RDB: Repairable Database Systems

Alexey Smirnov and Tzi-cker Chiueh

Experimental Computer Systems Lab, Department of Computer Science, SUNY Stony Brook

Motivation

Suppose you are a DBA and you have just noticed that your database has been compromised 24 hours ago. How would you repair the database?

Currently, the only way to do this is to restore a database backup and recommit all benign transactions *manually*.

Challenges: (1) How to tell which transactions are benign and which are malicious? Identifying the initial set of malicious transactions is not enough because initial damage can spread over the database by subsequent benign transactions. (2) The amount of data can be huge and the repair process is very error-prone. There is a need a way to automate it.

Ideally, an *intrusion-resilient* DBMS should be able to

- Track inter-transaction dependencies;
- Perform a selective transaction rollback.

We propose an implementation framework called **RDB** that can render an off-the-self DBMS intrusion resilient without modifying its internals. RDB has two major components: *tracking subsystem* which runs at run-time and *recovery subsystem* which runs offline.

Definition of Transaction Dependency

A *read set* of an SQL statement S is the set of rows fetched by this statement.

We will say that statement S_2 depends on statement S_1 if at least one row from the read set of S_2 was modified by S_1. We will say that transaction T_2 depends on transaction T_1 if at least one statement of T_2 depends on a statement from T_1. This definition is prone to both false positives and false negatives. Example of a false positive dependency:

A1	A2	A3
100	5	5
200	5	6
300	1	7

- T1: SET A2=5 WHERE A1<250
- T2: SELECT A3 WHERE A3>3

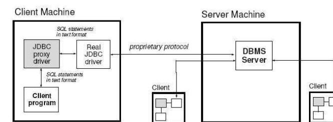
Also, in general it is impossible to determine all transaction dependencies by looking at the traffic between a client and the DB server only because part of the logic may be inside the application itself.

Transaction Dependency Tracking

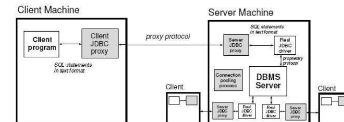
To track dependencies successfully, the tracking mechanism should be able to intercept both read and write actions performed by the database server. Possible ways of interception are:

- Database triggers (online) – cannot add a trigger for SELECT statements;
- Database log analysis (offline) – read operations are not logged; no run-time overhead is its big advantage.
- Tracking proxy (online) – a small program sitting between a client and a server that intercepts all SQL statements sent by the client and results sent back by the server.

RDB uses both second and third approaches to implement dependency tracking.



RDB inserts a proxy JDBC driver between the DB server and a client that transparently intercepts all queries and results. The proxy can be either on the client side or on the server side.



The following changes are made to the database at the time of its creation:

- A new field `tr_id` is added to each table. It contains the ID of last transaction that modified a particular row;
- Table `trans_dep(tr_id:INTEGER, dep_ids:VARCHAR)` – stores IDs of transactions that depend on transaction `tr_id`;
- Table `annot(tr_id:INTEGER, descr:VARCHAR)` – stores annotations for transaction tr_id;

The proxy uses its own transaction Ids because there is no standard way to access the internal transaction ID of a database.

SQL Statements Rewriting

As a part of the dependency tracking mechanism, SQL proxy rewrites certain classes of SQL statements coming from a client.

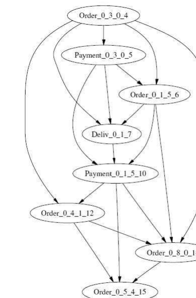
Original statement	Modified statement(s)
SELECT $t_1.a_1,\ldots,t_1.a_n,\ldots,t_k.a_{nk}$ FROM $t_1,\ldots,t_k$ WHERE c	SELECT $t_1.a_1,\ldots,t_1.a_n,\ldots,t_k.a_{nk}$, $t_1.trid,\ldots,t_k.trid$ FROM $t_1,\ldots,t_k$ WHERE c
SELECT SUM(*t.a*) FROM *t* WHERE *c* GROUP BY *t.b*	SELECT *t.trid* FROM *t* WHERE *c* SELECT SUM(*t.a*) FROM *t* WHERE *c* GROUP BY *t.b*
UPDATE *t* SET $a_1=v_1,\ldots,a_n=v_n$ WHERE c	UPDATE *t* SET $a_1=v_1,\ldots,a_n=v_n$, *trid=curTrID* WHERE c
INSERT INTO $t(a_1,\ldots,a_n)$ VALUES $(v_1,\ldots,v_n)$	INSERT INTO $t(a_1,\ldots,a_n,trid)$ VALUES $(v_1,\ldots,v_n,curTrID)$
COMMIT	INSERT INTO *trans_dep(curTrID,…)* COMMIT

Database Repair

The database is repaired by *compensating* malicious transactions. When using RDB, the repair process consists of the following steps:

- Database log analysis to reconstruct complete dependency information and generate compensating transactions;
- Dependency graph visualization;
- Repairing database by committing compensating transactions.

Different DBMSs provide different facilities for log analysis. We have studied three database servers: Oracle 9.2.0, PostgreSQL 7.2.2, and Sybase ASE 12.5. Eventually, all of them provide enough information to generate compensating transactions.



We used GraphViz – a free graph drawing software from AT&T.

The application allows the user to select an initial set of malicious transactions and computes its transitive closure. Then the result can be refined by the user to build the final set of transactions to be compensated.

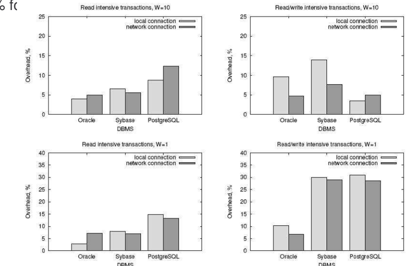
Performance Results

We used TPC-C benchmark to evaluate the run-time overhead of JDBC proxy. The size of the test database was about 4GB.

We varied the following parameters:

- Transaction mix (read intensive and read/write intensive);
- Connection type (local or over a network);
- Total footprint size W (effect of database cache);

Our results suggest that the overhead of the proxy is between 6% and 13% f



Contact Information

ECSL Lab at SUNY Stony Brook:
http://www.ecsl.cs.sunysb.edu

E-mail: {alexey,chiueh}@cs.sunysb.edu