Introduction to Data Management

NoSQL

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle
## Classical Database Application Problems

<table>
<thead>
<tr>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Online Transaction Processing)</td>
<td>(Online Analytical Processing)</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>
</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>OLTP (Online Transaction Processing)</td>
<td>OLAP (Online Analytical Processing)</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------</td>
</tr>
<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>
## 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>
<tr>
<td>Flights, banking, etc. (many users)</td>
<td>Business intelligence (few users)</td>
</tr>
</tbody>
</table>
Client-Server Applications
Client-Server Applications

Single server runs the entire database
Client-Server Applications

Single server runs the entire database

Could be:
- Your own computer
- Cloud-hosted DB
Client-Server Applications

Single server runs the entire database

Multiple client applications connect to DB server

Could be:
• Your own computer
• Cloud-hosted DB
**Client-Server Applications**

Single server runs the entire database

Could be:
- Your own computer
- Cloud-hosted DB

Multiple client applications connect to DB server

Could be:
- Query editor
- Java app (lab)
- Analyst app (Tableau)
Client-Server Applications

Single server runs the entire database

Multiple client applications connect to DB server

Could be:
- Query editor
- Java app (lab)
- Analyst app (Tableau)

Could be:
- Your own computer
- Cloud-hosted DB

ODBC/JDBC
Client-Server Applications

Sufficient for OLAP (simple)
Can’t scale connections for OLTP

Single server runs the entire database

Multiple client applications connect to DB server

Could be:
• Query editor
• Java app (lab)
• Analyst app (Tableau)

Could be:
• Your own computer
• Cloud-hosted DB

ODBC/JDBC
A new class of problem emerges in the late 90s and early 2000s (and is still a problem today)

What is Web 2.0?

• Social web (Facebook, Amazon, Instagram, …)
• Startup services need to scale quickly by orders of magnitude (shared-nothing architecture!)
• Exclusively OLTP workloads
3-Tiered Web Architecture

How do we architect an OLTP solution?
3-Tiered Web Architecture

How do we architect an OLTP solution?

Web/App servers (easily replicated for more users)

Browsers allow communication to servers
3-Tiered Web Architecture

How do we architect an OLTP solution?

ODBC/JDBC

HTTP+SSL
How do we architect an OLTP solution?

Scales to a point then DB becomes a bottleneck
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
Database Scaling Techniques

- Scale up via:
  - **Partitioning** (sharding)
  - Replication
RDBMS Partitioning

- Use multiple machines to distribute data
- Write performance ok
- Read performance suffers!
  - Join across servers may have huge network IO cost
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
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.
A hashtag on Twitter for a meetup in San Francisco to discuss systems like Google BigTable, Amazon Dynamo, CouchDB, etc.

Because #NoRDBMS doesn’t have quite the same ring to it

Introduction
This meetup is about “open source, distributed, non-relational databases”. Have you run into limitations with traditional relational databases? Don’t mind trading a query language for scalability? Or perhaps you just like shiny new things to try out? Either way this meetup is for you. Join us in figuring out why these newfangled Dynamo clones and BigTables have become so popular lately. We have gathered presenters from the most interesting projects around to give us all an introduction to the field.

Preliminary schedule
09:45: Doors open
10:00: Intro session (Todd Lipcon, Cloudera)
10:40: Voldemort (Jay Kreps, LinkedIn)
11:20: Short break
11:30: Cassandra (Avinash Lakshman, Facebook)
12:10: Free lunch (sponsored by Last.fm)
13:10: Dynomite (Cliff Moon, Powerset)
13:50: HBase (Ryan Rawson, Stumbleupon)
14:30: Short break
14:40: Hypertable (Doug Judd, Zverts)
15:20: CouchDB (Chris Anderson, couch.io)
16:00: Short break
16:10: Lightning talks
16:40: Panel discussion
17:00: Relocate to Kate O’Brien’s, 579 Howard St. @ 2nd. First round sponsored by Digg

Registration
The event is free but space is limited, please register if you wish to attend.

Location
Magma room, CBS interactive
235 Second Street
San Francisco, CA 94105
How NoSQL Solves Web Scaling

Modern problems require modern solutions
How NoSQL Solves Web Scaling

i give up
NoSQL in a Nutshell

- NoSQL ⇐ Looser data model
  - Give up built-in OLAP/analysis functionality
  - Give up built-in ACID consistency
NoSQL in a Nutshell

- 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

**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**
Partition tolerance + **Consistency**

**Diagram:**
- **Client** connected to **DB Node 1** and **DB Node 2**.
- Both DB nodes have the version `V_0`.

**Title:** RDBMS vs NoSQL Systems
Partition tolerance + **Consistency**

- **DB Node 1**
  - $V_0$
  - Write $V_1$

- **Client**
  - $V_0$

- **DB Node 2**
  - $V_0$
RDBMS vs NoSQL Systems

Partition tolerance + Consistency
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**

Diagram:

- **DB Node 1**: $V_1$
- **DB Node 2**: $V_0$
- **Client**

Connections:
- $V_1$ to Client
- $V_0$ to Client
- Client to Done
Partition tolerance + **Consistency**
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**

Diagram showing the relationship between DB Node 1 and DB Node 2 with a client requesting data (V1 or V0).
Partition tolerance + **Consistency**

- DB Node 1
  - $V_1$
- DB Node 2
  - $V_0$
- Client

Tries but fails to check consistency of $V$
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**
RDBMS vs NoSQL Systems

Partition tolerance + **Consistency**

- **DB Node 1**: $V_1$
- **DB Node 2**: $V_0$
- **Client**

Error/Timeout

Consistent!
But not available.
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**
Partition tolerance + **Availability**

RDBMS vs NoSQL Systems

- DB Node 1
- DB Node 2
- Client

$V_0$
Partition tolerance + **Availability**

- **DB Node 1**
  - $V_0$
  - Write $V_1$

- **Client**

- **DB Node 2**
  - $V_0$
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**

![Diagram](image)
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**

[Diagram: Two DB Nodes (V₁ and V₀) connected to a Client with a Done arrow.]
Partition tolerance + Availability

DB Node 1

\( V_1 \)

DB Node 2

\( V_0 \)

Client
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**

![Diagram showing DB Node 1 with V₁, DB Node 2 with V₀, and a client connected by lines indicating Read V.](image-url)
Partition tolerance + **Availability**

- **DB Node 1**: $V_1$
- **DB Node 2**: $V_0$
- **Client**

Tries but fails to check consistency of $V$
Partition tolerance + **Availability**
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**

Available!
But not consistent.
RDBMS vs NoSQL Systems

Partition tolerance + **Availability**

**IMPORTANT:**
These are only cases when the network infrastructure goes down. Usually, nodes should be able to check on other nodes.
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...
Proof of CAP Theorem

Partition tolerance + Consistency + Availability?

DB Node 1

$V_1$

Client

DB Node 2

$V_0$

Error/Timeout

Violates availability!
Proof of CAP Theorem

Partition tolerance + Availability + Consistency?

DB Node 1

$V_1$

Client

DB Node 2

$V_0$

$V_0$

Violates consistency!
On A Practical Note

- RDBMSs are intended to be highly consistent
  - Boost availability by sacrificing some consistency
- NoSQL systems 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
    - ...

February 26, 2020
NoSQL Data Models

Key-Value Database

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>AAA,BBB,CCC</td>
</tr>
<tr>
<td>K2</td>
<td>AAA,BBB</td>
</tr>
<tr>
<td>K3</td>
<td>AAA,DDD</td>
</tr>
<tr>
<td>K4</td>
<td>AAA,2,01/01/2015</td>
</tr>
<tr>
<td>K5</td>
<td>3,ZZZ,5623</td>
</tr>
</tbody>
</table>

Wide-Column Store (Extensible Record Store)

Graph Database

Document Store

XML

```xml
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```json
{
  "empinfo": {
    "employees": [
      {
        "name": "James Kirk",
        "age": 40,
      },
      {
        "name": "Jean-Luc Picard",
        "age": 45,
      },
      {
        "name": "Wesley Crusher",
        "age": 27,
      }
    ]
  }
}
```
Coming up

- Deeper look at using NoSQL stores
  - First we’ll look at key-value stores
  - Then we’ll look at semi-structured data
  - Then the query language SQL++ for AsterixDB