Yesquel

Scalable SQL Storage for Web Applications

Reece Doyle & Oscar Robinson
Database Systems

- Modern Web apps involve storing lots of data
- Two options: NoSQL vs SQL

**SQL**: Table based storage, standard query interface

**NoSQL**: Not table based - many methods:
- key-value
- document storage
- object based

all with different interfaces!
What’s wrong with SQL?

Popular SQL storage engines cannot be distributed amongst multiple servers like other web application components.
Scaling SQL?

- Replicas for balancing reads
- Manual techniques:
  - Sharding - e.g keys A-K on one server, L-Z on other
  - Caching
  - Denormalisation

This is all hard work for the developer!

Most now just use NoSQL instead: **sacrificing** the useful query functionality of SQL

Can SQL give NoSQL-like performance and scalability for Web applications while retaining the SQL feature set?
NoSQL System

- Highly scalable
- Application complexity
- Specialised interface
- Little functionality

Yesquel Goal

- Broad feature set
- Well-known standard interface

SQL System

- Poor scalability
Yesquel

A new storage system to give NoSQL-like performance with SQL’s rich queries

- Introduces a new scalable distributed data structure based on Distributed Balanced Trees
- Uses a novel caching method
- Multiple query processors

 Bounds:

- Only meets these goals with Web applications
- Distributed within datacentre not across WAN
Structured Query Language Primer: Useful Features

Designed for managing data in relational database management systems

**SQL Table 1**

<table>
<thead>
<tr>
<th>Student ID</th>
<th>First Name</th>
<th>Last Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>53792</td>
<td>Joe</td>
<td>Bloggs</td>
</tr>
</tbody>
</table>

Primary key: Student ID

Secondary index: Last Name

**Ordered Map 1**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>(John, Doe)</td>
</tr>
<tr>
<td>53792</td>
<td>(Joe, Bloggs)</td>
</tr>
</tbody>
</table>

**Ordered Map 2**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloggs</td>
<td>53792</td>
</tr>
<tr>
<td>Doe</td>
<td>12345</td>
</tr>
</tbody>
</table>
Structured Query Language Primer: Useful Features

Designed for managing data in relational database management systems

<table>
<thead>
<tr>
<th>Student ID</th>
<th>First Name</th>
<th>Last Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>53792</td>
<td>Joe</td>
<td>Bloggs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Degree Programme</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>MEng Computer Science</td>
</tr>
<tr>
<td>53792</td>
<td>BA Mathematics</td>
</tr>
</tbody>
</table>

Table 1 ⋈ Table 2

<table>
<thead>
<tr>
<th>Student ID</th>
<th>First Name</th>
<th>Last Name</th>
<th>Degree Programme</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>John</td>
<td>Doe</td>
<td>MEng Computer Science</td>
</tr>
<tr>
<td>53792</td>
<td>Joe</td>
<td>Bloggs</td>
<td>BA Mathematics</td>
</tr>
</tbody>
</table>
Standard SQL Architecture

Load Balancer

Application Server
Application Server
Application Server

Database Server
Query Processor
Storage Engine
Yesquel Architecture
DBT - Distributed Balanced Tree

- Query Processor results in requests to storage engine - manipulating values by key

Balanced Tree is good way of storing ranges of keys:

Have leaves on different physical servers
Accessing a key

Query processor has to get/put value for a key:
**Naive approach** - Search tree from root

```
5
128.1.2.3

2,3,4,5
128.1.2.4

6,7,8
128.1.2.5

QUERY PROCESSOR
```
Accessing a key

Query processor has to get/put value for a key:

**Naive approach** - Search tree from root
Accessing a key

Query processor has to get/put value for a key:

Naive approach - Search tree from root
Accessing a key

Query processor has to get/put value for a key:

**Naive approach** - Search tree from root
Accessing a key

Query processor has to get/put value for a key:

**Naive approach** - Search tree from root

![Diagram showing query processor accessing a key through a search tree.](image-url)
Accessing a key

Query processor has to get/put value for a key:

**Naive approach** - Search tree from root

Every query goes through root node!

**QUERY PROCESSOR**
- Get Key 3
- Key 1-5 at 128.1.2.4
- Get Key 3
- Returns Value at Key 3
Accessing a key

Query processor has to get/put value for a key:

**Naive approach** - Search tree from root

Every query goes through root node!

Communication overhead!

---

QUERY PROCESSOR
- Get Key 3
- Key 1-5 at 128.1.2.4
- Get Key 3
- Returns Value at Key 3
Caching

What if we just cache the inner nodes, their ranges don’t change often

What do we do when the cache goes stale? Just search from root with RPCs?
Back Down Search

QUERY PROCESSOR
Want key 5
Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-5 @ 128.1.2.4
6-8 @ 128.1.2.5
Back Down Search

QUERY PROCESSOR
Want key 5
Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-5 @ 128.1.2.4
6-8 @ 128.1.2.5
Back Down Search

QUERY PROCESSOR
Want key 5

Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-5 @ 128.1.2.4
6-8 @ 128.1.2.5
Back Down Search

QUERY PROCESSOR
Want key 5

Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-5 @ 128.1.2.4
6-8 @ 128.1.2.5
Back Down Search

QUERY PROCESSOR
Want key 5
Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-5 @ 128.1.2.4
6-8 @ 128.1.2.5
QUERY PROCESSOR
Want key 5

Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-5 @ 128.1.2.4  !!?
6-8 @ 128.1.2.5

Notice this is the only internal node that changed
Back Down Search

QUERY PROCESSOR
Want key 5

Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-5 @ 128.1.2.4
6-8 @ 128.1.2.5

Get Ranges
1-4 @ 128.1.2.4
5 @ 128.1.2.6
6-8 @ 128.1.2.5
Back Down Search

QUERY PROCESSOR
Want key 5
Cache:
1-n @ 128.1.2.2
1-8 @ 128.1.2.3
1-4 @ 128.1.2.4
5 @ 128.1.2.6
6-8 @ 128.1.2.5
Get Key 5
Value at key 5
Split by Load

Why do nodes split the keys they hold?

Because they’re having to handle too much load

Nodes split if node access rate in a second exceeds some threshold

Balance B+Tree by node load not node size

(although do also balance by size to avoid exceeding storage limits on nodes)
Concurrent Access

Transaction just fails if another transaction attempts to manipulate data it’s working on

Various methods to avoid this if possible:

● Multi Version Concurrency Control - Reads just snapshot a version of the key range and get that, ignoring any changes during the read
● Right splits - When node splits, move first half of keys to new node so appends to key range can occur concurrently
● etc.

All just ways of avoiding scenarios leading to transaction abort
Experimental Evaluation

Questions:

Key component evaluation

- How well does YDBT perform?

Architecture comparison

- How does Yesquel compare to NoSQL systems at their own game?
- How does Yesquel compare to existing SQL systems in terms of functionality?
Experimental Evaluation: YDBT

- Yesquel DBT compared against ‘Base’ and ‘Base+’

- Base corresponds to feature sets of DBTs in closed source Sinfonia and Minuet

Base: Optimistic Concurrency Control, No BackDown Search, No Load Splits

Base+: Optimistic Concurrency Control, No Load Splits, Has BackDown Search
## Back-Down Search Performance

Insert with 50 clients:

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>Base+/YDBT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries which access root node</td>
<td>39.8%</td>
<td>0.16%</td>
</tr>
</tbody>
</table>
Benefit of Load Splits

- Without Load Splits, scaling limited as can’t utilise processing power of extra servers (until size of data warrants a split).
Experimental Evaluation: Yesquel

Two testing setups:

- **Performance under key-value operations**
  - Offers a small granularity for metrics
  - Useful for gathering information on certain types of operations
  - Good for comparing different architectures per-operation
  - Gives little indication of performance in specific domain

- **Performance under a real Web application**
  - Real-world evaluation of Yesquel’s high-level aim
  - No comparison to other architectures
Experimental Evaluation: Comparisons

**Redis**

Most popular NoSQL key-value storage system\(^1\); used by Imgur, Github and Snapchat\(^2\)

**MySQL**

A central database system, limited to a single server

**MySQL-NDB (MySQL Network Database)**

Also known as MySQL Cluster, it is a distributed database system consisting of a MySQL front-end and the NDB distributed storage engine

Experimental Evaluation: Workloads

Read

A key $k$ is chosen randomly from a uniform distribution and its value is read

Update

In a transaction, reads the value of $k$ as above, then writes a new value to $k$

Scan

This workload is 95% scan operation, 5% update operations. The scan operation enumerates $n$ (1 to 5) consecutive key-value pairs starting from $k$ within a transaction
Experimental Evaluation: Results

One-server performance (Fig.15)
Experimental Evaluation: Results

Scalability (Fig. 17)

- **READ WORKLOAD**
  - read throughput (ops/s) vs. number of servers
  - Labels: redis, yesquel, mysql-ndb
  - Graphs show linear increase with number of servers

- **SCAN WORKLOAD**
  - scan throughput (ops/s) vs. number of servers
  - Labels: redis, yesquel, mysql-ndb
  - Graphs show different behavior compared to read workload
Experimental Evaluation: Real Web Application

Evaluating performance on a realistic workload

Databases store data for 190K pages from Simple English Wikipedia (430MB)

- Simulate the loading of an article page via query traces
- Choose the article to “visit” uniformly at random

Experiments:

- **Single server**: offers comparison with MySQL
- **Many servers**: uses baseline of ideal linear scalability… and MySQL!
Experimental Evaluation: Real Web Application

Evaluating performance on a realistic workload

Databases store data for 190K pages from Simple English Wikipedia (430MB)

- Simulate the loading of an article page via query traces
- Choose the article to “visit” uniformly at random

No comparison with a NoSQL system!
Related Work

- The YDBT back-down search is inspired by the give-up technique\[1\]

- OLAP vs OLTP: Scaling SQL for analytics
  - HANA SOE\[2\] aims to scale for both

- MoSQL, FoundationDB etc - SQL system above NoSQL storage engine
  - Proposed in Deuteronomy\[3\]

- Shared Nothing vs Shared Disk - Two approaches to distributed databases

References:
[3] Levandoski et al., Deuteronomy: Transaction support for cloud data.
Future Work

- Implement replit
- Replicate storage servers with Paxos
- Row stored procedures
  - NoSQL systems can update rows in a single RTT
  - Yesquel requires a read followed by a write - no blind writes or increments
  - Solution is to have stored procedures execute updates locally
- Implement a query optimiser designed for distributed systems
  - Factors in the distributed nature of the storage system when making decisions
  - Can have a big impact on the performance of a system
- Operators at the storage server
  - All relational operators currently executed at the client after data is retrieved
  - Instead, only retrieve the data from each node that’s necessary for the operation
  - E.g. not all data is required from all storage nodes to calculate the average of a value
Conclusion

Attempts to provide NoSQL performance with an SQL storage system

Doesn’t really evaluate the value of SQL’s features

- Do developers really care or have they adapted to NoSQL?

Still sacrificing performance to gain SQL feature set

Is it really needed?

Time will tell if it gains any traction - only published 6 months ago!