FIT: A Distributed Database Performance Tradeoff

Faleiro and Abadi
CS590-BDS
Thamir Qadah
Desirable features in Distributed Databases

- It is impossible to have it all. Only two out of the three are achievable simultaneously.
- CAP was proposed by Eric Brewer (a distributed systems researcher) and does apply perfectly to distributed database systems.
Assumptions and Definitions

- Sharded data (partitioned): data is distributed across multiple partitions.
- Distributed transactions have **read-sets** and **write-sets** from multiple partitions.
- Transactions must either COMMIT or ABORT.
  - ABORTS can be **logic-induced** or **system-induced**.
  - **Logic-induced** aborts by transaction logic based on application semantics.
    - Abort a balance transfer if source balance will be negative.
  - **System-induced** aborts by transactional system.
    - e.g. in order to avoid deadlocks.
Safety, Liveness and Atomicity

A database is required to satisfy the following properties, when processing distributed transactions:

- **Safety**: a transaction is allowed to commit if all partitions can commit, otherwise it must abort.

- **Liveness**: when a transaction is aborted by the system, and retried, it must eventually commit.

- **Atomicity**: All updates of a transaction must be reflected in the database state if it is committed, and none are reflected if it is aborted.
Distributed transactions

- Transactions involving data that reside on multiple nodes are called distributed transactions
Distributed transactions

- Transactions involving data that reside on multiple nodes are called **distributed transactions**

```
T:
  a = a - 5
  e = e + 5
```
Distributed transactions

- Transactions involving data that reside on multiple nodes are called distributed transactions.

```
Client (Application)  Transaction Manager
  T (part1) :  a = a - 5     T (part2) :  e = e + 5
  a,b,c,d          e,f,g,h
    P1              P2
```
Distributed transactions

- Transactions involving data that reside on multiple nodes are called **distributed transactions**
Fairness

● When to block a transaction from continuing its execution?
  ○ Blocking due to a concurrency control mechanism to ensure isolation and consistency (reason #1)
  ○ Blocking to improve throughput
    ■ e.g batching operations or log records

● When it is unfair to block a transaction from progressing?
  ○ For any reason other concurrency control

● Examples of unfairness:
  ○ Group Commit: batches log records write operations delays some transactions
  ○ Lazy transaction evaluation: batches transactions that have spatial locality
Synchronization Independence

- Transactions do not block each other even when they are conflicting
- Synchronization independence implies weak isolation
FIT Tradeoff

- Coordination among conflicting transactions has a cost.
- If the system pays this cost during executing the transaction, it is considered fair.
  - Conflicting transactions and non-conflicting transactions are treated equally as they all start ASAP.
- If the system pays this cost, before (or after) transaction execution, it is considered unfair.
- Intuitively, stronger isolation implies lower throughput.
  - Conflicting transactions are blocked from making meaningful progress due to synchronization and distributed coordination overhead.
- In general, most systems sacrifice fairness to obtain strong isolation and high throughput.
### FIT in action

<table>
<thead>
<tr>
<th>System</th>
<th>Fairness</th>
<th>Isolation</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-Store</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Calvin</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Google Spanner</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Cassandra</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>RAMP</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Silo</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Doppel</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Paper Criticism and Research Questions

- Why **fairness** is a desirable feature? Why do we need to guarantee **fairness**? Isn’t **liveness** enough?
- How to formally characterize **fairness**? If we bound the **unfairness**, does that make us fair?
Thank You