Information Sciences 179 (2009) 3659–3672

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Necessary and sufficient conditions for transaction-consistent global checkpoints in a distributed database system

Jiang Wu [a], D. Manivannan [a,*], Bhavani Thuraisingham [b]

[a] Department of Computer Science, University of Kentucky, Lexington, KY 40506, United States
[b] Department of Computer Science, University of Texas at Dallas, United States

ARTICLE INFO

Article history:
Received 14 November 2007
Received in revised form 9 June 2009
Accepted 14 June 2009

Keywords:
Checkpointing
Recovery
Distributed databases

ABSTRACT

Checkpointing and rollback recovery are well-known techniques for handling failures in distributed systems. The issues related to the design and implementation of efficient checkpointing and recovery techniques for distributed systems have been thoroughly understood. For example, the necessary and sufficient conditions for a set of checkpoints to be part of a consistent global checkpoint has been established for distributed computations. In this paper, we address the analogous question for distributed database systems. In distributed database systems, transaction-consistent global checkpoints are useful not only for recovery from failure but also for audit purposes. If each data item of a distributed database is checkpointed independently by a separate transaction, none of the checkpoints taken may be part of any transaction-consistent global checkpoint. However, allowing individual data items to be checkpointed independently results in non-intrusive checkpointing. In this paper, 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 distributed database. Such conditions can also help in the design and implementation of non-intrusive checkpointing algorithms for distributed database systems.

1. Introduction

It is a common practice to take checkpoint of a database from time to time, and restore the database to the most recent checkpoint when a failure occurs. It is desirable that a global checkpoint of a database records a state of the database which reflects the effect of a set of completed transactions and not the results of any partially executed transactions. Such a checkpoint of the database is called a transaction-consistent global checkpoint [23]. 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 the checkpoint. Such a checkpoint is guaranteed to be transaction-consistent, but this approach is not practical, since blocking newly-submitted transactions will increase transaction response time which may not be acceptable for the users of the database. Another approach would be to run a read only transaction which would read the entire database and save it to stable storage; the underlying concurrency control algorithm will ensure that the saved state is transaction-consistent. This would be inefficient especially in the presence of long-living transactions. A more efficient way would be to save (checkpoint) the state of each data item independently and periodically without blocking any transaction. However if each data item is checkpointed independently and periodically, some checkpoints of some data items may not be part of any transaction-consistent global checkpoint of the database and hence are useless.

* Corresponding author. Tel.: +1 859 257 9234; fax: +1 859 323 3740.
E-mail addresses: jwu6@cs.uky.edu (J. Wu), mani@cs.uky.edu, manivann@cs.uky.edu (D. Manivannan), bhavani.thuraisingham@utdallas.edu (B. Thuraisingham).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.06.016

In this paper, we address this issue and establish 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 a transaction-consistent global checkpoint incrementally from the checkpoints of each individual data item. By applying this condition, we can start from an useful checkpoint of any data item and then incrementally add checkpoints of other data items until we get a transaction-consistent checkpoint of the database.

1.1. Motivation and objectives

In a distributed system, to minimize the lost computation due to failures, the state of the processes involved in a distributed computation are periodically saved (checkpointed). When one or more processes involved in the distributed computation fails, the processes are restarted from a previously saved consistent global checkpoint. When processes are independently checkpointed, the checkpoints taken may not be part of any consistent global checkpoint and hence are useless [21]. Netzer and Xu [21] 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. Several checkpointing algorithms have been proposed for distributed systems [2,18,9,8,19,3,17].

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. Some of these are discussed in Section 2 and Section 3. Non-intrusive checkpointing algorithms 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. Motivated by the work of Netzer and Xu for distributed computations [21], in this paper, we 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.

1.2. Organization of the paper

The remainder of this paper is organized as follows. In Section 2 we introduce the background required for understanding the paper. Section 3 discusses related works. In Section 4 we present 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 prove its correctness; we also discuss the applications of our work. Section 5 concludes the paper.

2. Background

2.1. System model

We consider a model of distributed database system similar to the model in [23]. In this model, a *distributed database system* consists of a set of data items residing at various sites. Sites can exchange information via messages transmitted on a communication network, which is assumed to be reliable. The data items of the database are accessed by transactions and the transactions are controlled by transaction managers (TM) that reside at these sites. The TM is responsible for the proper scheduling of transactions using appropriate concurrency control algorithms in such a way that the integrity of the database is maintained. In addition, the data items at each site are controlled by a data manager (DM). Each DM is responsible for controlling access to data items at its site. 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 the data item so 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. In order to guarantee the integrity and efficiency of transaction processing, the following four properties, referred to as **ACID** [27], must be maintained.

- **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 execute 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 maximum performance, a proper schedule of transactions need to be arranged in which the operations of various transactions are interleaved as much as possible. Given a schedule, a directed graph, referred to as precedence graph [28] or serialization graph [27], can be constructed to illustrate the procedure of

all the transactions running in the database system. The serialization graph serves as an important tool to analyze transaction processing in the distributed database systems.

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 $\mathbf{X} = \{x_i | 1 \leqslant i \leqslant n\}$. 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 $\mathbf{C_i} = \{C_i^{k_i} | k_i : k_i \geqslant 0\}$. 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 obtained after termination of all transactions accessing data item x_i. To minimize checkpointing overhead, a data item is checkpointed only after the state of the data item changes. That is, after a data item is checkpointed, it is not checkpointed again until at least one other transaction has accessed and changed the data item.

Let $\mathbf{T} = \{T_i | 1 \leqslant i \leqslant m\}$ be a set of transactions that access the database system. 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 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. In order to maintain atomicity of transactions, $T_{C_i^{k_i}}$ is the local transaction which is required to be scheduled to access a data item 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 in the system is denoted by $\mathbf{T_C}$.

A global checkpoint of the database is a set $\mathbf{S} = \{C_i^{k_i} | 1 \leqslant i \leqslant n\}$ of local checkpoints consisting of one checkpoint for each data item. The set of checkpointing transactions that produce the global checkpoint $\mathbf{S}$ is denoted by $\mathbf{S_T} = \{T_{C_i^{k_i}} | 1 \leqslant i \leqslant n\}$. 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 which takes that checkpoint.

Each regular transaction is a partially ordered set of read and/or write operations (operations are partially ordered because two adjacent read operations in a transaction are not comparable). 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 (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) [23].

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 $\mathbf{T} \bigcup \mathbf{T_C}$, we define the relation $<_T$ between transactions in $\mathbf{T} \bigcup \mathbf{T_C}$ with respect to the schedule ε as follows:

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

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

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

A schedule is serial if the operations belonging to each transaction appear together in the schedule [27]. A schedule ε is serializable if the schedule has the effect equivalent to a schedule produced when the transactions are run serially in some order. The concurrency control algorithm ensures that a schedule of transactions running in the distributed database system is serializable. One important kind of serialization, called conflict serializability (CSR) [27,5] is considered in this paper. An execution $\varepsilon \in CSR$ iff the relation $<_T$ is acyclic [5]. 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 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 $<_T$.

Formal definition of a *transaction consistent* global checkpoint follows [23]:

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.

If the concurrency control algorithm guarantees an execution $\varepsilon \in CSR$, then the relation $<_T$ induces a directed acyclic graph (Dag) on $\mathbf{T} \bigcup \mathbf{T_C}$ and conversely [5]. 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 component of the global serialization graph. The *local serialization graph* induced by the transactions in $\mathbf{T} \bigcup \mathbf{T_{C_i}}$ accessing data item x_i is denoted by $G_{x_i}(V_{x_i}, E_{x_i})$: the vertex set $V_{x_i} = \{T_k \cup T_{C_i^{k_i}} | T_k \in \mathbf{T}$ 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: $E_{x_i}^{TT} = \{(T_i, T_j) | T_i, T_j \in V_{x_i}; T_i <_T T_j\}$.

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

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

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 $\mathbf{T} = \{T_1, \ldots, T_9\}$ accessing a database containing five data items $X = \{x_1, \ldots, x_5\}$.

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. $T_9 : W_9(x_1), W_9(x_5)$.

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 in fact 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 Figs. 1–3 The global serialization graph constructed from the local graphs is shown in Fig. 4. The global serialization graph $G = (V, E)$ is obtained by merging the local serialization graphs where

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

and

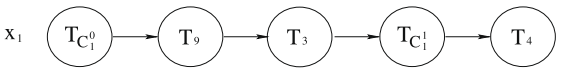

Fig. 1. Example of local serialization graph on x_1, on which a set of transactions $\{T_9, T_3, T_4\}$ have finished execution.

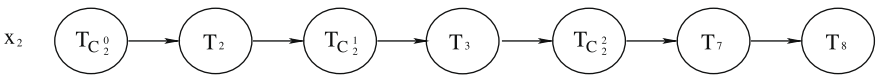

Fig. 2. Example of local serialization graph on x_2, which has been accessed by the set of transactions $\{T_2, T_3, T_7, T_8\}$.

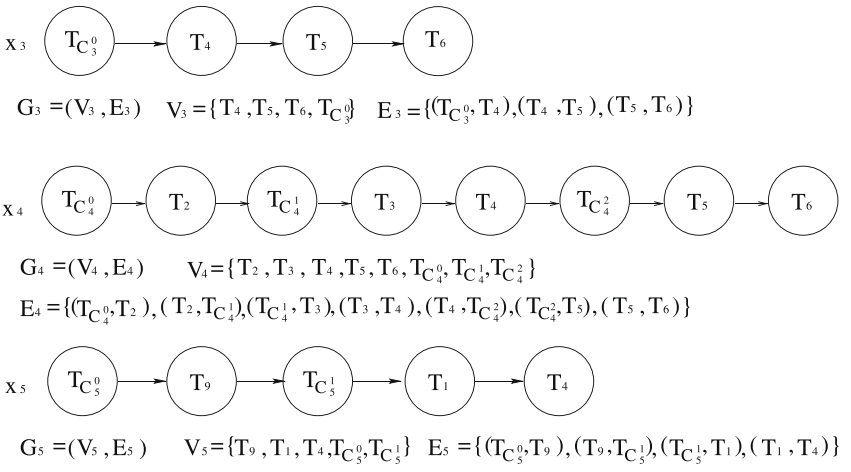

Fig. 3. Local serialization graphs for the rest of the data items: x_3, x_4, and x_5.

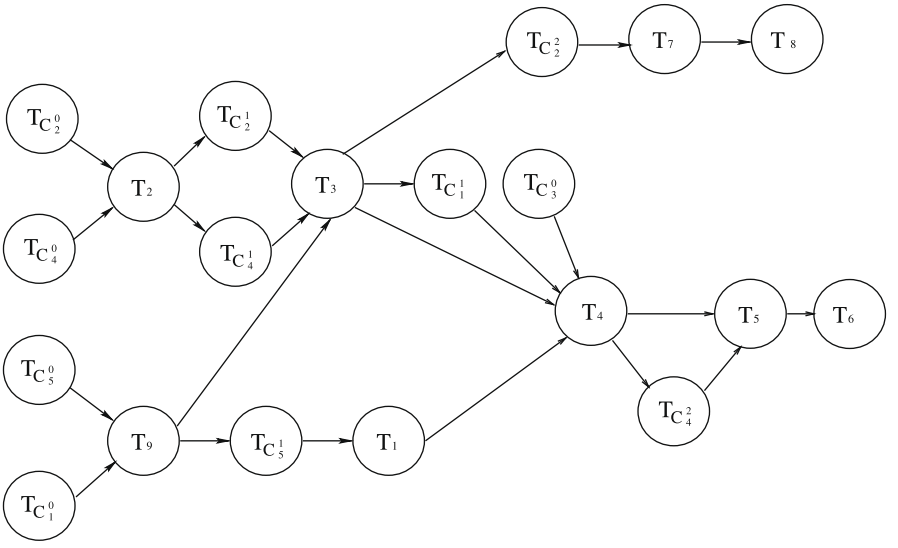

Fig. 4. Global serialization graph constructed from local serialization graphs on x_1, x_2, x_3, x_4, and x_5.

$$E = \{(T_{C_1^0}, T_9), (T_{C_2^0}, T_2), (T_{C_3^0}, T_4), (T_{C_4^0}, T_2), (T_{C_5^0}, T_9), (T_9, T_3), (T_3, T_{C_1^1}), (T_{C_1^1}, T_4), (T_2, T_{C_2^1}), (T_{C_2^1}, T_3), (T_3, T_{C_2^2}), (T_{C_2^2}, T_7),$$
$$(T_7, T_8), (T_2, T_{C_4^1}), (T_{C_4^1}, T_3), (T_3, T_{C_4^2}), (T_{C_4^2}, T_4), (T_4, T_5), (T_{C_5^0}, T_9), (T_9, T_{C_5^1}), (T_{C_5^1}, T_1), (T_1, T_4), (T_5, T_6)\}.$$

The graph in Fig. 4 is acyclic and hence the schedule is $\in$ *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 throughout the paper: $T_i \rightarrow^+ T_j$ 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). $T_i \rightarrow T_j$ 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 **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 any transaction in σ_2 has started execution. In particular, if $\sigma_1 \bigcup \sigma_2 = \mathbf{T}$, then the set of checkpoints **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 exists two transactions T_i and T_j that access x_i and $T_i \rightarrow^+ T_{C_i^{k_i}}, T_{C_i^{k_i}} \rightarrow^+ T_j$, then in the *local serialization graph* induced by the operations in $\mathbf{T} \cup \mathbf{T_C}$ 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 the operations in $\mathbf{T} \cup \mathbf{T_C}$ 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 $T_{C_i^{k_i}} \rightarrow^+ T_j$ and there exists $T_i \rightarrow^+ T_j$ without any checkpoint along the path in the local serialization graph, then $T_{C_i^{k_i}} \rightarrow^+ T_i$.
2. Similarly, if $T_i \rightarrow^+ T_{C_i^{k_i}}$ and there exists a path $T_i \rightarrow^+ T_j$ from T_i to T_j without any checkpoint along the path in the local serialization graph, then $T_j \rightarrow^+ T_{C_i^{k_i}}$.

Observations 1 and 2 are trivial. Observation 3 holds because in case 1, suppose $T_{C_i^{k_i}} \rightarrow^+ T_i$ is not true, then $T_i \rightarrow^+ T_{C_i^{k_i}}$ from Observation 1. Since $T_{C_i^{k_i}} \rightarrow^+ T_j$, 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 $T_i \rightarrow^+ T_j$ without any checkpointing transactions along the path. A similar argument can be used to prove the correctness of case 2 in Observation 3.

We make use of these Observations in the proof of the two important theorems in Section 4. Notice that the transactions T_i, T_j in the previous observations could be checkpointing transactions themselves.

3. Related work

The checkpointing algorithms for distributed database systems can be classified as log-oriented and dump-oriented [16]. In the log-oriented approach, periodically a dump of the database is taken 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 is applied to the dump until the 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 tr-consistent global checkpoint. Algorithms belonging to this class include [6] and [14]. 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 tr-consistent checkpoint of the database. The algorithms proposed in [29,24,22] take this approach. The basic idea behind the algorithm in [29] is to divide the transactions into two groups: those before or after the checkpointing process. This algorithm is non-intrusive but requires a copy of the database stored temporarily. This temporary copy is accessed by transactions that cannot be decided which group they belong to while the checkpointing is in progress. Pu [24] 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. [22] 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. The previous two algorithms are coordinated algorithms, in which one process initiates and coordinates the checkpointing activity. The algorithm proposed by Baldoni et al. [4] uses a non-coordinated approach, in which no process initiates checkpointing and each data item is checkpointed independently. Like the algorithm of [22], checkpoint numbers are used to synchronize the checkpointing process and forced checkpoints are taken to prevent useless checkpoints. This algorithm is fully distributed but may incur a large number of forced checkpoints, depending on the execution pattern of the transactions.

Pilarski et al. [23] 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 in a distributed system and tr-consistent checkpoints in a distributed database system. Moreover, they establish sufficient conditions for a set of checkpoints to be part of a tr-consistent global checkpoint. However, they do not establish necessary and sufficient conditions for a set of checkpoints to be part of a tr-consistent global checkpoint. Our goal in this paper is to establish the necessary and sufficient conditions for the checkpoints of a set of data items to be part of a tr-consistent global checkpoint. Kumar and Moe [13] present a performance evaluation of some existing recovery algorithms for databases.

Recently, many researchers have focussed on fuzzy checkpointing algorithms [10,25,15] that write dirty pages to disk and require transaction logs for reconstructing a tr-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 [7]. Fuzzy checkpointing does not obstruct the transaction processing but requires an undo/redo log to bring the inconsistent checkpoint back to a consistent state. In the next section, we present the necessary and sufficient conditions for the checkpoints of a set of data items to be part of a tr-consistent global checkpoint. Still, regular checkpointing approach appears to be suitable for database systems in which the entire database need not be loaded into memory, and hence we focus on such databases only.

4. Necessary and sufficient condition

In distributed database systems, it would be ideal if individual data items could be checkpointed without any coordination and a tr-consistent checkpoint could be constructed from the checkpoints of the individual data items whenever it is

needed for recovery. For this, we need to know what checkpoints could be combined to construct a tr-consistent global checkpoint. The following theorem establishes 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 set of transactions.

Theorem 1. *Let* $\mathbf{T} = \{T_1, \ldots, T_m\}$ *be a set of transactions accessing the database consisting of n data items* $\mathbf{X} = \{x_1, \ldots, x_n\}$. *Assume that each data item is checkpointed by a checkpointing transaction that runs at the site containing the data item. Let* $\mathbf{S} = \{C_i^{k_i} | 1 \leqslant i \leqslant n\}$ *be a set of checkpoints, one for each data item and let* $\mathbf{S_T} = \{T_{C^{k_i}} | 1 \leqslant i \leqslant n\}$ *be the set of checkpointing transactions that produce* **S**. *Let* ε *be a schedule over* **T**. *Then* **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 in the global serialization graph between any two checkpointing transactions in $\mathbf{S_T}$. Then we prove that the set **S** forms a tr-consistent global checkpoint. It is sufficient to prove that there exist a serialization order $\sigma_1\sigma_2$ of 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 has started execution.

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

(1) $\sigma_a = \{T_i \in \mathbf{T} | T_i \rightarrow^+ \mathbf{S_T}\}$,
(2) $\sigma_b = \{T_i \in \mathbf{T} | \mathbf{S_T} \rightarrow^+ T_i\}$,
(3) $\sigma_c = \{T_i \in \mathbf{T} |$ neither $T_i \rightarrow^+ \mathbf{S_T}$ nor $\mathbf{S_T} \rightarrow^+ T_i\}$.

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 $T_v \rightarrow^+ \mathbf{S_T}$ by definition. In particular $T_v \rightarrow^+ T_{C_i^{k_i}}$ 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.
T_v must have finished accessing every data item x_j before $T_{C_j^{k_j}} \in \mathbf{S_T}$ starts execution.

Proof of claim. The following three cases arise:

(1) $T_v \rightarrow^+ T_{C_j^{k_j}}$. This means T_v has finished accessing x_j before $T_{C_j^{k_j}}$ takes checkpoint on x_j.
(2) $T_{C_j^{k_j}} \rightarrow^+ T_v$. This case cannot arise since from $T_v \rightarrow^+ T_{C_i^{k_i}}$ we have $T_{C_j^{k_j}} \rightarrow^+ T_{C_i^{k_i}}$, a contradiction to the assumption that there is no path between any two checkpointing transactions in $\mathbf{S_T}$.
(3) Neither $T_v \rightarrow^+ T_{C_j^{k_j}}$ nor $T_{C_j^{k_j}} \rightarrow^+ T_v$. 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 has 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 $T_i \rightarrow^+ T_{C_v^{k_v}}$, and $T_{C_w^{k_w}} \rightarrow^+ T_i$. Hence $T_{C_w^{k_w}} \rightarrow^+ T_{C_v^{k_v}}$, 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 **S** is a tr-consistent global checkpoint, then we prove that no two elements 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 $T_{C_i^{k_i}} \rightarrow^+ T_{c_1} \rightarrow T_{C_j^{k_j}}$.

First we show that T_{c_1} starts execution after every checkpointing transaction in $\mathbf{S_T}$ has finished. Because of the path $T_{C^{k_i}} \rightarrow^+ T_{c_1}$, we know that T_{c_1} must start execution after $T_{C^{k_i}} \in \mathbf{S_T}$ has executed. 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 executed. On the other hand, on x_j, T_{c_1} has started execution before $T_{C_j^{k_j}} \in \mathbf{S_T}$ has started due to the edge $T_{c_1} \rightarrow T_{C_j^{k_j}}$, a contradiction.

Hence a global checkpoint **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. □

Theorem 1 is useful for verifying whether a given global checkpoint is tr-consistent. For instance, in Fig. 4, $\mathbf{S} = \{T_{C_1^0}, T_{C_2^1}, T_{C_2^0}, T_{C_4^1}, T_{C_5^0}\}$ forms a tr-consistent global checkpoint because no two elements in **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 Fig. 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 consistent global checkpoint because data item x_4 does not have a local 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 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 [21] 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 distributed computation to be part of a consistent global checkpoint. We generalize their definition of zigzag paths to checkpoints in distributed database systems and use it for establishing a necessary and sufficient condition for a set of checkpoints of a set of data items of a distributed database system to be part of a tr-consistent global checkpoint.

Definition 2. Let **T** be a set of transactions executing on a database. Let $C_i^{k_i}$ be a checkpoint taken by the checkpointing transaction $T_{C_i^{k_i}}$, and let $C_j^{k_j}$ be another checkpoint taken by checkpointing transaction $T_{C_j^{k_j}}$. We say a zigzag path with respect to **T** exists from $T_{C_i^{k_i}}$ to $T_{C_j^{k_j}}$ if there exists a set of transactions $\mathbf{T}' = \{T_{i_1}, T_{i_2}, \ldots, T_{i_v}\} \subseteq \mathbf{T}$ such that

(a) $T_{i_1} \in \mathbf{T}'$ is a transaction such that $T_{C_i^{k_i}} \rightarrow T_{i_1}$ in the global serialization graph;
(b) for any $T_{i_k} \in \mathbf{T}'(1 \leqslant k < v), T_{i_{k+1}} \in \mathbf{T}'(1 < (k+1) \leqslant v)$ is a transaction such that
  1: $T_{i_k} \leftarrow T_{i_{k+1}}$ (we call such an edge as reverse edge);
  or
  2: $T_{i_k} \rightarrow T_{i_{k+1}}$ or ($T_{i_k} \rightarrow T_{C_w^{k_w}}$ and $T_{C_w^{k_w}} \rightarrow^{+} T_{i_{k+1}}$ for some w);
(c) $T_{i_v} \in T'$ is a transaction such that $T_{i_v} \rightarrow T_{C_j^{k_j}}$;

For example, in the global serialization graph shown in Fig. 4,

1: A zigzag path exists from $T_{C_5^1}$ to $T_{C_4^2}$, the path being $T_{C_5^1} \rightarrow T_1 \rightarrow T_4 \rightarrow T_{C_4^2}$.
2: A zigzag path exists from $T_{C_5^1}$ to $T_{C_2^2}$, the path being $T_{C_5^1} \rightarrow T_1 \rightarrow T_4 \leftarrow T_3 \rightarrow T_{C_2^2}$.
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 a path in the global serialization graph is also a zigzag path but not conversely.

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 Fig. 4, include checkpoints $T_{C_5^1}$, the z-cycle being $T_{C_5^1} \rightarrow T_1 \rightarrow^{i} T_4 \leftarrow T_3 \rightarrow T_9 \rightarrow T_{C_5^1}$; and $T_{C_1^1}$, the z-cycle being $T_{C_1^1} \rightarrow T_4 \leftarrow T_3 \rightarrow T_{C_1^1}$. $T_{C_4^2}$ is also on a z-cycle. Next, we establish the necessary and sufficient condition formally.

Theorem 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 **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 **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 **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).

We claim that **S** is a tr-consistent global checkpoint. From Theorem 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 exists from A to B (note that a path in the graph is a zigzag path but not conversely), contradicting the assumption that no zigzag path exists between any two checkpoints in $\mathbf{S}'$.

Case 2: $A \in \mathbf{S} - \mathbf{S}'$ and $B \in \mathbf{S}\prime$. 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, $D \rightarrow^{+} B$, which also exists in the global serialization graph.Let T_u be the transaction (that accessed x_j and created the edge $T_u \rightarrow B$) 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 claim. Since $D \rightarrow^{+} B, T_u \rightarrow B$, and B is created by a checkpointing transaction that is also a transaction, we get $D \rightarrow^{+} T_u$ 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 $D \rightarrow^{+} T_u$ can be found in the global serialization graph. Then in the global serialization graph, the zigzag path from A to T_u, the reverse path $D \leftarrow^{+} T_u$, and the zigzag path from D to E constitutes a zigzag path from A to E, which is a contradiction to the assumption that no zigzag path exists between any two checkpoints in $\mathbf{S}'$.

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 where A is the first checkpoint on data item x_i with no zigzag path to any member of $\mathbf{S}'$.

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

(Only-if Part:) Conversely, suppose there exists a zigzag path between two checkpoints in $\mathbf{S}'$ (including zigzag cycle), 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 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 $T_{1,1} \leftarrow T_{2,1}, \ldots, T_{1,u} \leftarrow T_{2,u}$, as shown in Fig. 5a. Suppose those reverse edges are components of local serialization graph corresponding to data items $x_{1,1}, \ldots, x_{1,u}$ respectively. □

Claim.

$x_{1,1}, \ldots, x_{1,u}$ *cannot all be equal to* x_i*, where A takes place.*

Proof of claim. Suppose $x_{1,1}, \ldots, x_{1,u}$ are all equal to x_i. Then $A, T_{1,1}, T_{2,1}, \ldots, T_{1,u}, T_{2,u}$ are transactions accessing x_i (note 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) $A \rightarrow^{+} T_{2,u}$. If this is the case, a path $A \rightarrow^{+} B$ via $T_{2,u}$ exists and hence A and B cannot be part of a tr-consistent global checkpoint, by Theorem 1.

(2) $T_{2,u} \rightarrow^{+} A$. Since $T_{2,u} \rightarrow T_{1,u}$, we must have $T_{1,u} \rightarrow^{+} A$ from Observation 3. If this is the case, when we consider the reverse edge $T_{1,u-1} \leftarrow T_{2,u-1}$, the following two sub-cases arise:

(2.1) $A \rightarrow^{+} T_{2,u-1}$. In this case, a cycle $T_{1,u} \rightarrow^{+} A \rightarrow^{+} T_{2,u-1} \rightarrow^{+} T_{1,u}$ 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) $T_{2,u-1} \rightarrow^{+} A$. Since $T_{2,u-1} \rightarrow T_{1,u-1}$, we must have $T_{1,u-1} \rightarrow^{+} A$ from Observation 3. If this is the case, we need to consider the previous reverse edge $T_{1,u-2} \leftarrow T_{2,u-2}$ in the zigzag path and make a similar argument with that edge. Proceeding like this, we will end up with a path $T_{1,1} \rightarrow^{+} A$; since $A \rightarrow^{+} T_{1,1}$, we have $A \rightarrow^{+} A$, 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 $x_{1,1}, \ldots, x_{1,u}$ are all equal to x_i is wrong and hence the proof of the claim. This situation is illustrated in Fig. 5b. 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. Fig. 5c illustrates how we can get contradiction by showing the 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

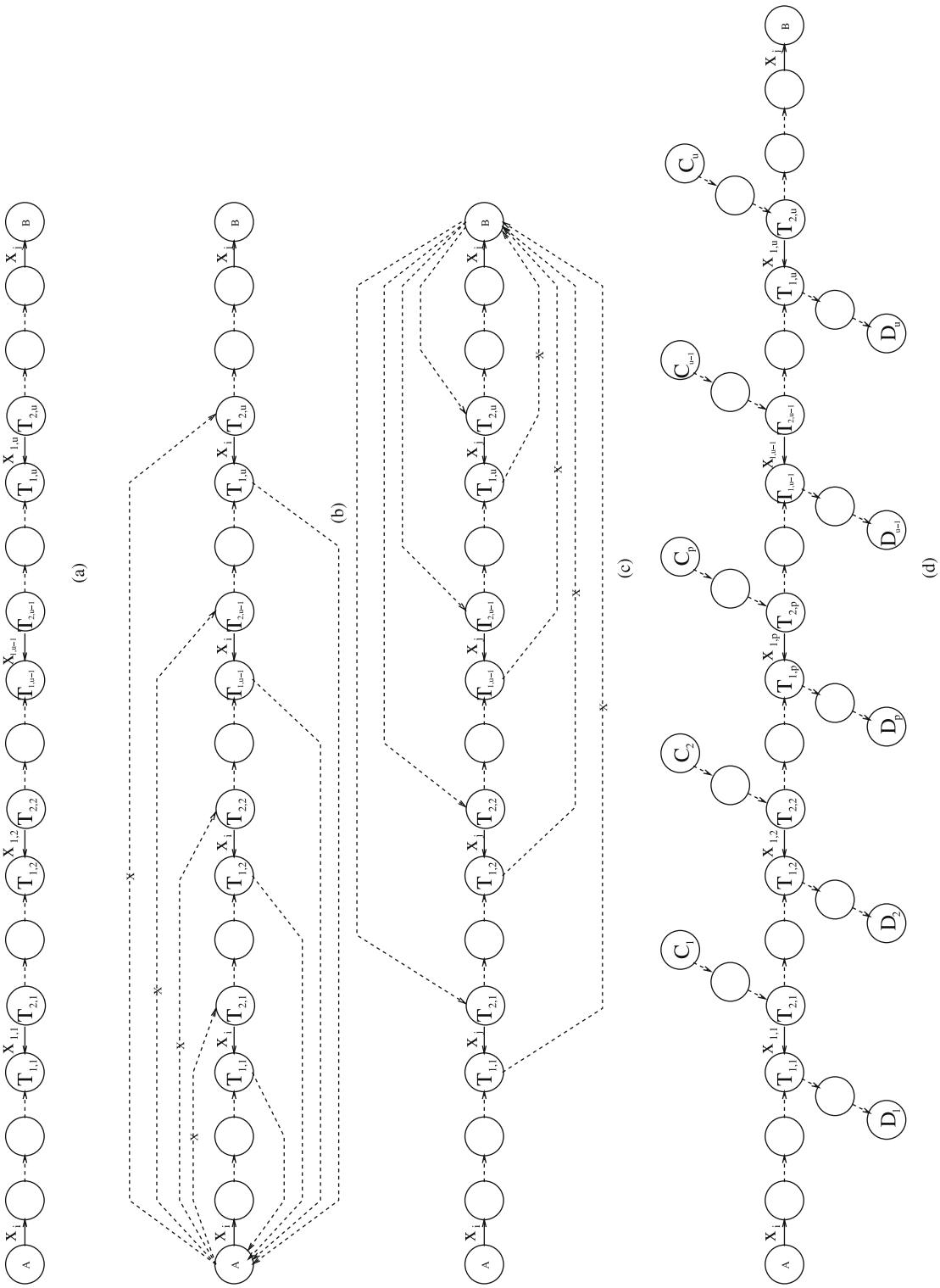

Fig. 5. Dependency of transactions I.

$T_{1,p} \leftarrow T_{2,p}$. Next we prove our claim that A and B cannot belong to a tr-consistent global checkpoint. Fig. 5d illustrates 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 Fig. 5d) can be combined with A to form a consistent global checkpoint due to the path $A \rightarrow^{+} D_1$ (from Theorem 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, any checkpoint taken after $T_{1,2}$ accessed, say D_2, cannot be combined with C_1 to form a consistent global checkpoint due to the path from $C_1 \rightarrow^{+} D_2$ (refer to Fig. 5d). 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 $C_u \rightarrow^{+} B$. 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, any checkpoint taken before $T_{2,u-1}$, say C_{u-1} cannot be combined with D_u to construct a tr-consistent global checkpoint containing due to the path $C_{u-1} \rightarrow^{+} D_u$. 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 is taken after $T_{1,p}$ accessed $x_{1,p}$, to construct a tr-consistent global checkpoint containing B.

Thus, there exists a data item $x_{1,p}$ which is neither x_i nor x_j. On such a data item, 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 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 with $w = 1$.

Next, assume that if there is a zigzag path from A to B which contains consecutive reverse edges with length at most k, then A and B together cannot belong to a tr-consistent global checkpoint. We prove that if there exists a zigzag path from A to B which contains consecutive reverse edges of length $k+1$, 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 $T_{1,1} \leftarrow \cdots \leftarrow T_{u_1,1}$ $(u_1 \leqslant k+2)$, $T_{1,2} \leftarrow \cdots \leftarrow T_{u_2,2}$ $(u_2 \leqslant k+2)$, $\cdots$, and $T_{1,v} \leftarrow \cdots \leftarrow T_{u_v,v}$ $(u_v \leqslant k+2)$. Thus, on the zigzag path from A to B that we consider, we have consecutive reverse edges of lengths $u_1 - 1, \ldots, u_v - 1$, $(u_i \leqslant k+2 \forall i)$. 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 $x_{1,1}, \ldots, x_{u_1-1,1}, \ldots, \ldots, x_{1,v}, \ldots, x_{u_v-1,v}$ respectively. Fig. 6a 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 $x_{1,1}, \ldots, x_{u_1-1,1}$ $\ldots, \ldots$, $x_{1,v}, \cdots$, $x_{u_v-1,v}$ are all the same as x_i. Then $A, T_{1,1}, \ldots, T_{u_1,1}$ $\ldots, \ldots$ $T_{1,v}, \cdots, T_{u_v,v}$ are transactions accessing x_i. Based on Observation 1, two cases arise:

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

(2) $T_{u_v,v} \rightarrow^{+} A$. Because of the sequence of reverse edges $T_{1,v} \leftarrow \cdots \leftarrow T_{u_v,v}$ on x_i, from Observation 3, we have $T_{1,v} \rightarrow^{+} A$. Then, when we consider the sequence of reverse edges $T_{1,v-1} \leftarrow \cdots \leftarrow T_{u_{v-1},v-1}$, the following two sub-cases arise:

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

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

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 $x_{1,1}, \ldots, x_{u_1-1,1} \ldots, \ldots x_{1,v}, \ldots, x_{u_v-1,v}$ are all x_i is wrong. In Fig. 6b, 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 $x_{1,1}, \ldots, x_{u_1-1,1} \ldots, \ldots x_{1,v}, \ldots, x_{u_v-1,v}$ can be equal to x_j. Fig. 6c illustrates this. Suppose $x_{1,1}, \ldots, x_{u_1-1,1} \ldots, \ldots x_{1,v}, \ldots, x_{u_v-1,v}$ are all x_j.

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 $T_{g,p} \leftarrow T_{g+1,p}$ which is one of the reverse edges in the sequence of reverse edges $T_{1,p} \leftarrow \cdots T_{u_p,p}$. Next, we prove that A and B cannot be part of a tr-consistent global checkpoint. Fig. 6d can help 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 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 $x_{1,2}$.

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 can be combined with C_{p-1}, to construct a 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 $T_{g+1,p}$ have accessed $x_{g,p}$.

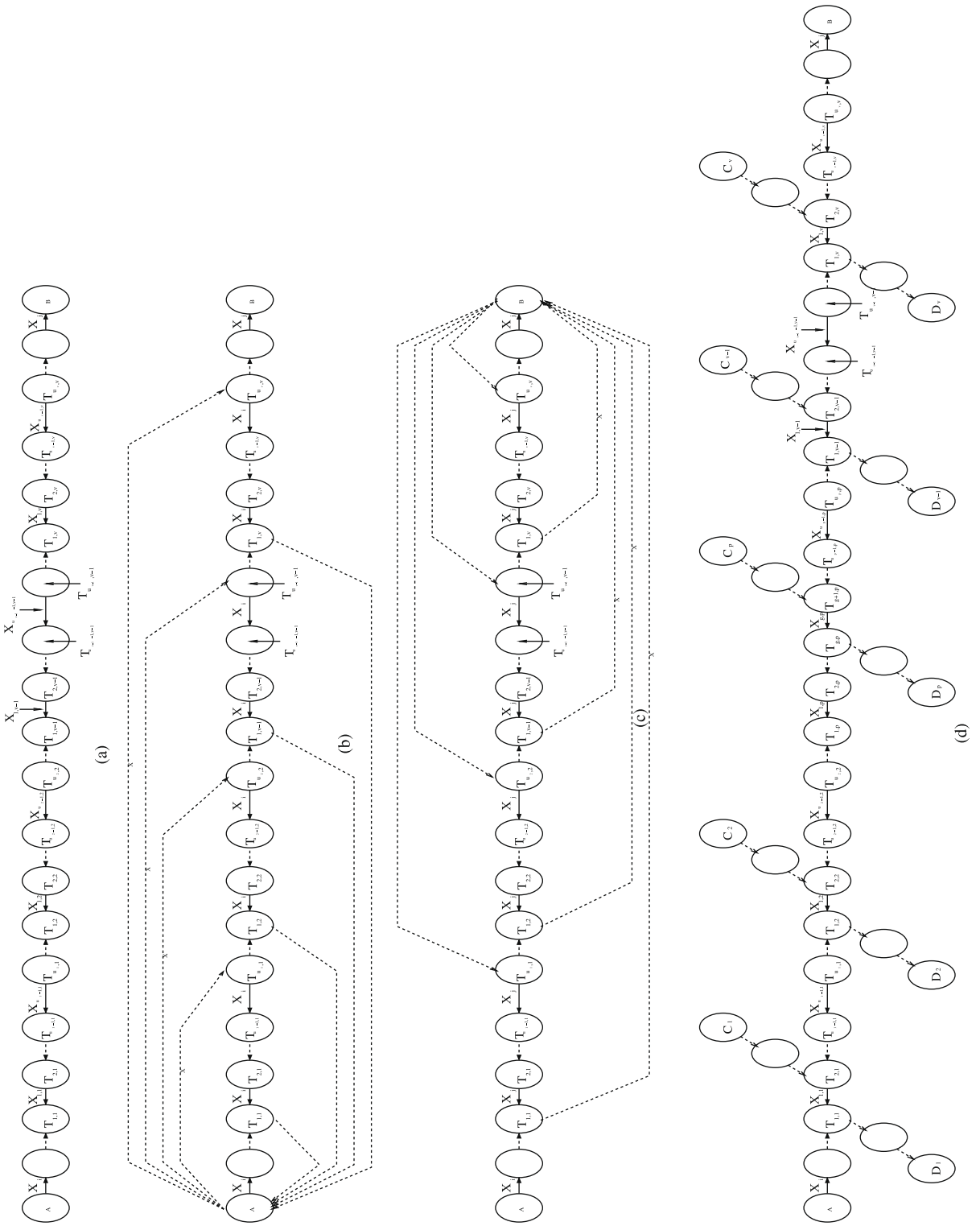

Fig. 6. Dependency of transactions II.

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 consistent global checkpoint. This proves the theorem. □

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.*

Proof. Follows from the Theorem 2 by taking $\mathbf{S}'$ as the singleton set containing the checkpoint. □

4.1. Applications

Corollary 1 and Theorem 2 are useful for constructing a 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 Theorem 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 can throw light on designing non-intrusive checkpointing algorithms that allow each of the data items to be checkpointed independently while at the same time making all checkpoints useful.

A federated database system (FDBS) is a collection of cooperating database systems [26,1,31,32,11]. Kleewein [12] 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. Eventhough 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. [20] 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. [20] 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.

5. Conclusion

Checkpointing has been traditionally used for handling failures in distributed database systems. An efficient checkpointing algorithm 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 not 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 distributed 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 a data item. Moreover, the necessary and sufficient conditions established can throw 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 (useful) part of a tr-consistent global checkpoint.

Acknowledgements

The authors thank the editor and the reviewers for their valuable and constructive comments which helped greatly in improving the content and presentation of the paper. This material is based in part upon work supported by the US Department of Treasury Award #TOS05060 and the US National Science Foundation Grant No. IIS-0414791. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Department of Treasury.

References

[1] A. Deshpande, J.M. Hellerstein, Decoupled query optimization for federated database systems, in: Proceedings of 18th International Conference on Data Engineering, 2002, pp. 716–727.
[2] Adnan Agbaria, Roy Friedman, Model-based performance evaluation of distributed checkpointing protocols, Performance Evaluation 65 (5) (2008) 341–365.
[3] Roberto Baldoni, Jean-Michel Hélary, Achour Mostéfaoui, Michel Raynal, Impossibility of scalar clock-based communication-induced checkpointing protocols ensuring the rdt property, Information Processing Letters 80 (2) (2001) 105–111.
[4] R. Baldoni, F. Quaglia, M. Raynal, Consistent checkpointing for transaction systems, The Computer Journal 44 (2) (2001) 92–100.
[5] R. Elmasri, S.B. Navathe, Fundamentals of Database Systems, Addison-Wesley, 2007.
[6] G. Ferran, Distributed checkpointing in a distributed data management system, in: Proceedings of the Real-Time Systems Symposium, Miami Beach, Florida, 1981, pp. 43–49.
[7] H. Garcia-Molina, K. Salem, Main memory database systems: an overview, IEEE Transactions on Knowledge and Data Engineering 4 (6) (1992) 509–516.
[8] Jean-Michel Hélary, Achour Mostéfaoui, Michel Raynal, Communication-induced determination of consistent snapshots, IEEE Transactions on Parallel and Distributed Systems 10 (9) (1999) 865–877.
[9] Jean-Michel Hélary, A. Mostéfaoui, R.H.B. Netzer, M. Raynal, Communication-based prevention of useless checkpoints in distributed computations, Distributed Computing 13 (1) (2000) 29–43.
[10] R.B. Hagmann, A crash recovery scheme for a memory-resident database system, IEEE Transactions on Computers 35 (9) (1986) 839–843.
[11] Jungkee Kim, Geoffrey Fox, Scalable hybrid search on distributed databases, in: Proceedings of International Conference on Computational Science, vol. 3, 2005, pp. 431–438.
[12] Jim Kleewein, Practical issues with commercial use of federated databases. in: Proceedings of the 22th International Conference on Very Large Data Bases, 1996, p. 580.
[13] Vijay Kumar, Shawn D. Moe, Performance of recovery algorithms for centralized database management systems, Information Sciences 86 (1) (1995) 101–147.
[14] H. Kuss, On totally ordering checkpoints in distributed databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1982, pp. 174–174.
[15] A.-P. Liedes, A. Wolski, SIREN: a memory-conserving, snapshot-consistent checkpoint algorithm for in-memory databases, in: Proceedings of the 22nd International Conference on Data Engineering, 2006, pp. 99–99.
[16] J. Lin, M.H. Dunham, A survey of distributed database checkpointing, Distributed Parallel Databases 5 (1997) 289–319.
[17] Yi Luo, D. Manivannan, Fine: a fully informed and efficient communication-induced checkpointing protocol, in: IEEE Proceedings of the 3rd International Conference on Systems (ICONS'08), IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 16–22.
[18] D. Manivannan, Q. Jiang, Jianchang Yang, M. Singhal, A quasi-synchronous checkpointing algorithm that prevents contention for stable storage, Information Sciences 178 (15) (2008) 3110–3117.
[19] D. Manivannan, Mukesh Singhal, Quasi-synchronous checkpointing: models, characterization, and classification, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 703–713.
[20] Juha Muilu, Leena Peltonen, Jan-Eric 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 (May) (2007) 718–723.
[21] Robert H.B. Netzer, Jian Xu, Necessary and sufficient conditions for consistent global snapshots, IEEE Transactions on Parallel and Distributed Systems 6 (2) (1995) 165–169.
[22] S. Pilarski, T. Kameda, A novel checkpointing scheme for distributed database systems, in: Proceedings of the Ninth ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems, Nashville, TN, 1990, pp. 368–378.
[23] S. Pilarski, T. Kameda, Checkpointing for distributed databases: starting from the basics, IEEE Transactions on Parallel and Distributed Systems 3 (1992) 602–610.
[24] C. Pu, On-the-fly, incremental, consistent reading of entire databases, in: Proceedings of the 11th Conference on Very Large Database, Morgan Kaufman Publishers, Los Altos, CA, Stockholm, 1985, pp. 367–375.
[25] K. Salem, H. Garcia-Molina, Checkpointing memory-resident databases, in: Proceedings of the Fifth International Conference on Data Engineering, 1989, pp. 452–462.
[26] Amit P. Sheth, James E. Larson, Federated database systems for managing distributed heterogeneous and autonomous databases, ACM Computing Surveys 22 (3) (1990) 183–286. September.
[27] A. Silberschatz, H.F. Korth, S. Sudarshan, Database System Concepts, McGraw-Hill Publishing Co., 2005.
[28] M. Singhal, N.G. Shivaratri, Advanced Concepts in Operating Systems, McGraw-Hill, 1994.
[29] S.H. Son, A.K. Agrawala, Distributed checkpointing for globally consistent states of databases, IEEE Transactions on Software Engineering 15 (10) (1989) 1157–1167.
[30] S.H. Son, An algorithm for non-interfering checkpoints and its practicality in distributed database systems, Information Systems 14 (5) (1989) 421–429.
[31] J.L. Zhao, A. Segev, A. Chatterjee, A universal relation approach to federated database management, in: Proceedings of 11th International Conference on Data Engineering, 1995, pp. 261–270.
[32] J. Leon Zhao, Schema coordination in federated database management: a comparison with schema integration, Decision Support Systems 20 (3) (1997) 243–257. July.