Do Relational Databases Belong in the Cloud?

Michael Stiefel
www.reliablesotware.com
development@reliablesotware.com
How do you model data in the cloud?
Relational Model

A query operation on a relation (table) produces another relation (table).

Based on the relational algebra and calculus, a query engine can produce provably correct results.
Declarative Language Allows Optimization
Architectural Assumption:
Data Outlasts Implementation
Data Separate From Code
Consistency Required

Transactional consistency
   No specification of insert, update or delete.
   Non clustered indices consistent with data

Design consistency
   Denormalized data must be kept consistent
   Lossless join decompositions
Transactional Consistency Means Holding Database Locks
Holding Locks Interferes With Availability and Scalability
Do Availability and Consistency Conflict?
Laws of Physics
Technology Limits
Economics
Laws of Physics
Latency Exists

Speed of light in fiber optic cable: 124,000 miles per second

Ideal ping Japan to Boston takes 100 ms.

Fetch 10 images for a web site: 1 second

Ignores Latency of the operation
Bandwidth is Not Cheap

Shannon's Law: $C = B \log_2 (1 + S / N)$
Capacity = bit / second
Bandwidth (hertz)
S/N * 5 to double capacity given bandwidth
Latency is Not Bandwidth

Size of the shovel vs. how fast you can shovel

Infinite shovel capacity (bandwidth) is limited by how fast one can shovel (latency).
Great Bandwidth Terrible Latency

Buy a two terabyte disk drive

Drive with it from Boston to New York
You can only move data so fast

You can only move so much data
Technology Limits
Connectivity is Not Always Available

Cell phone
Data Center Outages
Equipment Upgrades
Data redundancy to improve reliability
Offline mode on client for availability
Expensive to Move Data

Data naturally lives in multiple places
Computational Power gets cheaper faster than network bandwidth
Cheaper to compute where data is instead of moving it

* Distributed Computing Economics * Jim Gray
Economics Dictate Scale Out, Not Up

Cheap, commodity hardware argues for spreading load across multiple servers

Relational Databases were not designed to be run on clusters (shared disk subsystem)
Wind up Building a Distributed System
Can the relational database scale?
Traditionally, focus was on optimizing specific problems
Optimize Insert/Update or Read?

Data intensive relational applications:
- frequent small read / writes
- large size reads, but infrequent writes

Problems:
- Heavy workloads with frequent writes
- Scanning over large indices for queries
- Dirty reads can mean inconsistent data
What does it mean to scale?

Large Number of Users
Geographic Distribution
Hugh Amounts of Data
To Scale a Distributed System
Focus on Data, Not Just Computation
CAP Theorem

Can Have Any Two

Consistency

Availability

Tolerance to Network Partitioning

Eric Brewer
UC Berkeley, Founder Inktomi

Consistency and Availability

- Single site Database
- Database Cluster
- LDAP
- Two phase commit
- Validate Cache
Consistency and Partitioning

- Distributed Database
- Distributed Locking
- Pessimistic Locking
- Minority Partitions
  invalid
Availability and Partitioning

Forfeit Consistency

Google Big Table
Amazon Simple DB
Azure Storage Tables

Optimistic Locking
Can Denormalize
CAP Does *Not* Imply:

Never give up on Durability
Atomicity within a Partition
Inconsistency should be the exception
Partition Everywhere
No ACID within a Partition
Give up on Declarative Languages such as SQL
Then…

If we give up Consistency, how do we Partition?

If we Partition how do we recover system invariants?
Classic Ways to Partition
Distributed Objects

Distributed Objects Fail
  Separate Address Space
  Disparate Lifetimes
  Location is Not Transparent

RPC Model Fails
Cannot Hide Network
Distributed Transactions

Relational Model works with single node/cluster
- Complexity of relations
- Query plans with hundreds of options which query analyzer evaluates at runtime
- Normalization
- ACID Transactions

Quick hardware scale up difficult
Two Phase Commit works with infinite time
Better Ways to Partition

Non-Relational Approach
- Key Value / Tuple Store
- Document Store
- Column Family Store
- Graph Store

Relational Approach
- Sharding

NewSQL
For Better Partitioning, Look at Data Model
Relational: Given the structure of the data, what kind of questions can I ask?
Non Relational: Given the questions I want to ask, how do I structure the data?
Model Application Specific Questions
The aggregate is the unit of atomicity in a NoSql Data Model
Relational vs. Aggregate

```
Venue {
    Name
    Country
    Event []
    {
        Name
        Ticket Inventory
        Artist
        Date
        Genre
        Location
    }
}

Customers {
    Name
    Orders[]
    Event Name
    Number Tickets
}
```
Prioritized Query Restrictions

1. How many tickets are left for an event?
   date, location, event

2. What events occur on which date?
   date, artist, location

3. When is a particular artist coming to town?
   artist, location

4. When can I get a ticket for a type of event?
   genre

5. Which artists are coming to town?
   artist, location
Query Analysis
Most common combination: artist or date / location
Most common query: event / date / location
Partition based on location or venue
 Allows for geographic sensitivity
Partitioning may or may not imply denormalization
Each NoSql Data Model Treats Aggregates Differently
In general....

Code has integrity constraints
Code handles joined queries
No standard among vendors (lock in)
Key-Value treats the aggregate as opaque
Might have a opaque set of attributes
Key is the index to the aggregate
Ordered Key-Value allows for range queries
Only the application knows the schema
Column Family is a Two Level Aggregate
   Keys are first level
   Aggregates are the second level
   Aggregate is composed of other aggregate
Reads are common, Writes rare
Column Family Data Model (Cassandra)
Example

```json
Event
{
  key 100
  {
    Boston Symphony {phone: 617-555-1212, desc: Beethoven concert.},
    Lady Gaga {phone: 310-539-4242, desc: Monster concert},
  } //end concerts
  key 200
  {
    Boston Common Walk {desc: Swan boats are here.},
    Community Boating {phone: 617-555-1000, desc: Join us this summer.}
  } // end community events
}
```

- **Super Column Family**
- **Column**
- **Key**
- **Super Column**
- **Flexible Schema**
Document Database has aggregate of arbitrary complexity with an index on attribute data.
Mechanics of Relational Database Partitioning
Find Independent Units of Data
Separate Transactions From Queries

- Read
- Create
- Update
- Delete
Transactional Units Across Databases

Partitioning Function
Partitioning Mechanisms

Horizontal Partitioning
- Divide table rows across databases

Vertical Partitioning
- Divide table columns across databases
- Different tables in different databases
- Reference data can be copied
- Queries scan less data
Horizontal Partitioning

Each table contains identical columns.
Data is partitioned into different databases.

Each part is referred to as a shard.

Table is a single logical entity for updates and queries.
Indices for a shard must be in the same shard.

Sharding strategy based on use or query patterns.
Implementing Horizontal Partitions

Function that converts sharding property into a database location

Primary keys unique across all shards
- Shards hand out distinct ranges
- Shard id is part of primary key
- Pool hands out unique identifiers

No secondary keys across shards
No distributed transactions across databases
May need to UNION query results
Vertical Partitioning

Divide table columns across databases
Primary key identical for a given "row"
Data may or may not be normalized
A join across the partitions recreates the "row"
Vertical Partitioning Strategy

Columns used in different queries go in different partitions
Different business processes "own" a table.
  Leads to service oriented approach
  Design business processes to avoid cross table joins
Transactions within service boundary
Implementing Vertical Partitions

Primary or foreign keys may be used to recreate the row
No secondary keys across databases
Secondary indices in different partitions might diverge
Normalize columns not frequently used
No distributed transactions
NewSQL
New Relational Database Architectures

Examples:
  In-memory databases
  Google Spanner
In-memory Data Model

- equivalent to relational
- short lived transactions
- index look ups (no table scans)
- repeated queries with different parameters
Google Spanner

Globally distributed relational database
Synchronizes with atomic and GPS clocks
Uses Paxos protocol for consensus
Availability or Consistency ?
What is the Cost of an Apology?

Amazon
Airline reservations
Stock Trades
Deposit of a Bank Check
Deleting a photo from Flickr or Facebook
Sometimes the cost is too high

Authentication
  SAML tokens expire

Launching a nuclear weapon
Businesses Apologize Anyway

Vendor drops the last crystal vase
Check bounces
Double-entry bookkeeping requires compensation
  at least 13\textsuperscript{th} century
Eventually make consistent (partition healing)
Software State ≠ State of the World

Software approximates the state of the world
Best guess possible
Could be wrong
Other computers might disagree
How consistent?

Business Decision
What is the cost to get it absolutely right?
What is the cost of lost business?
Computers can remember their guesses
Can replicate to share guesses
May be cheaper to forget, and reconcile later
Design For Eventual Consistency

Decouple unrelated application functionality
Focus on atomic or invariant business operations, not database reads or writes.
No distributed transactions
Asynchronous processing
Eventual Consistency

Different computations might come to different conclusions
Define message based workflows for ultimate reconciliation and replication of results
Not the Whole Story

Databases are not the best integration technology
Object-Relational Mismatch
Certain problems match other data models
Services, not Data, Outlast Implementation
Application or Service Specific Databases
Case Study: Amazon Four Day Outage
Facts

April 21, 2011
One Day of Stabilization, Three Days of Recovery
Problems: EC2, EBS, Relational Database Service
Affected: Quora, Hootsite, Foursquare, Reddit
Unaffected: Netflix, Twillo
Netflix Explicitly Architected For Failure
Although more errors, higher latency, no increase in customer service calls or inability to find or start movies.
Key Architectural Decisions

Stateless Services

Data stored across isolation zones
  Could switch to hot standby

Had Excess Capacity (N + 1)
  Handle large spikes or transient failures

Used relational databases only where needed.
  Could partition data

Degraded Gracefully
Data Architecture

Separate databases:
  User, Accounts, Feedback, Transactions
Split by primary access path
No business logic in database
CPU intensive work in service tier
  Referential Integrity, Joins, Sorting
Avoids deadlock
Degraded Gracefully

Fail Fast, Aggressive Timeouts
Can degrade to lower quality service
  no personalized movie list, still can get list of available movies
Non Critical Features can be removed.
Suggested Reading

"Life Beyond Distributed Transactions: An Apostate's View" by Pat Helland
Conclusions

Scalability means Users, Bandwidth, Geography
Partitioning Changes the Data Model
Service Orientation Changes the Data Model
Design for Eventual Consistency
No need for scalability or service orientation,
    Relational Model works
Unified Data Model makes it hard to meet rapid change.