Tolerating Byzantine Faults in Databases

Ben Vandiver (benmv@mit.edu)
Hari Balakrishnan, Barbara Liskov, and Sam Madden

Bugs Cause Byzantine Faults

- Over 50% of reported bugs are non-crash faults
- Results from our informal survey:

<table>
<thead>
<tr>
<th>Bug Category</th>
<th>DB2 2/03-8/06</th>
<th>Oracle 7/06-11/06</th>
<th>MySQL 8/06-11/06</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS Crash</td>
<td>120</td>
<td>21</td>
<td>60</td>
</tr>
<tr>
<td>Non-Crash Faults</td>
<td>131</td>
<td>28</td>
<td>64</td>
</tr>
</tbody>
</table>

- Fail-stop model: faulty nodes crash & stop processing
- Byzantine model: faulty nodes capable of arbitrary behavior
- Novelty: Previous work excludes Byzantine faults

Model

- Middleware presents single-copy serializable view of replicated database
- Tolerating \( f \) faults requires \( 2f+1 \) database replicas
- Guarantees replicas stay synchronized and clients see correct answers for all transactions that commit
- Requires databases implement SERIALIZABLE isolation via rigorous 2-phase locking

Commit Barrier Scheduling

- With multiple concurrent transactions, database replicas could execute transactions in different orders
- Our solution, called Commit Barrier Scheduling, uses a primary/secondary scheme which replicates the primary’s ordering on the secondaries
- Statements initially execute on the primary, and are queued for secondaries when the primary finishes
- Rules for issuing statements on secondaries:
  - Commit-ordering rule (correctness) - A COMMIT for transaction \( T \) can be sent to a secondary only after the secondary has processed all queries of all transactions ordered before \( T \).
  - Transaction-ordering rule (performance) - A query from transaction \( T \) that was executed by the primary after the COMMIT of transaction \( T \) can be sent to a secondary only after it has processed all queries of \( T \).

Protocol

- Use Barriers to track ordering information
- Primary’s schedule as observed by the coordinator:
  - \( B=0 \):
    - \( T_1 \): \( r^{(x)} \), \( w^{(x)} \), \( C \)
    - \( T_2 \): \( r^{(y)} \), \( w^{(y)} \), \( C \)
    - \( T_3 \): \( r^{(z)} \), \( w^{(z)} \), \( C \)

- Use Voting to determine correct answers:
  - \( f+1 \) matching votes from up-to-date replicas
  - Faulty primary is tricky to deal with
  - \( T_1 \), \( T_2 \) each performs: \( A = A + 1 \), read \( A \) starts 0
  - \( f+1 \) votes for both answers!
  - Return primary’s answer directly, vote at commit, abort transaction at commit if client received incorrect answers.

Architecture

- Client
- Shepherd
- Coordinator
- Primary Replica Manager
- Secondary Replica Manager
- Primary DB
- Secondary DB
- Secondary DB

Performance Results

- Fault tolerance requires failure independent replicas
- Heterogeneous vendors:
  - Microsoft
  - Oracle
  - IBM
  - MySQL
  - PostgreSQL
  - Others

- Cross vendor bugs are very rare
- Heterogeneous versions for upgrade/regression tests
- Non-invasive solution leverages SQL standards
- Resolve SQL incompatibilities via translation or hiding
- CBS does not work with Snapshot Isolation (yet)

Heterogeneous Replication

- \( MySQL \) – 1 test machine, 1 db machine
- \( Passthrough \) – add no-op middleware machine
- CBS – Full shepherd running CBS, 3 MySQL db machines
- Serial – Shepherd serializes transactions instead of CBS
- 17% overhead for CBS at 80 clients

Conclusions

- Detect and tolerate serious bugs in databases with little performance overhead
- Find new bugs by running different database versions
- Future work in snapshot isolation, replica state repair, bug finding, and replication of the shepherd

Sponsors: Quanta Computer Inc, NSF
URL: http://www.pmg.csail.mit.edu/bft

Overhead testing using TPC-C

<table>
<thead>
<tr>
<th>Replica</th>
<th>Primary Faults</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 A=1</td>
<td>T2 A=1</td>
<td>T1 A=1</td>
<td>T2 A=1</td>
</tr>
</tbody>
</table>