of the number of forced checkpoints taken. In this case, the Average Improvement on each sample computation is 29.87% (represented by $AI = 29.87\%$). We varied the number of messages exchanged to 1500, 1600 and 1800 with the respective number of processes 15, 20 and 25; we also varied the parameter IN to 25-35, 40, 30-50 respectively. Figures 4.15, 4.16, and 4.17 show the results of our simulation for these scenarios. Under each scenario, the average

المنارة للاستشارات
www.manaraa.com

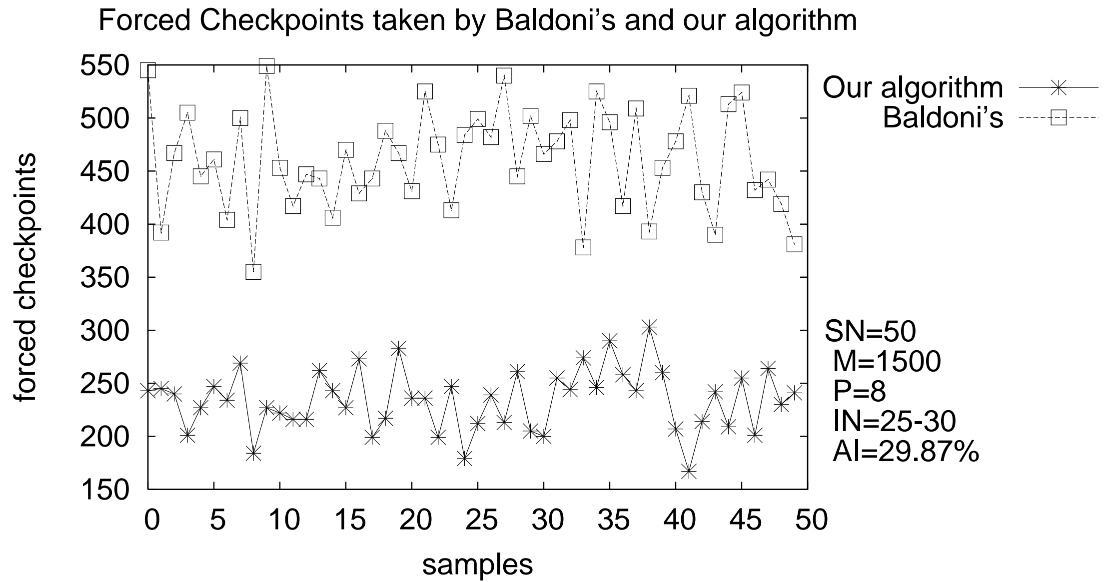


Figure 4.14: The results of our simulation for 50 random distributed computations each of which consisted of 8 processes ($P = 8$), exchanged 1500 messages and took basic checkpoints with parameter IN=25-30

improvement (AI) in terms of the number of forced checkpoints taken by processes is listed on the lower right corner of each figure. These results show that our protocol consistently takes significantly less number of forced checkpoints compared to the protocol of Baldoni et al. This is because our protocol is capable of tracking the formation of communication patterns that can lead to Z-cycles more efficiently and prevent them by taking less number of forced checkpoints.

4.6 Conclusion

In this chapter, we presented an enhanced model-based checkpointing protocol. Our protocol piggybacks an extra boolean array Pre of size $O(n^3)$ with each application message compared to the protocol in [34] to help eliminate Z-cycles in a more informed way. This extra information helps in detecting the formation of $ESZCs$ which uses a stronger predicate compared to the one used for detecting $SZCs$ in [34]. As a result of this stronger condition to detect potential Z-cycles, our protocol induces less

المنارة للاستشارات
www.manaraa.com

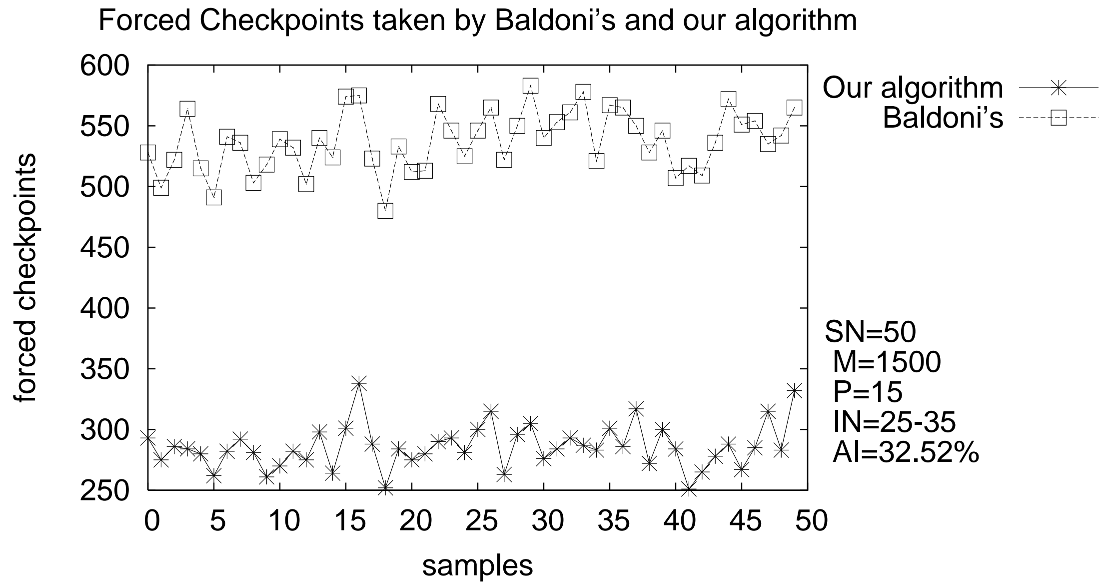


Figure 4.15: The results of our simulation for 50 random distributed computations each of which consisted of 15 processes, exchanged 1500 messages and took basic checkpoints with parameter IN=25-30

number of forced checkpoints than [34] as confirmed by our simulation results.

المنارة للاستشارات
www.manaraa.com

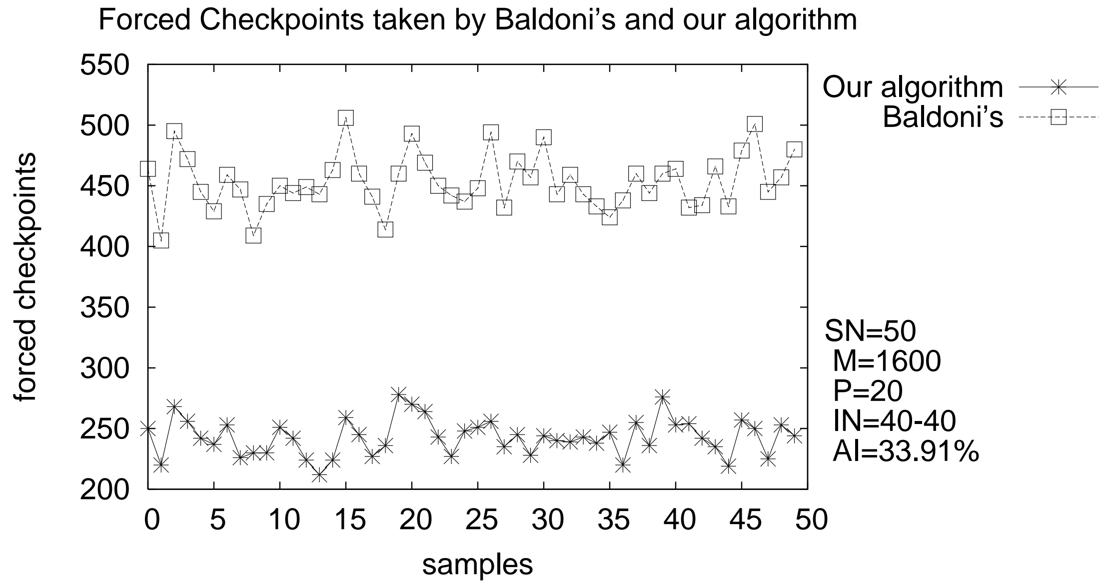


Figure 4.16: The results of our simulation for 50 random distributed computations each of which consisted of 20 processes, exchanged 1600 messages and took basic checkpoints with parameter IN=40-40

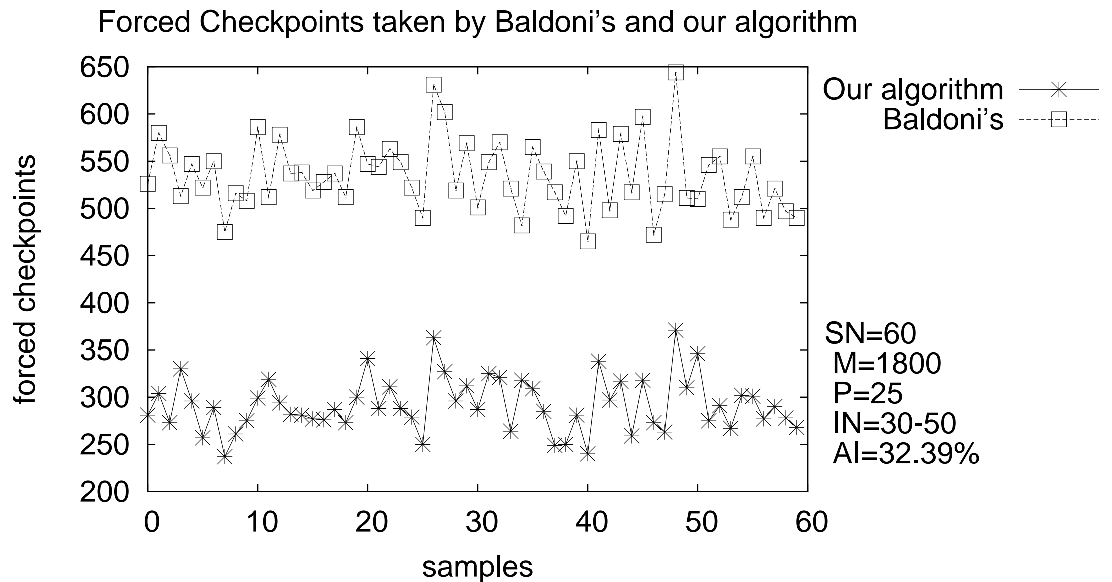


Figure 4.17: The results of our simulation for 60 random distributed computations each of which consisted of 25 processes, exchanged 1800 messages and took basic checkpoints with parameter IN=30-50

المنارة للاستشارات
www.manaraa.com

Chapter 5

Conclusion and Future Work

In this dissertation, we presented our research results in the area of fault-tolerance in distributed systems and distributed database systems. We established the necessary and sufficient conditions for a set of checkpoints of a set of data items to be part of a transaction-consistent global checkpoint and also presented a non-intrusive checkpointing protocol for distributed database systems. We then presented a low-overhead model-based communication-induced checkpointing protocol for distributed systems.

5.1 Research Accomplishments

Necessary and Sufficient Conditions

Netzer and Xu [7] introduced the notion of zigzag paths between checkpoints of processes involved in a distributed computation running in a distributed system. Based on this they also presented the necessary and sufficient conditions for a set of checkpoints of a set of processes involved in a distributed computation to be part of a consistent global checkpoint of the distributed computation. We generalized this notion of zigzag paths between checkpoints of processes involved in a distributed computation to zigzag paths between checkpoints of data items in a distributed database. Based on this generalization, we developed the necessary and sufficient conditions for a set of checkpoints of a set of data items of a distributed database systems to be

www.manaraa.com

part of a tr-consistent global checkpoint. This condition is useful for constructing tr-consistent global checkpoints from a set of local checkpoints of a set of data items incrementally. This can also shed light on developing non-intrusive checkpointing algorithms for distributed database systems.

A Non-intrusive checkpointing algorithm for Distributed Database Systems

Based on the intuition gained from the development of the necessary and sufficient conditions, we developed a non-intrusive checkpointing protocol for distributed database systems. It uses the concept of logical checkpoints to reduce the checkpointing overhead as well as recovery overhead. The performance of the protocol has been evaluated through mathematical modeling as well as simulation.

A Model-based Communication-induced Checkpointing Protocol for Distributed Systems

We introduced the concept of Extended Suspect Z-cycles around a checkpoint of a process involved in a distributed computation and proved that eliminating Extended Suspect Z-cycles eliminates Z-cycles. Extended Suspect Z-cycles are on-line trackable whereas Z-cycles are not on-line trackable. We designed data structures that each process involved in a distributed computation need to maintain to track Extended Suspect Z-cycles on-line and developed a communication-induced checkpointing algorithm which tracks Extended Suspect Z-cycles and prevents them. The protocol has been evaluated through simulation.

5.2 Future Work

We made some fundamental contributions to the areas of distributed database systems as well as distributed message passing systems. However, this is the first step in

المنارة للاستشارات
www.manaraa.com

developing useful algorithms and models that can be applied to real-world problems. In the future, we will continue our research in this direction and develop better model-based communication-induced checkpointing algorithms. We will also develop recovery algorithms based on the developed checkpointing algorithms.

A More Efficient Model-based Communication-induced Checkpointing Algorithm Preventing Fully-informed Suspect Z-cycles ($FSZC$) for Distributed Messaging Systems

We already have some initial results in the direction of developing a better communication-induced checkpointing algorithm. We found that there is still room to improve the model-based communication-induced algorithm we presented in Chapter 4 for distributed computation. We found out that Extended Suspect Z-cycle ($ESZC$) is just a special case of a set of checkpoint and communication patterns. We propose to define this general set of patterns as Fully-informed Suspect Z-cycles. We will extend the concept of Extended Suspect Z-cycles to Fully-informed Suspect Z-cycles by introducing the concept of c-causal Z-path. The c-causal Z-path is a subset of the causal Z-path where in the sequence of the causal messages $[m_1, m_2, ..., m_q]$, there exists a checkpoint after the receiving event of m_1 and before the sending event of the next message m_2. We use $[m_1, C, m_2, ..., m_q]$ to express such a sequence of causal messages. Here is our formal definition of $FSZC$ based on c-causal Z-path.

Definition 16. *An* **Fully-informed SZC** *(denoted as $FSZC$) is a Checkpoint and Communication Pattern $FSZC(I_{j,z}, C_{i,x}, \mu, I_{k,y})$ such that:* $\exists m, m' : C_{j,z} \circ m \circ C_{i,x} \circ$

المنارة للاستشارات
www.manaraa.com

$\mu \overset{k,y}{\bullet} m'$ *and* $receive(m') \in I_{l,w}$ *with*

$$\begin{cases} i & send(m) \in I_{j,z} \\ ii & \mu \text{ is prime in } M(C_{i,x}, P_k) \\ iii & \nexists e \in I_{j,z+1} : e \overset{hb}{\rightarrow} receive(\mu.last) \\ iv & \mu \bullet m' \text{ is not strongly visibly doubled} \\ v & \nexists \text{ a c-causal chain } \mu^* \text{ in } M(C_{j,z}, P_k) \text{ such that } receive(\mu^*.last) \overset{hb}{\rightarrow} receive(\mu.last) \\ vi & \nexists \text{ a c-causal chain } \mu^{**} : [m_1, C, m_2, ..., m_k, ..., m_q] \text{ in } M(C_{i,x-1}, P_k) \\ & \text{such that } receive(\mu^{**}.last) \overset{hb}{\rightarrow} receive(\mu.last), \\ & receive(m_k) \in I_{l,u}, u \leq w \text{ and } receive(m_k) \overset{hb}{\rightarrow} receive(m') \end{cases} \tag{5.1}$$

As we can see, we add two more conditions, i.e., (v) and (vi), to the definition of $ESZC$ to form $FSZC$. Also condition (iii) is actually a special case of condition (v) if we visualize a message sent and received in the same checkpoint interval between $C_{j,z}$ and $C_{j,z+1}$ exists. Similarly condition (iv) is a special case of condition (vi) if we imagine a message sent and received in the same checkpoint interval between $C_{i,x-1}$ and $C_{i,x}$ exists. However we can not combine conditions (iii) and (v) into one since no such message sent and received in the same checkpoint interval could exist and we have to track them by different means. So are conditions (iv) and (vi). The above definition shows the relation between $ESZC$ and $FSZC$: a $FSZC$ is also a $ESZC$ but the converse is not necessarily true. Together with the result from Chapter 4 Section 4.3, we know the relation among all model-based communication algorithms proposed in current literature [34, 45, 52, 51, 53] plus our future algorithm can be presented as in Figure 5.1. In the figure, each circle represents all the communication patterns that satisfy the conditions such as SZC, $ESZC$, $FSZC$ etc. that different algorithms intent to track. Apparently, the closer is the circle to the Z-cycle circle, the less forced checkpoints will be taken in the system, since optimally we only want

المنارة للاستشارات
www.manaraa.com

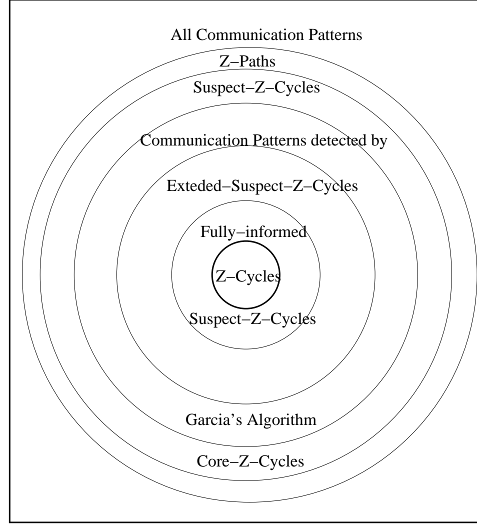


Figure 5.1: Relation of all algorithms including our future work

to prevent the real Z-cycles from happening even though it is not possible. Since $FSZC$ is the closest to Z-cycle, algorithm preventing $FSZC$ is likely to incur the least number of forced checkpoints.

Tracking $FSZC$ can help reduce even more forced checkpoints. On the other hand, preventing $FSZC$ guarantees the checkpointing algorithm not producing any useless checkpoint, which we will prove formally by using mathematical induction.

Proposing More Efficient Model-based Communication-induced Checkpointing Algorithms for Distributed Systems

We will develop more efficient model-based communication-induced checkpointing algorithms. We probably can discover some new model that is capable of preventing Z-cycles better. These new models may reduce the forced checkpoints even more. However, from the experience of discovering $FSZC$ and designing algorithm for tracking it we learn that we have to design complicated data structures as well as algorithms to accomplish the task. For example, from SZC to $FSZC$, the complexity of data structure has increased from a $O(n^2)$ integer array to a $O(n^3)$ integer array. Therefore while designing new model-based communication-induced checkpointing algorithms,

المنارة للاستشارات
www.manaraa.com

we have to balance between the complex data structures used and the reduction in forced checkpoints.

Rollback Recovery Algorithms for Model-based Communication-induced Checkpointing Algorithms

Our proposed model-based communication-induced checkpointing algorithms focused on the checkpointing processes where they try to produce only useful checkpoints and reduce the number of forced checkpoints as much as they can. In fault tolerance, how to roll back to a consistent global checkpoint is another important issue besides how to take useful checkpoints. In the index-based communication-induced algorithms, the rollback process is simple. In model-based communication-induced algorithms since there is no index involved, we have to use other heuristics such as the vector clock to discover recovery lines. There is some work in the literature discussing the recovery methods for model-based communication-induced algorithms such as constructing the Rollback Dependency Graph [25] and performing reachability analysis to calculate the recovery line. We need to study them thoroughly and adapt the appropriate approaches to all our newly proposed algorithms.

Comparing Our New Model-based Communication-induced Checkpointing Algorithm with Index-based Communication-induced Checkpointing Algorithms

We have understood the relation among all the algorithms in the domain of model-based communication-induced checkpointing algorithms (see Picture 5.1). We still want to explore the relation between model-based communication-induced checkpointing algorithms and index-based communication-induced checkpointing algorithms, and compare the performance in terms of the communication overhead and the number of forced checkpoints. Our expectation is that model-based communication-

المنارة للاستشارات
www.manaraa.com

induced checkpointing algorithms would produce less forced checkpoints since it has better picture of the checkpoint and communication patterns in the system but it may incur more communication overhead. It is a tradeoff and we want to evaluate quantitatively the relation between communication overhead and forced checkpoints. This will help us to develop better communication-induced algorithms that balance both factors. It also helps us to categorize all the current communication-induced algorithms so that we can use the most efficient algorithm under different circumstances such as some applications require minimum forced checkpoints while others may prefer minimum communication overhead.

Designing Algorithm to Identify Zigzag Paths in the Global Serialization Graphs

We have proved that a set $\mathbf{S}'$ of checkpoints, each checkpoint of which is from a different data item, can belong to the same tr-consistent global checkpoint with respect to a serializable schedule of a set of transactions iff no checkpoint in $\mathbf{S}'$ has a zigzag path to any checkpoint (including itself) in $\mathbf{S}'$ in the global serialization graph corresponding to that schedule. The next question is how to identify zigzag paths in the global serialization graph. Therefore we need to design the algorithm to identify the zigzag paths in the global serialization graph based on our Definition 14 of zigzag paths. Such an algorithm will help us identify useless checkpoints and may also throw light on new non-intrusive algorithms for distributed database systems.

المنارة للاستشارات
www.manaraa.com

Bibliography

[1] A. Silberschatz, H. F. Korth and S. Sudarshan, *Database System Concepts*, 1996.

[2] M. Singhal and N. G. Shivaratri, *Advanced Concepts in Operating Systems*, 1994.

[3] B. Randell, Reliable computing systems, *Operating Systems: an Advanced Course*, SpringerVerlag, New York, 1979.

[4] R Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, Addison-Wesley, 2007.

[5] R. Baldoni, F. Quaglia and M. Raynal, Consistent checkpointing for transaction systems, *The Computer Journal*, **44**(2), 92-100, 2001.

[6] K. M. Chandy and L. Lamport, Distributed snapshot: determining global states of distributed systems *ACM Transactions on Computer Systems*, 63-75, Feburary 1985.

[7] R. H. B. Netzer and J. Xu, Necessary and sufficient conditions for consistent global snapshots, *IEEE Transactions on Parallel and Distributed Systems*, **6**(2), 165-169, Feburary 1995.

[8] A. P. Sheth and J. E. Larson, Federated database systems for managing distributed heterogeneous, and autonomous databases, *ACM Computing Surveys*, **22**(3), 183-286, September 1990.

[9] Vijay Kumar and Shawn D. Moe, Performance of recovery algorithms for centralized database management systems, *Information Sciences*, **86**(1), 101-147, September 1995.

[10] R. B. Hagmann, A crash recovery scheme for a memory-resident database system, *IEEE Transactions on Computers*, **35**(9), 839-843, 1986.

[11] S. Pilarski and T. Kameda, Checkpointing for distributed databases: starting from the basics, *IEEE Transactions on Parallel and Distributed Systems*, **3**, 602-610, 1992.

[12] H. Garcia-Molina and K. Salem, *IEEE Transactions on Knowledge and Data Engineering*, **4**(6), 509-516, December 1992.

المنارة للاستشارات
www.manaraa.com

[13] J. Lin and M. H. Dunham, A survey of distributed database checkpointing, *Distributed Parallel Databases*, **5**, 289-319, 1997.

[14] G. Ferran, Distributed checkpointing in a distributed data management system, *Proc. Real-Time Systems Symposium*, Miami Beach, Florida, 43-49, 1981.

[15] H. Kuss, On totally ordering checkpoints in distributed databases, *Proceedings of the ACM International Conference on Management of Data*, 1982.

[16] R. D. Schlichting and F. B. Schneider, Fail-stop processors: An approach to designing fault-tolerant computing systems, *ACM Trans. Computing Systems*, **1**(3), 222-238.

[17] J. L. Zhao and A.Segev and A. Chatterjee, A universal relation approach to federated database management, *Proceedings of Eleventh International Conference on Data Engineering*, 1995.

[18] J. L Zhao, Schema coordination in federated database management: a comparison with schema integration, *Decision Support Systems*, **20**(3), 243-257, July 1997.

[19] J. Muilu and L. Peltonen and J.-E. Litton, The federated database a basis for biobank-based post-genome studies, integrating phenome and genome data from 600000 twin pairs in Europe, *European Journal of Human Genetics*, **15**, 718-723, May 2007.

[20] J. Kim and G. Fox, A hybrid keyword search across peer-to-peer federated databases, *Proceedings of International Conference on Advances in Databases and Information Systems*, 2004.

[21] J. Kleewein, Practical issues with commercial use of federated databases, *Proceedings of the 22th International Conference on Very Large Data Bases*, 1996.

[22] A. Deshpande and J.M. Hellerstein, Decoupled query optimization for federated database systems, *Proceedings of 18th International Conference on Data Engineering*, 2002.

[23] K. Salem and H. Garcia-Molina, Checkpointing memory-resident databases, *Proceedings of the Fifth International Conference on Data Engineering*, 452-462, 1989.

[24] A.-P. Liedes and A. Wolski, SIREN: a memory-conserving, snapshot-consistent checkpoint algorithm for in-memory databases, *Proceedings of the 22nd International Conference on Data Engineering*, April 2006.

[25] L. Alvisi and K. Marzullo, Trade-offs in implementing causal message logging protocols, *Proceedings of the 1996 ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing Systems (PODC)*, 58-67, 1996.

المنارة للاستشارات
www.manaraa.com

[26] S. H. Son and A. K. Agrawala, Distributed checkpointing for globally consistent states of databases, *IEEE Transactions on Software Engineering*, **15**(10), 1157-1167, October 1989.

[27] C. Pu, On-the-fly, incremental, consistent reading of entire databases, *Proceedings of the 11th Conference on Very Large Database*, Morgan Kaufman Pubs. (Los Altos, CA), Stockholm, 367–375, 1985.

[28] S. Pilarski and T. Kameda, A novel checkpointing scheme for distributed database systems, *Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, Nashville, TN, 1990.

[29] D. Manivannan and M. Singhal, Quasi-synchronous checkpointing: models, characterization, and classification, *IEEE Transactions on Parallel and Distributed Systems*, **10**(7), 703-713, July 1999.

[30] S. H. Son, An algorithm for non-interfering checkpoints and its practicality in distributed database systems, *Information Systems*, **14**(5), 421-429, 1989.

[31] K. M. Chandy and L. Lamport, Distributed snapshots: determining global states of distributed systems, *ACM Transactions on Computer Systems*, **3**(1), 63-75, 1985.

[32] L. Lamport, Time, clocks and the ordering of events in a distributed system, *Communications of the ACM*, **21**(7), 558-565, 1978.

[33] D. Manivannan and M. Singhal, A low-overhead recovery technique using quasi synchronous checkpointing, *Proceedings of IEEE International Conference on Distributed Computing Systems*, 100-107, 1996.

[34] R. Baldoni, F. Quaglia and B. Ciciani, A VP-accordant checkpointing protocol preventing useless checkpoints, *Proceedings of Seventeenth IEEE Symposium on Reliable Distributed Systems*, **10**(7), 61-67, 1998.

[35] R. Baldoni, J. M. Helary and M. Raynal, Rollback-dependency trackability: a minimal characterization and its protocol, *Information and Computation*, (165), 144-173, 2001.

[36] R. Baldoni, J. M. Helary and A. Mostefaoui and M. Raynal, A communication-induced checkpointing protocol that ensures rollback-dependency trackability, *Proceedings of IEEE International Symposium on Fault Tolerant Computing*, 68-77, 1997.

[37] ”I. C. Garcia and L. E. Buzato, An efficient checkpointing protocol for the minimal characterization of operational rollback-dependency trackability, *Proceedings of the 23th IEEE International Symposium on Reliable Distributed Systems*, **18**(20), 126-135, 2004.

[38] J. M. Helary, A. Mostefaoui and R. H. B. Netzer and M. Raynal, Communication-based prevention of useless checkpoints in distributed computations, *Distributed Computing*, (13), 29-43, 2000.

المنارة للاستشارات
www.manaraa.com

[39] J. Tsai, An efficient index-based checkpointing protocol with constant-size control information on messages, *IEEE Transactions on Dependable and Secure Computing*, **2**(4), 287-296, 2005.

[40] J. Tsai, S. Y. Kuo and Y. M. Wang, Theoretical analysis for communication-induced checkpointing protocols with rollback-dependency trackability, *IEEE Transactions on Parallel and Distributed Systems*, **9**(10), 963-971, 1998.

[41] J. Tsai, On properties of RDT communication-induced checkpointing protocols, *IEEE Transactions on Parallel and Distributed Systems*, **14**(8), 755-764, 2003.

[42] J. Tsai, S. Y. Kou and Y. M. Wang, More properties of communication-induced checkpointing protocols with rollback-dependency trackability, *Journal of Information Science and Engineering*, **21**, 239-257, 2005.

[43] R. Baldoni, F. Quaglia and P. Fornara, An index-based checkpointing algorithm for autonomous distributed systems, *IEEE Transactions on Parallel and Distributed Systems*, **10**(2), 181-192, 1999.

[44] Y. M. Wang, Consistent global checkpoints that contains a given set of local checkpoints, *IEEE Transactions on Computers*, **46**(4), 456-468, 1997.

[45] I. C. Garcia and L. E. Buzato, Checkpointing using local knowledge about recovery lines, University of Campinas, Brazil, TR-IC-99-22, 1999.

[46] R. Koo and S. Toueg, Checkpointing and rollback-recovery for distributed systems, *IEEE Trans. Software Eng.*, **13**(1), 23-31, 1987.

[47] J. Tsai, S. Y. Kuo and Y. M. Wang, Evaluations on domino-free communication-induced checkpointing protocols, *Information Processing Letters*, **69**(1), 31-37, 1999.

[48] M. Singhal and N. G. Shivaratri, *Advanced Concepts in Operating Systems*, McGraw-Hill, 105, 1994.

[49] J. Fidge, Timestamps in message-passing systems that preserve the partial ordering, *Proceedings of the 11th Australian Computer Science Conference*, **10**(1), 56-66, 1988.

[50] F. Mattern, Virtual time and global states of distributed systems, *Parallel and Distributed Algorithms*, Elsevier Science, North-Holland, 215-226, 1989.

[51] J. Wu, Y. Luo and D. Manivannan, An enhanced model-based communication-induced checkpointing protocol for distributed systems, *Proceedings of the 25th Internation Conference on Parallel and Distributed Computing and Networking*, 2007.

[52] F. Quaglia, R. Baldoni and B. Ciciani, *Computational Fluid Dynamics*, Hermosa, Albuquerque, NM, 1976.

المنارة للاستشارات
www.manaraa.com

[53] J. Wu and D. Manivannan, An enhanced model-based checkpointing protocol for preventing useless checkpoints, *International Journal of Parallel, Emergent and Distributed Systems*, **24**(5), 383-406, 2009.

[54] N. H. Vaidya, Staggered consistent checkpointing, *IEEE Transactions on Parallel and Distributed Systems*, **10**(7), 694-702, July 1999.

[55] R. E. Strom and S. A. Yemini, *Optimistic Recovery: an Asynchronous Approach to Fault-Tolerance in Distributed Systems*, 1984.

[56] Y. M. Wang and W. K. Fuchs, Lazy checkpoint coordination for bounding rollback propagation *In Proc. 12th IEEE Int. Symp. on Reliable Distributed Systems*, IEEE Computer Society Press, 78-85, 1993.

[57] J. Wu, D. Manivannan and B. Thuraisingham, Necessary and sufficient conditions for transaction-consistent global checkpoints in a distributed database system, *Information Sciences*, Elsevier, **179**(20), 3659-3672, September 2009.

[58] J. Wu and D. Manivannan, An efficient non-intrusive checkpointing algorithm for distributed database systems, *Springer Lecture Notes in Computer Science Series*, **4308**, 82-87, 2006.

[59] J. Wu, D. Manivannan and B. Thuraisingham, Transaction-consistent global checkpoints in a distributed database system, *Proceedings of the 2008 International Conference on Data Mining and Knowledge Engineering* , 2008.

المنارة للاستشارات
www.manaraa.com

Vita

Personal Data:

Name: Jiang Wu

Date of Birth: 10/24/1975

Place of Birth: Xi'an, Shaanxi, China

Educational Background:

- Master of Science in Computer Science, Bowling Green State University, 2002.
- Bachelor of Engineering in Communication Engineering, Xidian University, China, 1998.

Professional Experience:

- Data Warehouse Analyst, 04/2007 - present.
  Information System, Marshfield Clinic, Marshfield, WI, USA.

Publications:

- Jiang Wu and D. Manivannan, An enhanced model-based checkpointing protocol for preventing useless checkpoints, *International Journal of Parallel, Emergent and Distributed Systems*, **24**(5):383-406, 2009.
- Jiang Wu, Yi Luo and D. Manivannan, An enhanced model-based communication-induced checkpointing protocol for distributed systems, in *Proceedings of the 25th Internation Conference on Parallel and Distributed Computing and Networking*, 2007.
- Jiang Wu, D. Manivannan and Bhavani Thuraisingham, Necessary and sufficient conditions for transaction-consistent global checkpoints in a distributed

المنارة للاستشارات
www.manaraa.com

database system, *Information Sciences*, Elsevier, **179**(20):3659-3672, September 2009.

- Jiang Wu, D. Manivannan and Bhavani Thuraisingham, Transaction-consistent global checkpoints in a distributed database system, in *Proceedings of the 2008 International Conference on Data Mining and Knowledge Engineering (ICDMKE 2008)*, London, U.K., July 2-4, 2008.

- Jiang Wu and D. Manivannan, An efficient non-intrusive checkpointing algorithm for distributed database systems, *Springer Lecture Notes in Computer Science Series*, No.4308 pp:82-87.

المنارة للاستشارات
www.manaraa.com