Programming at Scale: Consistency

cs378h
Today

Questions?

Administrivia

Agenda:
• Concurrency & Consistency at Scale
Data-Parallel Computation Systems

Execution
- Application
- Storage
- Language
  - SQL
  - Sawzall
  - Hive
  - Pig, Hive
  - LINQ, SQL
  - DryadLINQ
  - ≈SQL

Parallel Databases
- SQL
- BigTable
- S3
- SQL Server

```
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```

```
CREATE EXTERNAL TABLE lines(line string)
LOAD DATA INPATH 'books' OVERWRITE INTO TABLE lines;

-- create a virtual view that splits the lines
SELECT word, count(*) FROM lines
  LATERAL VIEW explode(split(text, ' ')) lTable as word
  GROUP BY word;
```

```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```
(Yet) Another Framework

**Still not a perfect framework**

**Cons:**
- Many dimensions contain sub-dimensions
- Many concerns fundamentally coupled
- Dimensions are often un- or partially-ordered

**Pros:**
- **Makes important concerns explicit**
- Cleanly taxonomizes most modern systems

---

**Data Model**

- Key Value Stores
- Document Stores
- Wide-column Stores
- Eventual: BASE

**Implementation Techniques**

- Replication
- Storage
- Query Support
- Sharding / Partitioning

**Pros:**
- Basically Available
- Soft State
- Eventually Consistent
- Atomicity
- Consistency
- Isolation
- Durability

**Cons:**
- Many dimensions contain sub-dimensions
- Many concerns fundamentally coupled
- Dimensions are often un- or partially-ordered

• Shared-Disk
• Range-Sharding
• Primary-Backup
• Commit-Consensus
• Logging
• Secondary Indexing
• Query Planning
• Materialized Views
• Analytics
Consistency

How to keep data in sync?

- Partitioning → single row spread over multiple machines
- Redundancy → single datum spread over multiple machines
Consistency: the core problem

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

How should we implement write? How to implement read?
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:
  
  1. **Consistency:**
     • all nodes see same data at any time
     • or reads return latest written value by any client
  
  2. **Availability:**
     • system allows operations all the time,
     • and operations return quickly
  
  3. **Partition-tolerance:**
     • system continues to work in spite of network partitions

Why is this “theorem” true?

if(partition) { keep going } → !consistent && available
if(partition) { stop } → consistent && !available
CAP Implications

- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability.

**PACELC:**

```java
if(partition) {
    choose A or C
} else {
    choose latency or consistency
}
```

CAP is flawed
Consistency Spectrum

Faster reads and writes

More consistency

Eventual → Strong (e.g., Sequential)
Spectrum Ends: Eventual Consistency

• Eventual Consistency
  • If writes to a key stop, all replicas of key will converge
  • Originally from Amazon’s Dynamo and LinkedIn’s Voldemort systems

Faster reads and writes

More consistency

Strong (e.g., Sequential)
Spectrum Ends: Strong Consistency

• **Strict:**
  • Absolute time ordering of all shared accesses, reads always return last write

• **Linearizability:**
  • Each operation is visible (or available) to all other clients in real-time order

• **Sequential Consistency** [Lamport]:
  • "... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.
  • After the fact, find a “reasonable” ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.

• **ACID** properties
Many Many Consistency Models

- Amazon S3 – **eventual** consistency
- Amazon Simple DB – **eventual** or strong
- Google App Engine – **strong** or eventual
- Yahoo! PNUTS – **eventual** or strong
- Windows Azure Storage – **strong** (or eventual)
- Cassandra – **eventual** or strong (if R+W > N)
- ...

**Question:** How to choose what to use or support?
Some Consistency Guarantees

- **Strong Consistency**: See all previous writes.
- **Eventual Consistency**: See subset of previous writes.
- **Consistent Prefix**: See initial sequence of writes.
- **Bounded Staleness**: See all "old" writes.
- **Monotonic Reads**: See increasing subset of writes.
- **Read My Writes**: See all writes performed by reader.

**Diagram:**

- **Strong**
- **Prefix**
- **Bounded**
- **Monotonic**
- **RMW**
- **Eventual**

**Metric =** set of allowable read results
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
                score = Read ("home");
                Write ("home", score + 1);
            }
        }
        hScore = Read("home");
        vScore = Read("visit");
        if (hScore == vScore)
            play-overtime
    }
}
Official Scorekeeper

score = \textbf{Read} ("visitors");
\textbf{Write} ("visitors", score + 1);

Desired consistency?
\textbf{Strong} = Read My Writes!
Referee

vScore = \textbf{Read} ("visitors");
hScore = \textbf{Read} ("home");
if vScore == hScore
play-overtime

Desired consistency?

\textbf{Strong consistency}

<table>
<thead>
<tr>
<th>Strong Consistency</th>
<th>See all previous writes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Radio Reporter

do {
  BeginTx();
  vScore = Read (“visitors”);
  hScore = Read (“home”);
  EndTx();
  report vScore and hScore;
  sleep (30 minutes);
}

Desired consistency?

Consistent Prefix
Monotonic Reads
or Bounded Staleness

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = \textsf{Read} ("visitors");
hScore = \textsf{Read} ("home");
write article;

Desired consistency?
 Eventual
 Bounded Staleness

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Desired consistency?

**Strong Consistency** (1st read)

**Read My Writes** (2nd read)

```
Wait for end of game;
score = Read (“home”);
stat = Read (“season-goals”);
Write (“season-goals”, stat + score);
```
Stat Watcher

\[
\begin{align*}
d &\{ \\
&\quad \text{stat} = \text{Read} \text{ (“season-goals”)}; \\
&\quad \text{discuss stats with friends;} \\
&\quad \text{sleep (1 day)}; \\
&\}
\end{align*}
\]

Desired consistency?

**Eventual Consistency**

<table>
<thead>
<tr>
<th>Strong Consistency</th>
<th>See all previous writes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
**Official scorekeeper:**

score = \texttt{Read} (“visitors”);
Write (“visitors”, score + 1);

**Statistician:**

Wait for end of game;
score = \texttt{Read} (“home”);
stat = \texttt{Read} (“season-goals”);
Write (“season-goals”, stat + score);

**Referee:**

do {
    vScore = \texttt{Read} (“visitors”);
    hScore = \texttt{Read} (“home”);
    report vScore and hScore;
    sleep (30 minutes);
}

**Sportswriter:**

While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = \texttt{Read} (“visitors”);
hScore = \texttt{Read} (“home”);
write article;

**Radio reporter:**

do {
    vScore = \texttt{Read} (“visitors”);
    hScore = \texttt{Read} (“home”);
    report vScore and hScore;
    sleep (30 minutes);
}

**Stat watcher:**

stat = \texttt{Read} (“season-runs”);
discuss stats with friends;

**Strong Consistency**

- Strong Consistency
- Monotonic Reads
- Consistent Prefix

**Bounded Staleness**

- Read My Writes
- Bounded Staleness

**Eventual Consistency**

- Read My Writes
- Eventual Consistency
Sequential Consistency

- weaker than strict/strong consistency
  - All operations are executed in some sequential order
  - Each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order

Why is this weaker than strict/strong?

Nothing is said about “most recent write”
Linearizability

• Assumes sequential consistency and
  • If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency
  • Difference between linearizability and serializability?
    • Granularity: reads/writes versus transactions

• Example:
  • Stay tuned...relevant for lock free data structures
  • Importantly: a property of concurrent objects
Causal consistency

- Causally related writes seen by all processes in same order.
- Causally?
- Concurrent writes may be seen in different orders on different machines

Causal:
If a write produces a value that causes another write, they are causally related.

\[
X = 1 \\
\text{if}(X > 0) \{ \\
\quad Y = 1 \\
\}
\]

Causal consistency \(\rightarrow\) all see \(X=1, Y=1\) in same order

<table>
<thead>
<tr>
<th>P1: (W(x)a)</th>
<th>P2: (R(x)a) (W(x)b)</th>
<th>P3: (R(x)b) (R(x)a)</th>
<th>P4: (R(x)a) (R(x)b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not permitted

Permitted
# Consistency models summary

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict</td>
<td>Absolute time ordering of all shared accesses matters.</td>
</tr>
<tr>
<td>Linearizability</td>
<td>All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp</td>
</tr>
<tr>
<td>Sequential</td>
<td>All processes see all shared accesses in the same order. Accesses are not ordered in time</td>
</tr>
<tr>
<td>Causal</td>
<td>All processes see causally-related shared accesses in the same order.</td>
</tr>
<tr>
<td>FIFO</td>
<td>All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order</td>
</tr>
</tbody>
</table>

(a)  

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak</td>
<td>Shared data can be counted on to be consistent only after a synchronization is done</td>
</tr>
<tr>
<td>Release</td>
<td>Shared data are made consistent when a critical region is exited</td>
</tr>
<tr>
<td>Entry</td>
<td>Shared data pertaining to a critical region are made consistent when a critical region is entered.</td>
</tr>
</tbody>
</table>

(b)