Presentation by
CS 655 – Advanced Topics in Distributed Systems
Computer Science Department
Colorado State University

CAP Theorem

“The CAP theorem states that any networked shared-data system can have at most two of three desirable properties:
+ consistency (C) equivalent to having a single up-to-date copy of the data;
+ high availability (A) of that data (for updates); and
+ tolerance to network partitions (P).” (Brewer, 2012)


Partition Detection

When communication retries exceed a specified time bound.
+ Partition information is not global.
+ Time bound is specified based on required response time.
+ If small, slow network can be mistaken for partition.

### Partition Management

- **Partition mode:**
  - Remain available, but with limited operations.
  - Delay operation (become unavailable) without user's knowledge.

- **Partition recovery:**
  - Restore consistency and compensate for mistakes.

![Partition Diagram](image1)

### Other Reasons to Reduce Consistency

- High latency, especially in wide-area systems. (e.g. PNUTS)
- Concurrent access (throughput), especially with long transactions.

**Conclusion:**
- Increase both availability and consistency
- Largely depends on the application

### DB on S3

- Cloud computing can provide high levels of scalability, availability, and throughput to distributed systems.
- Regular DB systems with high consistency levels cannot be deployed in such settings.
- The authors of this paper built a DB system in the cloud settings for specific target applications: web-based applications that highly benefit the three features provided by cloud computing and can settle for a weakly consistent DB.

### S3

- S3 is designed to store large objects that are rarely updated.
- The authors store the DB records which are small objects that are frequently updated.

- To minimize the cost of using S3 when retrieving or updating the records:
  - Multiple records are packed in a single object (page).
  - Eventual consistency is used.

### Storage Organization

- **Bucket = Collection (Table)**
  - **Root Node**
  - **Intermediate Node**
  - **Leaf Node**
  - **Page**

- **B-Tree Index-Organized Table**

---

10/29/13
System Architecture

- Functions provided:
  - Create, delete, read, and update records.
  - Scan collection.
  - Commit and abort.

Buffer pool of pages:
- Read pages from S3 (TTL).
- Commit and abort.
- Update pages.

Update Protocol

Amazon SQS

Update Protocol

Transaction Atomicity

- “All or none” of the updates compromising a transaction should be applied.
- Each client has a local queue of commit log records (ATOMIC queue).
- Additional information: commit/transaction id.
- Log records are sent to the PU queue, then deleted from ATOMIC queue.
- In the case of client failure, it will check the ATOMIC queue when it restarts.
- What if the client does not restart? They ensured the “all” part, but not the “none” part.

Client-Side Consistency Models

- Monotonic Reads
  - If a has been read, any successive read must result in the same value or a newer value.
  - Clients maintain information about past cached pages timestamps.

- Monotonic Writes
  - A write on x is completed before any successive writes.
  - Preserve order per client using counters.
  - A (client id, counter value) is added to the log record, and in the header of the updated page reflecting the client's last update.
  - During checkpoint, out-of-order logs are detected and unprocessed.

Client-Side Consistency Models

- Read your writes
  - A local write on x will be seen by a successive read.
  - Local updated copy.
  - What if the pending update has not been performed? (confusing)

- Write follows read
  - A write on a previously read x, will be written on the same version or a newer one.
  - Pending updates.

Who performs the updates? When?

- Not recommended: Watchdogs and owners.
- They are not suitable because they waste resources and are vulnerable to failure. (could use scheduler)
- No need for locks.
- Readers and Writers (Clients).
- Each page has a header, and each B-tree’s root node metadata contain last update timestamp.
- When logging or reading check timestamp.
- Trade-off between cost and fast updates in setting time threshold.

Client-Side Consistency Models

- Monotonic Reads
  - If x has been read, any successive read must result in the same value or a newer value.
  - Clients maintain information about past cached pages timestamps.

- Monotonic Writes
  - A write on x is completed before any successive writes.
  - Preserve order per client using counters.
  - A (client id, counter value) is added to the log record, and in the header of the updated page reflecting the client's last update.
  - During checkpoint, out-of-order logs are detected and unprocessed.
Query Processor?

- A layer above the record manager.
- Problems?

SQL Query Processing

- A variant of MapReduce with BigTable as a data structure.
- Select-group-aggregate applications.

Example

Entity-Relationship Diagram

SQL Query:

```
SELECT p.patrick, AVG(p.age) AS average FROM Patron p
WHERE p.weight > 150
GROUP BY p.pattyeColor
```

Proposed Changes

- Store all data in a BigTable.
- Launch a map instance for each tablet.
- Code is small while tablets are large → more efficient distribution.
- Tablet is input instead of individual text lines to map functions.
  - Ability to skip rows with same id (data organization is explained next).
- Use of multiple reduce phases
  - With the ability to bypass the system’s sort operation.
Data Organization

- BigTable data is addressed by row id and column name.
- Each tuple is mapped to a group of rows (for each attribute) with the same row id.
- Why? A row can have multiple columns.
- Tablet size is 100 MB
- Row groups are not split.
- Distributed to all nodes.
- Exploit the entity-relationship diagram in constructing the records (clustering).

Query Evaluation

- Example:
  - All Seattle libraries that have loaned books in poor condition to blue-eyed patron after January 1, 2008.
- SQL Query:
  ```sql
  select distinct L.Name 
  from library L join book B 
  on L.Name = B.Library 
  where L.Name = 'Seattle' 
  and B.Condition = 'poor' 
  and B.Patent = 'blue' 
  and B.Loan > '2008-01-01' 
  and P.Patent = 'blue' 
  SQL query/usage in Johnson, (2009)
  ```
Query Processing Algorithm

+ Can be used to process a wide range of SQL queries (select-group-aggregate).
+ Scalable for large DBs.
+ Possible Problem: flow of large fraction of the DB
  ➔ high network traffic ➔ slow system sorts.
+ The author's conclusion: SQL evaluation can be done in the clouds, but its performance is still an open question.

PNUTS

+ Designed for Yahoo's web applications.
+ Geographically distributed database system.
+ Requirements: high availability, scalability, and response time.
+ Per-record consistency (relaxed consistency).

Per-Record Consistency vs. Eventual Consistency

+ In eventual consistency, updates will eventually be applied to all replicas but they can be out of order.
+ **Too weak** for some applications:
  + Example: Photo sharing application (photos with access privileges).

\[(1)\text{ Change access}\quad (2)\text{ Upload Photos}\quad \cdots\quad (1)\text{ Change access}\quad (2)\text{ Upload Photos}\]

+ In Per-record consistency, updates will eventually be applied to all replicas but in the same order.

Data Organization and Queries

+ Simple relational data model with flexible schemas.
+ Tables are horizontally partitioned in **tablets**, and distributed to many servers.
+ PNUTS query language only supports **single tables** (limitation).
+ Read and write of single records or small group of records.
+ Predicates can be used for evaluating scans at the servers.
+ Multiget operation: get records from one or more tables, using a set of primary keys and predicates.

Consistency Model

+ **Per-record timeline consistency**
  + Updates to a record are applied to all the record's replicas in the same order.
  + All replicas move forward in time.

\[
\begin{array}{cccc}
\text{Insert} & \text{Time} & \text{Update} & \text{Time} \\
1 & 10 & 2 & 20 \\
2 & 20 & 3 & 30 \\
3 & 30 & 4 & 40 \\
\end{array}
\]

+ **Implementation**:
  + Assign a master replica for each record.
  + All updates to a record are forwarded to the master replica of this record.
  + The master assignment is **adaptive** to load change.
  + The replica that receives most of the write requests becomes the master.
API Calls ➔ Different Consistency Levels

- Write
  - Returns the version number of the record
- Read-­‐any
- Read-­‐critical (required_­version)
- Read-­‐latest
- Test-­‐and-­‐set-­‐write (required_­version)
  - Can be used when you update a read value such as increments.
  - Validation is the applications responsibility.

May have higher latency

Update Protocol

- Use Yahoo Message Broker (YMB).
- A topic-­‐based publish/subscribe system
- An update is committed by publishing it in YMB, then it is asynchronously propagated to the replicas.
- YMB guarantees message delivery to all topic subscribers.
- YMB provides partial ordering of messages.
  - Order from one publisher is preserved, but order between different publishers is not.
  - So they use per-­‐record mastership: updates are published by only the record's master.
  - All updates are forwarded to the master.

Record-­‐level Mastering

- A table can have records mastered in different clusters (regions).
- Write locality on a per-­‐record basis is their observation in their web application that promoted this mechanism.
  - Example: 85% write locality in 1 week of updates in 9.8 million Yahoo's users records.
  - Exploit the high locality for performance improvement.
- Each record has a metadata containing:
  - Master id. (for forwarding updates)
  - Origin of last N updates. (for master-­‐reassignment)

Tablet Mastership

- When inserting new records, primary keys constraint must be enforced.
- Assign a tablet master to order, accept, or reject insert operations.
- All insert operations are forwarded to the tablet master.

Processing Multi-­‐record Requests

- Scatter-­‐gather engine:
  - Component in the router.
  - Split it to parallel multiple single-­‐record requests or single-­‐tablet scans.
  - Then gathers the results and sends them back.
- Server side processing is chosen:
  - To maintain one connection per client.
  - To optimize it using data locality.
Amazon SimpleDB

Opposite to S3, SimpleDB is designed for small data.
- NoSQL data store with flexible schema.
  - Organized in domains (tables), each is a group of items (records), and each item has multiple attribute-value pairs.
- Select queries for single domains (limitation).
- Automatically creates indexes for the data for quick queries. (S3 stores raw data).

Consistency Level

- Geographically distributed replicas.
- A write is successful when all replicas are updated.
  - In concurrent (overlapped) writes:
    + Read result is unknown.
    + Depends on the order where SimpleDB received and processed the requests (FIFS).
    + Whenever you have multiple clients writing to the same items, implement some concurrently control mechanisms, such as timestamp ordering, to ensure you are getting the data you want.
- 2 read options:
  + Eventually consistent read.
  + Consistent read (may have higher latency).
- Conditional put/delete.

Examples

- Scenario 1: no R/W or W/W overlap
- Scenario 2: R/W overlap
- Scenario 3: W/W overlap

SimpleDB Combined with S3

- S3 storage is cheaper.
  - Different physical storage: S3 uses disks optimized for inexpensive storage of large objects, while SimpleDB uses disks optimized for fast data access.
  - SimpleDB is faster with query capabilities.
  - Free data transfer within the AWS environment.
- To reduce cost, SimpleDB can be combined with S3:
  + Store large objects in S3 and store pointers to them with metadata in SimpleDB.

SimpleDB Size Limitation

- A domain can grow only to 10GB.
- Data can be split into multiple domains (not automatic).
- Max 250 domains (form filled for requesting more).
- Limited request capacity (under 25 writes/second).
- DynamoDB has no storage size or request capacity limits, and provides automatic partitioning.
Amazon DynamoDB

Amazon DynamoDB is similar to SimpleDB, but without the size limitations, and with automatic partitioning.

- Faster: uses Solid State Drives (SSD).
- More expensive storage price, but cheaper request costs.
- Not suitable for large data that is rarely accessed. (S3 was recommended for this case)
- Provisioned throughput: user specifies the request capacity (read and write) and can modify it later.
- Uses horizontal partitioning for scaling up.

Relation to Amazon Dynamo?

- The use of hashing in partitioning the tables.
- DynamoDB uses the hash of the primary key, or a composite hash-range for composite keys.
- Creates a hash index for fast record access.

Relational Queries

- SimpleDB and DynamoDB are NoSQL database services.
- Better scalability but no complex operations such as joins.
- Amazon RDS provides “fully featured” relational database.
- DynamoDB can be integrated with Amazon Elastic MapReduce (EMR) for complex data analysis.

Discussion and Comparison

- S3, SimpleDB, DynamoDB, and PNUTS.
- All systems are trying to achieve high scalability, availability, and throughput.
- Trade-off between consistency and throughput.
- SimpleDB, DynamoDB, and PNUTS provided only single-table queries.
Discussion and Comparison

- S3
  - May be good with large objects that are rarely changed.
  - Enforcing frequently updated small objects may not be a good idea (2 layers of eventual consistency).
  - Needs application layer for DB implementation.
- SimpleDB
  - Better DB support: small objects with fast access and selection queries.
  - Only suitable for small systems.
- DynamoDB
  - Maybe better business solution for larger systems.
- PNUTS
  - Stronger consistency by ensuring order per record.

Regarding Amazon Solutions

<table>
<thead>
<tr>
<th></th>
<th>S3</th>
<th>SimpleDB</th>
<th>DynamoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheaper</td>
<td></td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Faster</td>
<td>o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Less management requirement</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Higher consistency</td>
<td></td>
<td></td>
<td>o</td>
</tr>
</tbody>
</table>

References

5. Amazon SimpleDB
   http://aws.amazon.com/simplesdb/
6. Amazon DynamoDB
   http://aws.amazon.com/dynamodb/