Eventual consistency, Data Quality, and CAP

Saibabu Devabhaktuni, PayPal

May 13, 2016

Agenda

- Data Consistency
- Data Quality
- CAP
- Summary

Why are we talking about this?

- Single databases cannot keep scaling up
- Databases needs to be “split” to scale
  - Requires application workflows to split on database boundaries
  - No transactions across boundaries
  - May require data model change
  - Access patterns also change
  - Sharding/Replication are popular methods to scale out
- Databases are now disjoint- “Distributed, Shared-Nothing”
- Disjoint databases will fail disjointedly
- Introduces inconsistencies at various levels

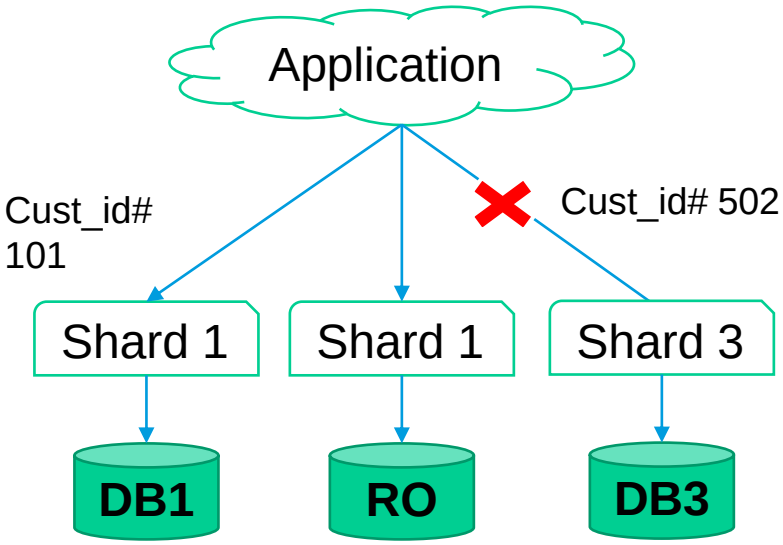
Data Consistency Types

- Fully consistent across all datasets
- Consistent at individual record level in a table
- Relationally consistent with stale data
- Relationally not consistent with stale data

Application Consistency Types

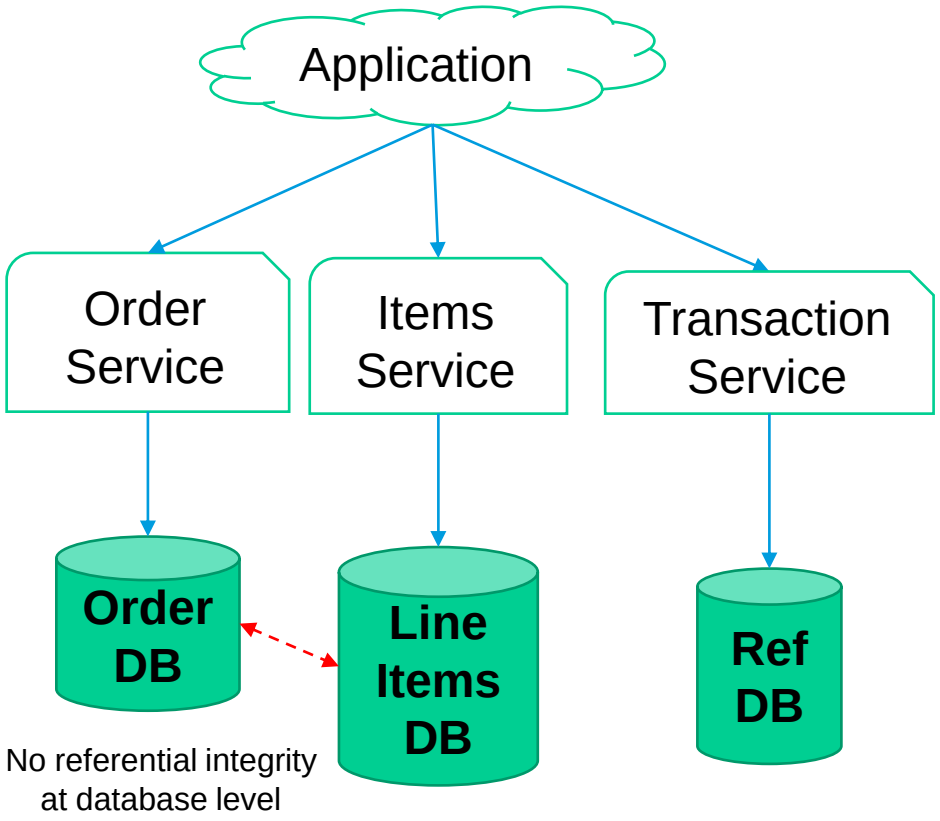
- Consistency at workflow level
- Consistency at transaction level
- Consistency at data model level
- Consistency at business entity level

Consistency at Transaction Level



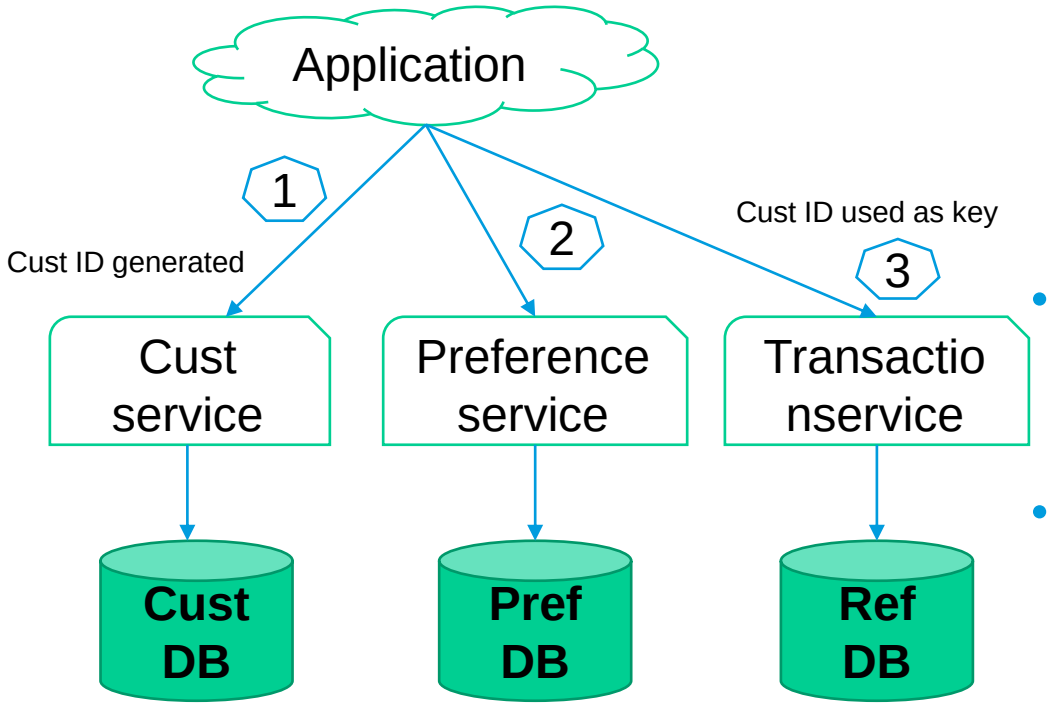
- Cust_id# 101 address updated on Shard 1
- Subsequent read go to lagged Shard 1 RO
- Transaction based on Shard 1 RO query cause inconsistency

Consistency at Data Model Level



- Application uses multiple services and Databases
  - “Order DB” stores Order details
  - ”Line Items DB” stores Order’s line items
  - ”Ref DB” stores transaction details
- Data model requires referential integrity between Order (parent) and Line Items (child)
- Data model’s integrity constraint needs to be maintained at the application level rather than traditional database level
- Transactions cannot span database boundaries in such distributed databases

Consistency at Business Entity Level



- Application uses multiple services and Databases
  - “Cust DB” stores Customer details
  - ”Pref DB” stores Customer’s preferences
  - ”Ref DB” stores transaction details
- Although distributed and asynchronous, Preference and Transaction service depends on ID generated by Cust DB
- Cust ID’s failures all subsequent services

Data Quality

- Maintaining business rules within and across data entities at rest and during state changes
- Impact based on quick detection of data quality issues
- Ability to repair data quality issues

CAP

- Consistency: Ability to read latest data
- Partition: Data distributed across nodes with each node acting as a partition
- Availability: Ability to access data distributed across nodes within a system
- CAP = You can choose only one of either C or A in the presence of P

Distributed Database Failure Categories

- Any logical entity with the same code can fail as a whole
- Cascading node failures can lead to cluster failure
- Cascading replication problems
- Cascading data quality problems
- Cascading capacity problem can lead to poor performance

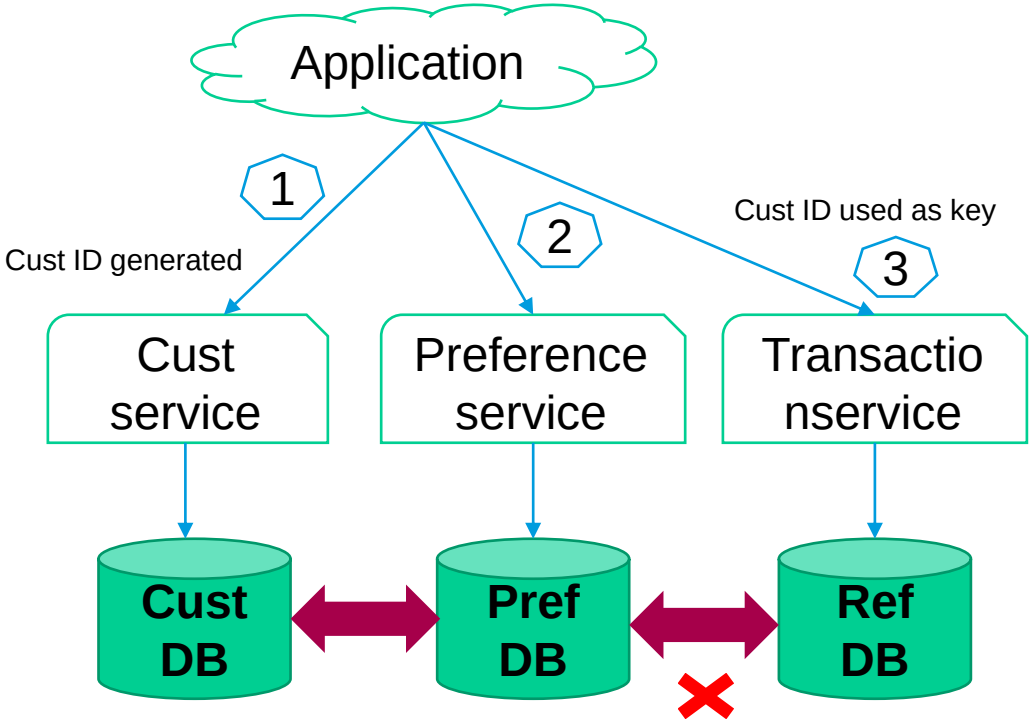
Design Approach

- Ability to partition by region
- Micro services
- Shard by business entity (e.g., customer)
- CAP on each shard

Design for CAP

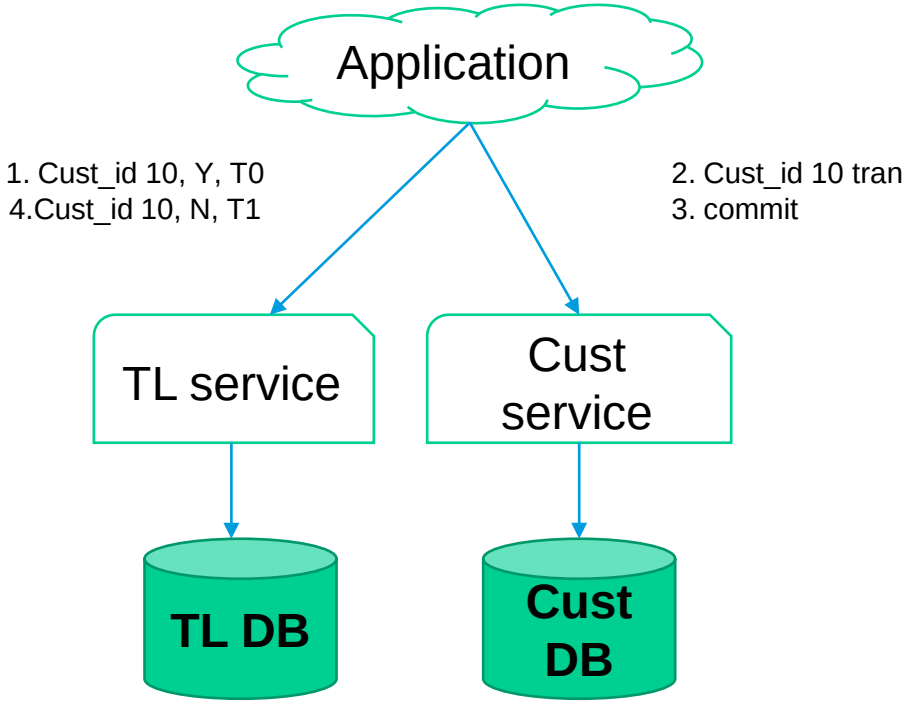
- C and A at most granular business entity level
- Transaction lock (TL) at entity level in a separate data store
- Consistent reads with replication lag and TL
- Consistent reads with quorum based reads
- Consistent writes during partition by leveraging TL
- Batch based framework to handle stale TL

CAP with 2pc



- Same scenario as before *but with DB Links between all databases*
- Distributed transactions with 2 phase commit
- Provide C and P for transactions
- But a big hit on “A”!
- Network breaks or downtime on ANY of the 3 databases breaks ALL functionality
- Does not scale linearly

Transaction Lock



- K/V pair lock at entity level
- Define staleness time (i.e., 5 mins, 15 mins)
- Framework to handle and purge stale data

Summary

- Implement data consistency per application work flow and transaction requirements
- Define eventual consistency at entity level
- Don’t decouple data quality from eventual consistency
- Apply CAP at data partition level
- Manage CAP at entity level to get varying degree of C, A, and P at micro service level