Eventually Consistent

Werner Vogels

ACM Queue: October 2008
Communications of ACM: January 2009
Beneath Infrastructure Services

• Infrastructure Services
  – Provide resources for huge computing platforms
  – Amazon S3 (Simple Storage Service), EC2 (Elastic Compute Cloud), SimpleDB etc.
  – Security, scalability, performance etc.

• Beneath
  – Massive Distributed System
    • Worldwide scale
Distributed Systems - Issues

• The issue first sprung up in databases in late 1970s
  – “Notes on Distributed Databases" by Bruce Lindsay
  – ACID (atomicity, consistency, isolation, durability) in DBMS

• Mid 1990 large internet systems became popular
  – Here availability was perhaps the most important property.
CAP Theorem

- Eric Brewer, UC Berkeley.
- Three properties in a shared system:
  - Data Consistency
  - System availability
  - Tolerance to network partition

Only Two can be achieved
CAP Theorem

- Network partition natural in large distributed system.
- Then,

  Data Consistency Vs. Availability

- No much flexibility in Availability
  - Data is either available or not available.
Consistency

• So the focus of this article is mainly on consistency

• Two points of view:
  – The client point of view
    • How do they observe the data updates
  – The server point of view
    • How updates flow through the system
    • What guarantee can the system give.
Client Side Consistency

• Three components involved:
  – Storage system
  – Process A
  – Process B & C, independent of A.

• Types of consistency:
  – **Strong Consistency**
    • After update any access would return only updated value.
  – **Weak Consistency**
    • No guarantee about updated value to be returned.
    • Inconsistency Window
      – period between actual update and returned update value
Client Side Consistency

• Types (cont):
  – Eventual Consistency
    • Specific case of weak consistency
    • If no new updates and no failures, the max size of inconsistency window can be calculated.
      – Communication delay, no. of replicas, system load etc.
    • Eg. DNS
      – Updates to a name are distributed in configured pattern
      – Time controlled caches
Client Side Consistency

• Variations of Eventual Consistency:
  – Causal consistency
    • A informs B. C has to wait.
  – Read-Your-Write Consistency
    • After A writes, it see only the updated value.
  – Session Consistency
    • Read-Your-Write over a session period.
  – Monotonic Read Consistency
    • If a process has a value, then no subsequent access return any previous value
  – Monotonic Write Consistency
    • system guarantees to serialize the writes by the same process
Server Side Consistency

- **Definitions:**
  - N = no. of nodes that store replicas
  - W = no. of replicas that need to acknowledge the receipt of update before update completes
  - R = no. of replicas contacted during read.

- **Strong Consistency**
  - W+R>N – write and read set overlap.
  - Primary backup RDBMS scenario – N=2, W=2, R=1
  - Synchronous replication.
  - Typically
    - R=1 and W=N (for optimized read)
    - W=1 and R-N (for optimized fast write).
Server Side Consistency

• Eventual Consistency
  – $R+W \leq N$
  – Asynchronous replication
    • Typically $W<N$
    • Lazy update for remaining nodes.
  – Can add versions to the write
Conclusion

• Amazon Dynamo
  • A key value storage system
  • Amazon's e-commerce platform and web services
  • Allows applications service owner to make trade-offs between consistency, availability and performance

• Takeaway
  • Eventually consistent – the way
    – Monotonic read with session consistency
    – Monotonic read and read-your-write
  • Adding versions to writes
Dremel: Interactive Analysis of Web Scale Datasets

Sergey Melnik et al.

VLDB 2010
Motivation

• Data in web and scientific computing are non-relational
• Data structures in programming languages, messages in distributed systems usually have nested representation
• Google's data processing are mostly on nested structured data

Nested Data
Dremel - Introduction

• A scalable, interactive, ad-hoc query system for analysis of read only nested data.
• Aggregation queries over trillion-row tables in seconds
• Interactive analysis of large datasets shared over cluster of commodity machines
• Can be used in conjunction with MR
Concepts

• Columnar representation
• Striped storage
• High level SQL Language
• Serving trees
  • Like in distributed search engines
Nested Data

\[ \tau = \text{dom} \mid (A_1 : \tau[*|?], \ldots, A_n : \tau[*|?]) \]

- **dom** - integer, floating point, numbers, string......
- \( A_i \) - multiplicity label
- * - Repeated field
- ? - optional field - may be missing from the record
Columnar vs. Row-wise Storage

If you want to access a particular value alone
– in case of row-wise you'll need to read the whole row.
– in case of columnar you can read a particular value alone. To read the values of A.B.C, we don't need to read A.B.D or A.B.E etc.

Each of country, code, url etc. will form a table respectively, coded red, green and blue in colour respectively.
Repetition and Definition Levels

• Repetition level
  • Level, from the root, at which repetition happens.

• Definition level
  • Level, from root, at which the value is defined
  • Useful esp. in case of NULL/Empty values.
  • data at leaf level has info about it's parent field

<table>
<thead>
<tr>
<th>value</th>
<th>r</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>en-us</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>en</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>en-gb</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Splitting into Columns

- Columns striped with the repetition and definition values

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>value</td>
<td>value</td>
<td>value</td>
<td>value</td>
<td>value</td>
</tr>
<tr>
<td>10</td>
<td><a href="http://A">http://A</a></td>
<td>20</td>
<td>NULL</td>
<td>en-us</td>
<td>us</td>
</tr>
<tr>
<td>20</td>
<td><a href="http://B">http://B</a></td>
<td>40</td>
<td>NULL</td>
<td>en</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
<td>60</td>
<td>NULL</td>
<td>NULL</td>
<td>gb</td>
</tr>
<tr>
<td></td>
<td><a href="http://C">http://C</a></td>
<td>80</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Record Assembly

• Finite State Machines ( FSM )
• The values and levels are read accordingly appended to the output records.
• Transitions are labeled with the repetition level
  – Repetition level value decides what to read next
Query Language

- Based on SQL
- Take as input one or multiple nested tables
- Produce a nested table and output schema

```sql
SELECT DocId AS Id,
      COUNT(Name.Language.Code) WITHIN Name AS Cnt,
      Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;
```

**Example Output**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name</td>
<td>Cnt: 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Query Execution

- Many Dremel queries are one-pass aggressions.
  - They've discussed only those in the paper
- Joins, indexing and updates not discussed
- Tree Architecture (Serving Tree)
  - Multi level serving tree to execute query
  - Root server receives the query
  - Breaks them and routes the queries to next levels of server.
  - Leaf servers actually communicate with the storage layer
SELECT A, COUNT(B) FROM T GROUP BY A

When the root server receives the above query, it determines all tablets, i.e., horizontal partitions of the table, that comprise T and rewrites the query as follows:

SELECT A, SUM(c) FROM (R_1^1 UNION ALL ... R_n^1) GROUP BY A

Tables R_1^1, ..., R_n^1 are the results of queries sent to the nodes 1, ..., n at level 1 of the serving tree:

R_i^1 = SELECT A, COUNT(B) AS c FROM T_i^1 GROUP BY A

T_i^1 is a disjoint partition of tablets in T processed by server i at level 1. Each serving level performs a similar rewriting. Ultimately, the serving reach the leaves, which run the tablets in T in...
Query Dispatcher

• Dremel is multiuser system
• Several queries executed simultaneously.
• Query dispatchers schedules the queries based on priorities
  – Load balancing.
  – Fault tolerance
    • when one server becomes slow or a tablet replica becomes unreachable.