Evolution of computing systems

- Check out the Living Computers museum in downtown Seattle
- A reproduction of mainframe + minicomputers

Classical Database Application Problems

- Two main types of data management problems that require speed
- We call these “online” processing engines: speed matters

Classical Database Application Problems

<table>
<thead>
<tr>
<th>OLTP (Online Transaction Processing)</th>
<th>OLAP (Online Analytical Processing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction-heavy workloads</td>
<td>Complex query workloads</td>
</tr>
</tbody>
</table>

Classical Database Application Problems

<table>
<thead>
<tr>
<th>OLTP (Online Transaction Processing)</th>
<th>OLAP (Online Analytical Processing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction-heavy workloads</td>
<td>Complex query workloads</td>
</tr>
<tr>
<td>Many simple lookup or single join queries</td>
<td>Many joins, aggregations, etc.</td>
</tr>
</tbody>
</table>
Classical Database Application Problems:

<table>
<thead>
<tr>
<th>OLTP (Online Transaction Processing)</th>
<th>OLAP (Online Analytical Processing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction-heavy workloads</td>
<td>Complex query workloads</td>
</tr>
<tr>
<td>Many simple lookup or single-join queries</td>
<td>Many joins, aggregations, etc.</td>
</tr>
<tr>
<td>Many small updates and inserts</td>
<td>Little to no updates</td>
</tr>
<tr>
<td>Managing consistency is critical</td>
<td>Query optimization and processing is critical</td>
</tr>
</tbody>
</table>

OLTP (Online Transaction Processing):
- Transaction-heavy workloads
- Complex query workloads
- Many simple lookup or single-join queries
- Many small updates and inserts
- Managing consistency is critical

OLAP (Online Analytical Processing):
- Transaction-heavy workloads
- Complex query workloads
- Many joins, aggregations, etc.
- Little to no updates
- Query optimization and processing is critical

3-Tiered Web Architecture:
- How do we architect an OLTP solution?
- Performance issues if we try to scale DB servers
- Scales to a point then DB becomes a bottleneck

RDBMS Replication:
- Create multiple copies of each database partition
- Improves fault tolerance
- Read performance ok
- Write performance suffers!
  - Need to write same value to multiple servers

Distributed RDBMS Consistency Bottleneck:
- RDBMS scaling makes consistency hard
  - Partitioning: Need to coordinate server actions
  - Replication: Need to prevent inconsistent versions
  - ACID is hard to maintain
A hashtag on Twitter for a meetup in San Francisco to discuss systems like Google BigTable, Amazon Dynamo, CouchDB, etc.

How NoSQL Solves Web Scaling

NoSQL in a Nutshell

- NoSQL -> Looser data model
  - Give up built-in OLAP/analysis functionality
  - Give up built-in ACID consistency

NoSQL works for Web 2.0 business models

- No OLAP anyway
- Availability is more important than consistency for Web 2.0
- Facebook:
  - I don’t care if I don’t see every like in real time
  - I care if I can’t send a like
- Amazon:
  - I don’t care if my cart forgot an item
  - I care if I can’t put an item into my cart

Let’s Drop ACID

- RDBMSs have the ACID consistency model
- NoSQL sys. have the BASE consistency model
Let's Drop ACID

- RDBMSs have the ACID consistency model
- NoSQL sys. have the BASE consistency model
- Basically Available
  - Most failures do not cause a complete system outage
- Soft state
  - System is not always write-consistent
- Eventually consistent
  - Data will eventually converge to agreed values

Why the Sacrifice?

Why can’t we have both Consistency and Availability?

CAP Theorem

- Old name: Brewer’s Conjecture
- In a distributed data store, one can only provide two of the following three guarantees:
  - Consistency
    - Every read receives the most recent write or an error
  - Availability
    - Every request must respond with a non-error
  - Partition tolerance
    - Continued operation in presence of dropped or delayed messages

RDBMS vs NoSQL Systems

- Distributed RDBMS
  - Partition tolerance + Consistency
- NoSQL Systems
  - Partition tolerance + Availability
RDBMS vs NoSQL Systems

- Distributed RDBMS
  - Partition tolerance + Consistency
- NoSQL Systems
  - Partition tolerance + Availability

Both must provide partition tolerance by virtue of being distributed systems.

Partition tolerance + Consistency

RDBMS vs NoSQL Systems

Client

DB Node 1

DB Node 2

Write

Done

December 6, 2019

NoSQL
RDBMS vs NoSQL Systems

Partition tolerance + Consistency

December 6, 2019

RDBMS vs NoSQL Systems

Partition tolerance + Consistency

NoSQL

31

V1
DB Node 1

V0
DB Node 2

Client

V1
DB Node 1

V0
DB Node 2

Client

Tries but fails to check consistency of V

RDBMS vs NoSQL Systems

Partition tolerance + Availability

December 6, 2019

NoSQL

35

V0
DB Node 1

V0
DB Node 2

Client

V0
DB Node 1

V0
DB Node 2

Client

Error/Timeout

36
RDBMS vs NoSQL Systems

Partition tolerance + Availability

<table>
<thead>
<tr>
<th>DB Node 1</th>
<th>DB Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>V₁</td>
<td>V₂</td>
</tr>
<tr>
<td>Client</td>
<td>Client</td>
</tr>
</tbody>
</table>

Proof of CAP Theorem

- 2002 original paper (S. Gilbert & N. Lynch)
- More digestible blog post (M. Whittaker)
- Proof by contradiction: Assume we had a system that guaranteed availability, consistency, and partition tolerance...

On A Practical Note

- RDBMSs are intended to be highly consistent
  - Boost availability by sacrificing some consistency
- NoSQL sys. are intended to be highly available
  - Boost consistency by sacrificing some availability
- Most applications OK with some compromise
  - "Return most of data most of the time"
  - DBMS choice has many factors
    - Consistency/Availability requirements
    - Scalability
    - Usability
    - OLAP/analysis requirements
    - ...
Key-Value Store

- Data model:
  - (key, value) pairs
  - Key → string/integer/..., unique for the entire data
  - Value → anything

NoSQL Data Models

- Key-Value Database
  - Data model:
    - (key, value) pairs
    - Key → string/integer/..., unique for the entire data
    - Value → anything

- Wide-Column Store (Extensible Record Store)
  - Key to value pairs
  - “A hash table”
  - “A multidimensional hash table”

- Graph Database
  - Entities and relationships
  - “Unstructured graph”

- Document Store
  - Key to document pairs
  - “Semi-structured file collection”

Key-Value Modeling

Represent all Flights as KV pairs

<table>
<thead>
<tr>
<th>Potential KV pairings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
</tr>
<tr>
<td>Value</td>
</tr>
</tbody>
</table>

12/6/19
### Key-Value Modeling

Represent all Flights as KV pairs

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>Single flight record</td>
</tr>
<tr>
<td>Date</td>
<td>All flight records on that day</td>
</tr>
<tr>
<td>(origin, destination)</td>
<td>All flight records between the cities</td>
</tr>
</tbody>
</table>

### DynamoDB API

- Create, Read, Update, Delete (CRUD) actions
  - Create → PutItem
  - Read → GetItem
  - Update → UpdateItem (Document store functionality)
  - Delete → DeleteItem
- Read consistency
  - Eventually consistent (default, may be stale data)
  - Strongly consistent (gets most recent written data)
- As of December 2018, ACID is “supported”
  - TransactWriteItems
  - TransactGetItems

### NewSQL

- NewSQL addresses the issue of scalable relational data in high-profile data use cases
  - Banking/Billing
  - User authorization
  - Order fulfillment
- First NewSQL system is H-Store in 2007 (commercialized as VoltDB)

### Why Not NewSQL Always

- Some NewSQL systems are like RDBMS on top of NoSQL systems
  - Google implements Spanner on top of BigTable
  - Cockroach Labs implements CockroachDB on top of RocksDB
- NewSQL vs traditional RDBMS: You only need a system that is as powerful as you need
- NewSQL is NoSQL: Relational models are still a poor choice when you have unstructured data

### NewSQL Features

- Simple to manage system
- Relational Transaction processing
- High availability
- Allows unstructured data
What if we want even faster queries than Spark?