Building a scalable row, column or document-style store with MySQL and Shard-Query

12/15/2012
Agenda

- Introduction
- The new age of multi-core
- Enter Shard-Query
- Focus on OLAP (reporting and analytics)
- Row store (normalized)
- Column store (star schema)
- Document store (hybrid schema)
- QA
Introduction
FOSS I maintain

- Shard-Query
  - MPP distributed query middleware for MySQL
  - Work is divided up using sharding, table partitioning and SQL features.
  - Distributes work over many machines in parallel
  - http://code.google.com/p/shard-query
• Flexviews

  • Caches result sets and can refresh them efficiently based on only the database changes since the last refresh

  • Refresh cost is directly proportional to the number of changes to the base data

  • http://code.google.com/p/flexviews
THE NEW AGE OF MULTI-CORE
The new age of multi-core

“If your time to you is worth saving, then you better start swimming. Or you'll sink like a stone. For the times they are a-changing.”
- Bob Dylan
Moore’s law

- The number of transistors in a CPU doubles every 24 months
- In combination with other component improvements, this allowed doubling of CPU clock speed approximately every 18 months
- Frequency scaling beyond a few GHz has extreme power requirements
- Increased power = increased heat
Moore’s law today

- Speed now doubles more slowly: 36 months
- Power efficient multiple-core designs have become very mature
- Larger number of slower cores which use less power in aggregate
Question:

Is multi-core faster?
A program is only faster on a multiple core CPU if it can use more than one core.
Why?

1. A physical CPU may have many logical cpu

2. Each logical CPU runs at most one thread at one time

3. Max running threads:
   1. Physical CPU count x CPU core density x threads per core
      - Dual CPU with 3 HT cores = 2 x 3 x 2 = 12 threads
What is a thread?

• Every program uses at least one thread
• A multithreaded program can do more work
  – But only if it can split the work into many threads
• If it doesn’t split up the work: then there is no speedup.
What is a task?

• An action in the system which results in work
  – A login
  – A report
  – An individual query

• Tasks are single or multi-threaded
  • Tasks may have sub-tasks
Response Time

- Time to complete a task in wall clock time
- Response time includes the time to complete all sub-tasks
  - You must include queue time in response time
Throughput

- How many tasks complete in a given unit of time
- Throughput is a measure of overall work done by the system
Throughput vs Response time

Response time = \( \frac{\text{Time}}{\text{Task}} \)

Throughput = \( \frac{\text{Tasks}}{\text{Time}} \)
Efficiency

- This is a score about how well you scheduled the work for your task. Inefficient workloads underutilize resources.

\[
\left( \frac{\text{Resources Used}}{\text{Resources Available}} \right) \times 100
\]
Efficiency Example

\[
\left( \frac{\text{Resources Used}}{\text{Resources Available}} \right) \times 100
\]

- CPU bound workload given 8 available cores
  - When all work is done in single thread: 12.5% CPU efficiency
  - If the work can be split into 8 threads: 100% CPU efficiency
Lost time is lost money

• You are paying for wall clock time
  • Return on investment is directly proportional to efficiency.

• You can’t bank lost time
• If you miss an SLA or response time objective you lost the time investment, plus you may have to pay an penalty
• It may be impossible to get critical insight
Scalability

When resources are added what happens to efficiency?

• Given the same workload, if throughput does not increase:
  – Adding even more resources will not improve performance
  – But you may have the resources for more concurrent tasks
  – This is the traditional database scaling model
Scalability

When resources are added what happens to efficiency?

• If throughput increases
  • The system scales up to the workload
  • When people ask if a system is scalable, this is usually what they want to know.
Scalability

When resources are added what happens to efficiency?

• If throughput goes down there is negative scalability
  – Mutexes are probably the culprit
  – This is still the biggest contention point for databases with many cores
  – This means you can’t just scale up a single machine forever. You will always hit a limit (currently 32-48 cores).
Response time is very important!

• Example:
  – It takes 3 days for a 1 day report
    • It doesn’t matter if you can run 100 reports at once
    • Response time for any 1 report is too high if you need to make decisions today about yesterday
Workload

- A workload consists of many tasks
- Typical database workloads are categorized by the nature of the individual tasks
  - Typical database workloads are OLTP and OLAP
OLTP

- Frequent highly concurrent reads and writes of **small** amounts data per query.
- Simple queries, little aggregation
- Many small queries naturally divide the work over many cores
Known OLTP scalability issues

- Many concurrent threads accessing critical memory areas leads to *mutex contention*
- Reducing global mutex contention main dev focus
- Mutex contention is still the biggest bottleneck
  - Prevents scaling up forever (32 cores max)
OLAP (analytical queries)

- Low concurrency reads of **large** amounts of data
- Complex queries and frequent aggregation
- STAR schema common (data mart)
- Single table (machine generated data) common
- Partitioning very common
Known OLAP scalability issues

• IO bottleneck usually gets hit first
• However, even if all data is in memory it still may be impossible to reach the \textit{response time objective}
  • Queries may not be able to complete fast enough to meet business objectives because:
    • MySQL only supports nested loops*
    • All queries are single threaded

* MariaDB has some support for hash joins
You don’t need a bigger boat

• Buying a bigger server probably won’t help for individual queries that are CPU bound.
• Queries are still single threaded.
You need to change the workload!

• Turn OLAP queries into something more like OLTP
  – Split one complex query into many smaller queries
  – Run those queries in parallel
  – Put the results together so it looks like nothing happened
  – This leverages multiple cores and multiple servers

• Doing this *manually* for all queries is not practical.
Life sometimes give you lemons
Enter Shard-Query

- Shard-Query **transparently** splits up complex SQL into smaller SQL statements (sub tasks) which run concurrently
  - MySQL proxy (small LUA interface to shard-query)
  - REST / HTTP GUI
  - PHP OO interface
Why not get a different database?

• Because MySQL is a great database and sticking with it has advantages
  • Operations knows how to work with it (backups!)
  • It is FOSS (alternatives are very costly or very limited)
  • MySQL’s special dialect means changes to apps to move to an alternative database
  • The alternatives are either a proprietary RDBMS* or map/reduce.

* Vertica, Greenplum, Vectorwise, AsterData, Teradata, etc…
Smaller queries are better queries

- This is closer to the OLTP workload that the database has been optimized for
  - Smaller queries use smaller temporary tables resulting in more in-memory aggregation
  - This reduces the usage of temporary tables on disk
- Parallelism reduces response time
  - The small queries run in parallel for a massive speedup
Data level parallelism

- **Table level partitioning**
  - One big table is split up into smaller tables internally
  - Reduce IO and contention on shared data structures in the database.
  - MySQL *could* operate on partitions in parallel
    - But MySQL doesn’t
    - Shard-Query *does*
Sharding
- Horizontally partition data over more than one server.
- Shards can naturally be operated on in parallel.
- This is called shared-nothing architecture, or data level parallelism.
- This is also called scaling out.
Shard-Query

MySQL Proxy
REST Interface
OO Interface

SQL

Mapper
Parser
Flow

Gearman
Workers

DB 0

Mapping and config node

Gearman message queue manages work

Storage Nodes (shards)
Map/Config Node and Shards

Directory Mapper
Shard selection is based on a mapping table. The shard column value is looked up in the table.

Table: sales

<table>
<thead>
<tr>
<th>customer_id</th>
<th>order_date</th>
<th>item</th>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1/2/2013</td>
<td>phone</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>200</td>
<td>1/4/2013</td>
<td>netbook</td>
<td>2</td>
<td>1000</td>
</tr>
<tr>
<td>300</td>
<td>1/5/2013</td>
<td>mouse</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>400</td>
<td>1/3/2013</td>
<td>printer</td>
<td>1</td>
<td>175</td>
</tr>
</tbody>
</table>

Shard Column: customer_id

Map Table

<table>
<thead>
<tr>
<th>schema_name</th>
<th>column_name</th>
<th>column_value</th>
<th>shard_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>sales</td>
<td>customer_id</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>sales</td>
<td>customer_id</td>
<td>200</td>
<td>2</td>
</tr>
<tr>
<td>sales</td>
<td>customer_id</td>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>sales</td>
<td>customer_id</td>
<td>400</td>
<td>1</td>
</tr>
</tbody>
</table>

* simplification
Hash mapping of shards

**Hash Mapper**
Shard selection is based on the modulus of the shard column value over the number of shards.

**Table: sales**

<table>
<thead>
<tr>
<th>customer_id</th>
<th>order_date</th>
<th>item</th>
<th>qty</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1/2/2013</td>
<td>phone</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>200</td>
<td>1/4/2013</td>
<td>netbook</td>
<td>2</td>
<td>1000</td>
</tr>
<tr>
<td>300</td>
<td>1/5/2013</td>
<td>mouse</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>400</td>
<td>1/3/2013</td>
<td>printer</td>
<td>1</td>
<td>175</td>
</tr>
</tbody>
</table>

Shard Column: `customer_id`

Hash function:
COLUMN_VALUE % NUM_SHARDS = SHARD_ID
Unsharded tables and joins

- Tables that do not contain the shard key are replicated to all shards.
- This allows joins between sharded and unsharded tables (lookup tables/dimension tables).
- This means Shard-Query works best for star and snowflake schema.
- Many open source tools like Mondrian work well with a star schema.
But not just for sharding

- The name is misleading as you have choices:
  - **Partition** your data on one big server to scale up
  - **Shard** your data onto multiple servers to scale out
  - **Do both** for extreme scalability
    - Or neither, but with less benefits
Parallelism of SQL dataflow

- Shard-Query can add parallelism and improve query performance based on SQL language constructs:
  - IN lists and subqueries are parallelized
  - BETWEEN on date or integer operands adds parallelism too
  - UNION ALL and UNION queries are parallelized
  - Uncorrelated subqueries are materialized early, are indexed and run in parallel (inside out execution)

* They have to have features necessary to enable parallelism. You really need to partition and shard for best results
Shard-Query is not for OLTP

• Parser and execution strategy is “relatively expensive”
  – For OLAP this time is a small fraction of the overall execution time and the cost is “cheap”
  – Tying to turn OLTP into OLTP is wasted effort and waste reduces efficiency
  – But OLTP still caps out at 32 cores on a single box
    • So what do I do for OLTP?
Sharding for OLTP

• Use **Shard-Key-Mapper**
  • Use this helper class to figure out which shard to use
  • Then send queries directly to that shard (bypass parser)
  • This could be made transparent with mysqlnd plugins

• You can then still use SQ when complex query tasks are needed

• This is a topic for another day
Shard-Query Parallelism

- Shard-Query adds *intra-query* parallelism to MySQL
  - This means that more than one thread operates on a SQL query
  - This can result in massive speedups because the work is divided up between multiple threads and even multiple machines
  - Data distribution should be even to ensure best parallelism
Intra-Query Parallelism
You get to move all the pieces at the same time
Shard-Query 2.0 Features

• **Automatic sharding and massively parallel loading**
  – Add new shards then spread new data over them automatically
  – You can manually move records between shards

• **Complex query support**
  – GROUP BY, HAVING, LIMIT, ORDER BY, WITH ROLLUP, derived tables, UNION, etc

• But not:
  – Joins between tables sharded on different keys (soon)
Shard-Query 2.0 Features

• Massively parallel loader / distributed loader
  – Can split loading into chunks split over many threads

• Long query support
  – Supports asynchronous jobs for running long queries

• DML/DDL support
  – Supports almost all MySQL SQL including INSERT, UPDATE, DELETE
  – DDL is intelligently broadcast to all shards
Shard-Query 2.0

SQL

DATA

Data Flow

Shard-Query + Gearman + Sharding and/or Partitioned Tables = Parallel Execution

Task1

Task2

Task3

Task4

Shard1

Shard1

Shard2

Shard2

Partition 1

Partition 1

Partition 2

Partition 2

GUI Proxy PHP OO

Open Source MPP database engine
Code Maturity

• Revision 1, May 22, 2010 0.0.1
  • 270 lines of PHP code checked into Google Code
  • Limited select support, no aggregation pushdown
  • One developer (me) – only suited for limited audience as a POC
  • Not object oriented, or a framework, just a simple CLI PHP app

• Revision 447, Jan 28, 2013 – Shard Query beta 2.0.0
  • Over 9700 lines of PHP code
  • Full coverage of SELECT statement plus custom aggregate function support
  • Support for all almost DML and DDL operations (except create/drop database)
  • Two additional developers
    • Special thanks to Alex Hurd who is a production tester and contributor of the REST interface
    • And Andre Rothe, who contributes to the SQL parser
  • REST web interface, MySQL proxy Lua script, OO PHP library framework
  • Feature complete and stable to a level fit for community use
## EC2 Instance Sizes

### Instance Sizes Compared

<table>
<thead>
<tr>
<th>Instance Size</th>
<th>Feature</th>
<th>Measure</th>
<th>Price Per hour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>m2.xlarge</strong></td>
<td>cores</td>
<td>2</td>
<td>0.41 USD</td>
</tr>
<tr>
<td><strong>The Pawn</strong></td>
<td>ecu</td>
<td>6.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ecu/core</td>
<td>3.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>storage</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory</td>
<td>17.1</td>
<td></td>
</tr>
<tr>
<td><strong>hs1.8xlarge</strong></td>
<td>cores</td>
<td>16</td>
<td>3.1 USD</td>
</tr>
<tr>
<td><strong>The King</strong></td>
<td>ecu</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ecu/core</td>
<td>2.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>storage</td>
<td>2048</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory</td>
<td>60.5</td>
<td></td>
</tr>
</tbody>
</table>

For a few cents more, we get much more CPU power in aggregate if eight smaller machines are used.

### Scale Out

<table>
<thead>
<tr>
<th>Instance Size</th>
<th>Feature</th>
<th>1x Machine</th>
<th>8x Machines</th>
<th>8X Price Per hour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>m2.xlarge</strong></td>
<td>cores</td>
<td>2</td>
<td>16</td>
<td>3.28 USD</td>
</tr>
<tr>
<td><strong>The Pawn</strong></td>
<td>ecu</td>
<td>6.5</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ecu/core</td>
<td>3.25</td>
<td>3.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>storage</td>
<td>420</td>
<td>3360</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory</td>
<td>17.1</td>
<td>136.8</td>
<td></td>
</tr>
<tr>
<td><strong>hs1.8xlarge</strong></td>
<td>cores</td>
<td>16</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td><strong>The King</strong></td>
<td>ecu</td>
<td>35</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ecu/core</td>
<td>2.25</td>
<td>2.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>storage</td>
<td>2048</td>
<td>16384</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory</td>
<td>60.5</td>
<td>484</td>
<td></td>
</tr>
</tbody>
</table>
Distributed row store w/ Galera

- Each shard is an Percona XtraDB Cluster
  - HA sharding solution with simple Xtrabackup backup
  - Support massive ingestion rates via MPP loader
  - real time ad-hoc complex querying (sub second)
  - For OLTP access to the same data, use Shard-Key-Mapper
Distributed row store w/ Galera

- All the components support HA
  - Galera, Gearman, Apache, PHP, MySQL proxy
  - Redundancy can be fully geographically distributed
  - Use partitioning at the table level too
  - Use InnoDB compression
Simple In-Memory COUNT(*) query performance

<table>
<thead>
<tr>
<th>Days</th>
<th>8 Pawns</th>
<th>The King</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.552858</td>
<td>40.84573</td>
</tr>
<tr>
<td>2</td>
<td>5.090356</td>
<td>81.4457</td>
</tr>
<tr>
<td>3</td>
<td>8.064888</td>
<td>129.0382</td>
</tr>
<tr>
<td>4</td>
<td>10.74412</td>
<td>171.9059</td>
</tr>
<tr>
<td>5</td>
<td>13.32697</td>
<td>213.2316</td>
</tr>
<tr>
<td>6</td>
<td>16.0227</td>
<td>256.3633</td>
</tr>
<tr>
<td>7</td>
<td>18.50571</td>
<td>296.0914</td>
</tr>
<tr>
<td>8</td>
<td>21.02053</td>
<td>336.3285</td>
</tr>
<tr>
<td>9</td>
<td>25.3414</td>
<td>405.4624</td>
</tr>
<tr>
<td>10</td>
<td>29.69324</td>
<td>475.0918</td>
</tr>
<tr>
<td>11</td>
<td>32.93455</td>
<td>526.9529</td>
</tr>
<tr>
<td>12</td>
<td>36.5517</td>
<td>584.8272</td>
</tr>
<tr>
<td>13</td>
<td>40.19016</td>
<td>643.0426</td>
</tr>
<tr>
<td>14</td>
<td>42.75</td>
<td>699.1011</td>
</tr>
<tr>
<td>15</td>
<td>44.69</td>
<td>750.4571</td>
</tr>
</tbody>
</table>

Lower is better
Distributed column store

Each shard is a Infobright database
  Pros
    • Hash joins, small data footprint, no indexes

Cons
  • Single threaded loading (one thread per shard, can’t take advantage of MPP loader)
  • Append only (LOAD DATA INFILE)
  • No partitions
Distributed column store

- There is a bug in Infobright Community Edition that is currently preventing Shard-Query from working properly
- I’m working on a workaround for the bug.
- I expect to have the fix released by the end of next week.
Distributed document store

- The “schema” is simply
  
  ```
  CREATE TABLE document_type (  
    document_id bigint auto_increment primary key,  
    doc longtext  
  );
  ```

- Unfortunately, finding rows by anything other than the `document_id` is not possible without using LIKE 😞

- Or is it?
Need to extract/index the data

• We could create a function that extracts a key from a JSON document

• Use the function in SQL to extract the data

• Find a way to create a “function index” for the extraction
create function json_extract (
    v_doc longtext charset utf8,
    v_key longtext charset utf8
) returns int unsigned
comment 'Extracts JSON value'
language SQL
deterministic
modifies sql data
sql security invoker
begin
    declare v_start int default 0;
    declare v_end int default 0;
    declare v_search text charset utf8 default "";

    set v_search := concat("", v_key, ":");
    set v_start := instr(v_doc, v_search);
    if v_start = 0 or v_start is null then
        return null;
    end if;

    set v_start := v_start + length(v_search);
    if(substr(v_doc, v_start, 1) = ")") then
        set v_start := v_start + 1;
    end if;

    set v_end := locate("",v_doc);
    if (v_end = 0) then
        set v_end := locate(",", v_doc);
        if v_end = 0 then
            set v_end := locate("}", v_doc);
        end if;
    end if;

    if v_end = 0 then
        return null;
    end if;

    return cast(substr(v_doc, v_start, v_end + if(substr(v_doc, v_start+1,1) = "", -1,0)) as UNSIGNED);
end;
Distributed document store

• Flexviews allows the creation of “materialized views” which are cached views that can be updated efficiently when rows are changed.

• But the view is updated asynchronously

• You can refresh the view periodically.

• You can index the view
Create SQL to extract data

-- SQL to extract the data
SELECT
order_id as order_id,
document_store.json_extract(doc,"customer_id") as customer_id,
document_store.json_extract(doc,"order_id") as order_total,
document_store.json_extract(doc,"order_date") as order_date
FROM document_store.orders as orders;
That SQL as Flexviews SQL API

call flexviews.create_mvlog('document_store','orders');

call flexviews.create('document_store','orders_idx', 'INCREMENTAL');
set @mvid := last_insert_id();

call flexviews.add_table(@mvid, 'document_store', 'orders', 'orders', NULL);

call flexviews.add_expr(@mvid, 'COLUMN', 'order_id', 'order_id');

call flexviews.add_expr(@mvid, 'COLUMN', 'document_store.json_extract (doc,"customer_id")', 'customer_id');

call flexviews.add_expr(@mvid, 'COLUMN', 'document_store.json_extract (doc,"order_id")', 'order_total');

call flexviews.add_expr(@mvid, 'COLUMN', 'document_store.json_extract (doc,"order_date")', 'order_date');
Enable and use the “view”

call flexviews.enable(@mvid);
Query OK, 0 rows affected (34 min 33.23 sec)

-- build btree indexes
alter table orders_idx
add key(order_id),
add key(customer_id),
add key(order_date);
Query OK, 0 rows affected (4 min 33.23 sec)

--use the “index”
mysql> select count(*) from document_store.orders_idx where customer_id = 36\G
*************************** 1. row ***************************
count(*): 28416
1 row in set (0.16 sec)