Online Recovery of a Distributed Database from Malicious Attack

A. Chakraborty+, M.K. Garg+, A.K. Majumdar+, S. Sural*
Department of Computer Science & Engineering+
School of Information Technology*
Indian Institute of Technology, Kharagpur, India
anindyac@cse.iitkgp.ernet.in, gargmk2000@yahoo.com,
{akmj@cse, shamik@sit}.iitkgp.ernet.in

Abstract

In this paper, we consider the problem of recovery from committed malicious transactions in distributed databases. We define several useful dependency relations among transactions and based on them present an online recovery scheme for restoring the consistency of a database.

Keywords: Distributed Database, Damage Assessment, Online Recovery, Transaction dependency graph.
Topics: Distributed and Parallel Databases, Privacy and security in database.

1 Introduction

Online approaches have been proposed for recovering database consistency after attack [1]. However, this model in general cannot restore the consistency of a database and also suffers from the problem of damage leakage. In this paper, we present an online recovery scheme which restores the consistency of a distributed database after attack. We assume that vertical partitioning is used.

2 Online Recovery Approach

Let, W = {T | T is a transaction in the window of vulnerability}. Suppose, H is the serialized history corresponding to W and M$\subseteq$W is the set of malicious transactions. At the start of H, the database state was D_I, at the end of H it is D_E. D_C is a state that would have been reached if starting from the database state D_I, the transactions in W-M were executed (i.e. if the transactions in M never occurred).

Definition 1. ***Consistent database state*** *can be defined as follows – (a) The initial database state D_I is consistent. (b) Database state reached from a consistent state after execution of a schedule containing no malicious transaction is also consistent.*

The objective of the recovery algorithm is to take the database to state D_C starting from state D_E.

On a site s, the k-th sub-transaction of a transaction T_i is denoted by T^s_{ik}. column_read_set(T^s_{ik}) and column_write_set(T^s_{ik}) are the set of columns read and modified by the queries in T^s_{ik}. On a site s for two sub-transactions T^s_{ik} and T^s_{jl}, column_dependent_subtran(T^s_{ik},T^s_{jl}) is a binary relation defined as column_dependent_subtran(T^s_{ik},T^s_{jl}) $\iff$ [[column_read_set(T^s_{ik}) $\cap$ column_write_set(T^s_{jl}) $\neq \emptyset$] $\vee$ [column_read_set(T^s_{jl}) $\cap$ column_write_set(T^s_{ik}) $\neq \emptyset$]] $\wedge$ [$T_j <_H T_i$] (i.e. T_i occurs after T_j in H).

A committed transaction T_i is dependent on committed transaction T_j if at least one of the sub-transactions of T_i is column dependent on some sub-transaction of T_j. We term this relationship as column_dependent_tran(T_i,T_j). column_dependent is the transitive closure of column_dependent_tran.

For a site $s, G_{Ls}(V_{Ls},E_{Ls})$ (Local Dependency Graph (LDG)) is a DAG where V_{Ls} = {T_i| some sub-transaction T_{ik} of T_i is executed on site s} and E_{Ls} = {(T_i, T_j)| $\exists$ T^s_{ik}, T^s_{jl} [*column_dependent_subtran*(T^s_{jl},T^s_{ik})]}. $G_G(W,E_G)$ (Global Dependency Graph (GDG)) is a DAG where an edge $(T_i, T_j) \in E_G$ iff *column_dependent_tran*(T_j,T_i). Let, V_A = {T_k : T_k $\in$ M (the set of malicious transactions) or *column_dependent*(T_k,T_i) is true where T_i $\in$ M and T_k $\in$ W}. G_A is a subgraph of G_G induced by the set of vertices $V_A \subseteq$ W.

We use Central Recovery Coordinator(CRCO), LDG Generator, Compensation Manager (CM), Middle Tier (MT). A single instance of CRCO runs in the system. An instance of CM, MT and LDG Generator run on each site in the system. The recovery phases are Damage Assessment (DA), Resume, Compensation and Re-execution. The DA phase is started by an intrusion detector by sending M to CRCO. After Resume phase, while accepting a new transaction, MT first checks that after execution whether this transaction will become a part of G_A. If yes, then that transaction is blocked, otherwise the new transaction is allowed to execute. The recovery algorithms are given in Algorithm 1, 2, 3 and 4.

Algorithm 1 : Algorithm At CRCO
site_list = {s | s is a site} **/*DA Phase*/**
affected_trans = M, success_list = {}, LDG_list = {}, affected_graph = {}, E_G = {}
for all ($s \in$ site_list) **do**
    sendMsg(s, MT, "BLOCK")
while ($\exists s[s \in$ site_list $\wedge s \notin$ success_list]) **do**
    Wait for a SUCCESS message and E_{Lx}
    if (SUCCESS message received) **then**
        LDG_list = LDG_list $\bigcup$ {E_{Lx}}
        success_list = success_list $\bigcup$ {x}
Combine all the LDGs in LDG_list to build G_G
for all ($(T_i, T_j) \in E_G$) **do**
    if ($T_i \in$ affected_trans) **then**
        affected_graph = affected_graph $\bigcup$ {(T_i, T_j)}
        if ($T_j \notin$ affected_trans) **then**
            affected_trans = affected_trans $\bigcup$ {T_j}
for all ($s \in site\_list$) **do**
    sendMsg(s, MT, "GDGA" + V_A
Wait for SUCCESS message from all MTs **/*Resume Phase*/**
ack_list ={}, compensated = {}, reexecuted = {}, cur_compensating = {}, cur_reexecuting = {} **/*Compensation Phase*/**
for all ($T_i \in V_A \wedge T_i \notin$ reexecuted) **do**
    if (SUCCESS message arrives for a transaction T_x from site s) **then**
        ack_list = ack_list $\cup$ {s,T_x}
        if ($\forall st[st \in site\_list \Rightarrow (st, T_x) \in ack\_list]$) **then**
            if ($T_x \in$ cur_compensating) **then**
                cur_compensating = cur_compensating - {T_x}
                compensated = compensated $\cup$ {T_x}
    else if (parentsof(T_i) $\subseteq$ compensated **and** $T_i \notin$ compensated
        $\cup$ cur_compensating) **then**
        cur_compensating = cur_compensating $\cup$ {T_i}
        for all ($s \in$ site_list) **do**
            sendMsg(s, CM, "COMPENSATE T_i")
    else if ($T_i \notin$ M **and** parentsof(T_i) $\subseteq$ reexecuted **and** $T_i \in$ compensated **and** $T_i \notin$ reexecuted $\cup$ cur_reexecuting **and** childrenof(T_i) $\subset$ compensated) **then**
        submit R_i to the database for execution
        cur_reexecuting = cur_reexecuting $\cup$ {T_i}
    else if ($T_i \in$ cur_reexecuting **and** R_i committed) **then**
        cur_reexecuting = cur_reexecuting - {T_i}
        reexecuted = reexecuted $\cup$ {T_i}
end

Algorithm 2 : Algorithm At MT on site s
while(*true*) **do**
    Wait for some message to arrive
    if (BLOCK message is received) **then**
        Stop accepting new transactions **/*DA Phase*/**
        sendMsg(s, LDG Builder, "START")
    else if (GDGA message is received) **then**
        for all ($T_i \in V_A$) **do /*Resume Phase*/**
            Abort the transaction T_i
        Resume MT for accepting new transactions
        sendMsg("NULL", CRCO, "SUCCESS")
end

Algorithm 3 : At LDG Generator at site s
while (*true*) **do /*DA Phase*/**
    E_{Ls} = {}
    Wait for START message to arrive **/*from MT on site s*/**
    if (START message is received) **then**
        for all ($T_{ik} \in V_{Ls}$) **do**
            if ($\exists T_{jl}[T_{jl} \in V_{Ls} \wedge$ column_dependent_subtran(T_{ik}, T_{jl})]) **then**
                E_{Ls} = E_{Ls} $\bigcup$ {(T_i, T_j)}
        sendMsg("NULL", CRCO, "SUCCESS "+E_{Ls})
end

Algorithm 4 : Compensation Algorithm At CM at site s
cleaned_item_list = {}
while (*true*) **do**
    Wait for COMPENSATE message to arrive
    if (COMPENSATE T_i message is received) **then**
        if (T_i has a sub-transaction executed on this site) **then**
            build C_i = {(x,v) | x $\in$ WS(T_i) and $\forall$ j,u (x,j,u) $\notin$ cleaned_item_set. v is the value of x before x was modified by T_i}
            submit C_i to the database for execution
            cleaned_item_set = cleaned_item_set $\cup$ {(x,i,v) | (x,v) $\in C_i$}
            sendMsg("NULL", CRCO, "SUCCESS")
end

3 Analysis of the Proposed Approach

Note, the proposed approach prevents damage leakage. Complete damage leakage prevention implies recovery time is less in the proposed approach compared to [1]. Moreover, termination is guaranteed (when $V_A = \emptyset$) and no additional termination detection algorithm is required.

Central Dependency Graph (CDG) is the graph that would have been built if all the transactions had executed on a single site.

Theorem 1 : CDG and GDG are isomorphic.

Theorem 2 : G_A is isomorphic to the affected graph built from the CDG, say CDG_A.

Theorem 3 : Compensating the transactions in G_A in bottom up order (as in static recovery algorithm) or in top down order (by maintaining a cleaned_item_set, as in Algorithm 4) is equivalent.

Theorem 4 : A transaction T_i can be re-executed, independent of any other transaction, after its parents in G_A (except the malicious transactions) have been re-executed and T_i and all its children in G_A have been compensated.

It can be shown that the proposed online recovery approach has a message complexity of $O(|V_A|)$.

4 Conclusion

In this paper, we have identified the problems caused by committed malicious transactions in distributed database systems and developed a set of dependency relationships. Based on these, an online scheme for recovering the database from the damage has been proposed.

Acknowledgments

This work is partially supported by a research grant from the Department of Information Technology, Ministry of Communication and Information Technology, Government of India, under Grant No. 12(34)/04-IRSD dated 07/12/2004 and by a grant from Microsoft Research.

References

[1] P. Liu, X. Hao, "Efficient Damage Assessment and Repair in Resilient Distributed Database Systems", IFIP TC11/WG11.3 Fifteenth Annual Working Conference on Database and Security, pp. 75–89, July 15-18, 2001.