DIVIDE AND CONQUER IN THE CLOUD:

One BIG SERVER OR MANY SMALL ONES?

Justin Swanhart

FOSDEM 2013

PLMCE 2013

Agenda

- Introduction
- The new age of multi-core (terminology and background)
- Enter Shard-Query
- Performance comparison
- Q/A

Introduction

Who am I?

My name is Justin “Swany” Swanhart

I’m a trainer at Percona

FOSS I maintain

- Shard-Query
  - MPP distributed query middleware for MySQL*
  - Work is mostly divided up using sharding and/or table partitioning (divide)
  - Distributes work over many machines in parallel (conquer)

FOSS I maintain

- Flexviews

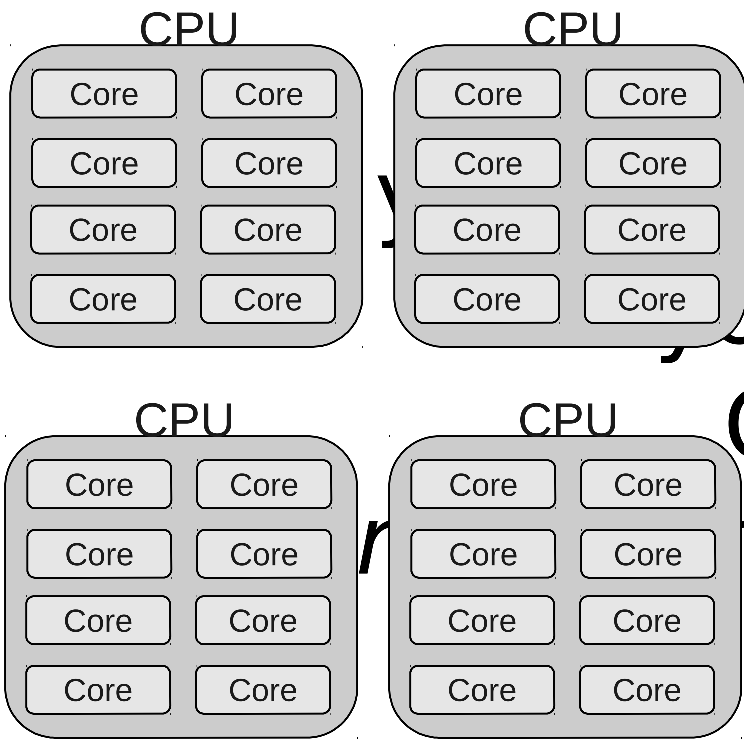
- Caches result sets and can refresh them efficiently based on only the database changes since the last refresh

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

This talk is about Shard-Query

PERFORMANCE IN THE NEW AGE OF MULTI-CORE

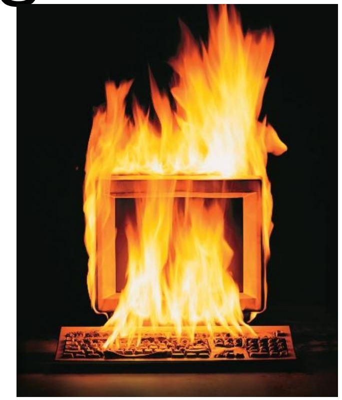
The new age of multi-core



y ne to you is worth saving,
ou better start swimming.
Or you'll sink like a stone.
r nes they are a-changing.”
\- **Bob Dyla**

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?

Answer: It depends

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:
4. Physical CPU count x CPU core density x threads per core
    1. Dual CPU with 3 HT cores = 2 x 3 x 2

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 = Time / Task

Throughput = Tasks / Time

Efficiency

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

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

Efficiency Example

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

- 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 <u>resources are added</u> 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 the biggest contention point for databases with many cores
  - This means you can't just scale up a single machine forever. <u>You will always hit a limit.</u>

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

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 *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 limited 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!

You need to change the workload!

- Turn OLAP 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

Life sometimes give you lemons

Enter Shard-Query

- Shard-Query ***transparently*** splits up complex SQL into smaller SQL statements (sub tasks) which run concurrently
  - Proxy
  - 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. For many reasons these are undesirable.

* 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

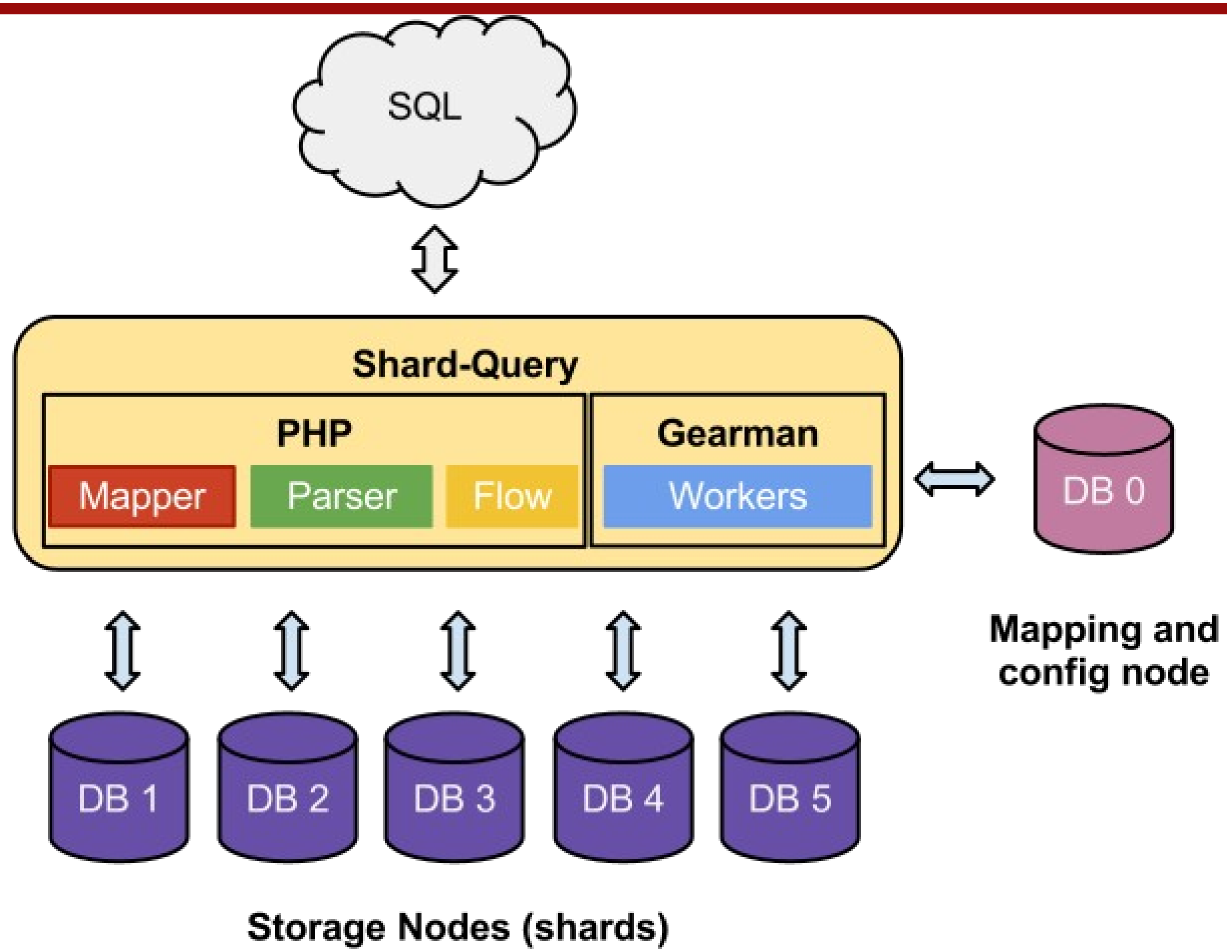
Data level parallelism

- 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 it doesn't**

Data level parallelism (cont)

- 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


SQL
Shard-Query
PHP
Mapper
Parser
Flow
Gearman
Workers
DB 0
Mapping and config node
DB 1
DB 2
DB 3
DB 4
DB 5
Storage Nodes (shards)

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

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

Why: Amdahl's Law

- **Speedup of parallelizing a task is bounded by the amount of serialized work the task must perform**

f(N) = maximum speedup by parallelizing a process

$$f(N) = \frac{1}{(1 - P)} + \frac{P}{N}$$

N = Number of threads that may be run in parallel
P = Portion in percent, of the runtime of the program that may be parallelized
(1 – P) = Portion in percent, of the runtime which is serialized

Amdahl’s Law Example

$$f(N) = \frac{1}{(1 - P) + \frac{P}{N}}$$

Your task: SUM a set of 10000 items

- Serially summing the items takes 10000* seconds

- Solution: Partition the set into 10 subsets
- Sum items using one thread per partition

* Times are exaggerated for demonstration purposes.

Amdahl’s Law Example

$$f(N) = \frac{1}{(1-P)} + \frac{P}{N}$$

- Summing 10 sets items in parallel takes 1000 seconds (10000s/10=1000s)
  - Add an <u>extra 10 seconds</u> to add up the 10 sub-results
  - This is .1% of the original runtime

Max speedup = (.1–99.9))+(99.0/10) = 9.98X

-

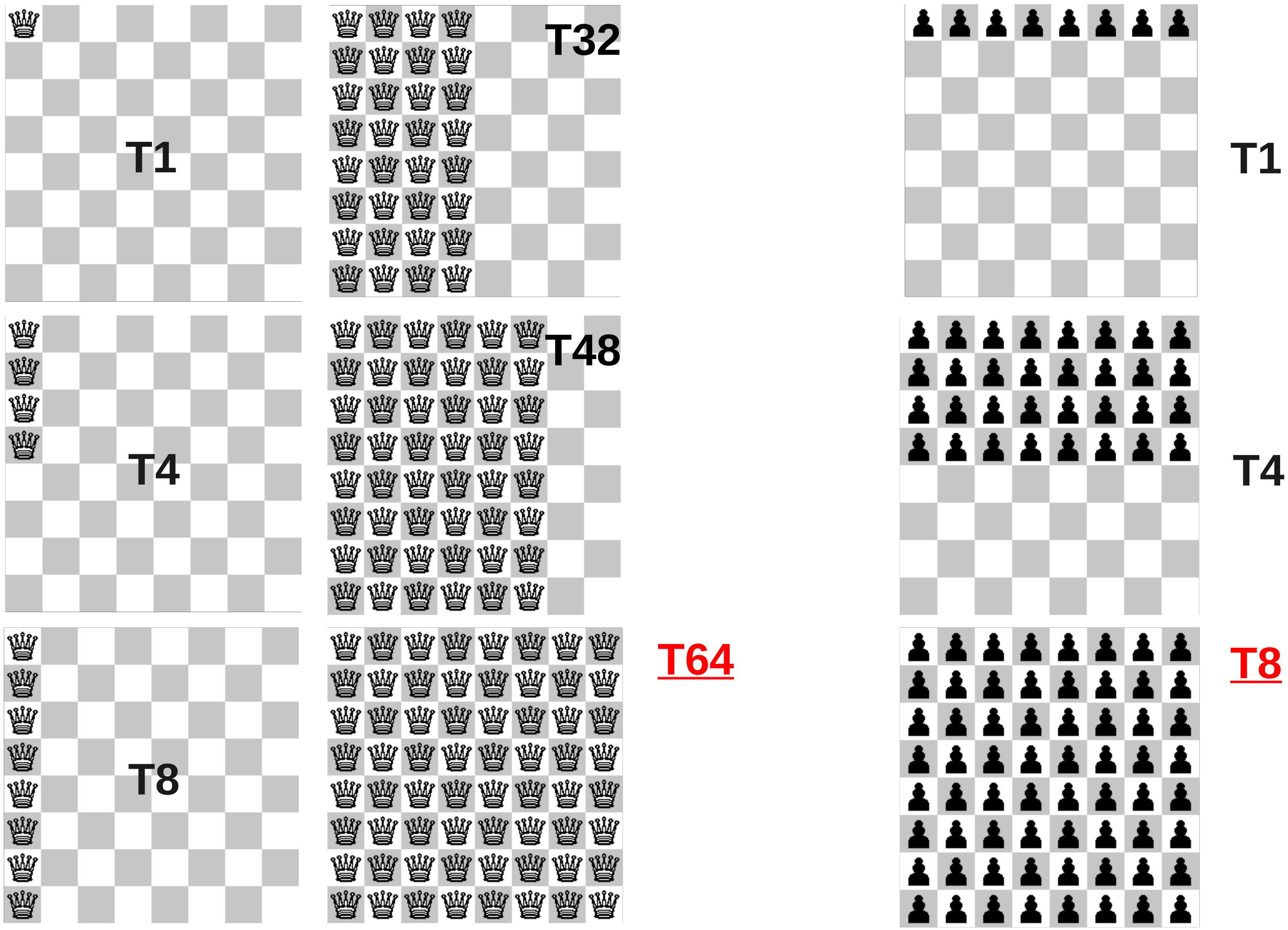
* Times are exaggerated for demonstration purposes.

Amdahl’s Law Example 2

Your task: STDDEV a set of 10000 items

- Some aggregate functions like STDDEV can’t be distributed like SUM
- Serially doing a STDDEV takes 10000 seconds
- Because it can’t be split, we are forced to use one thread
- The STDDEV takes 10000 second (10000/1)
- Max speedup = 1.0 (no speedup)
- Overhead may reduce efficiency

You get to move all the pieces at the same time



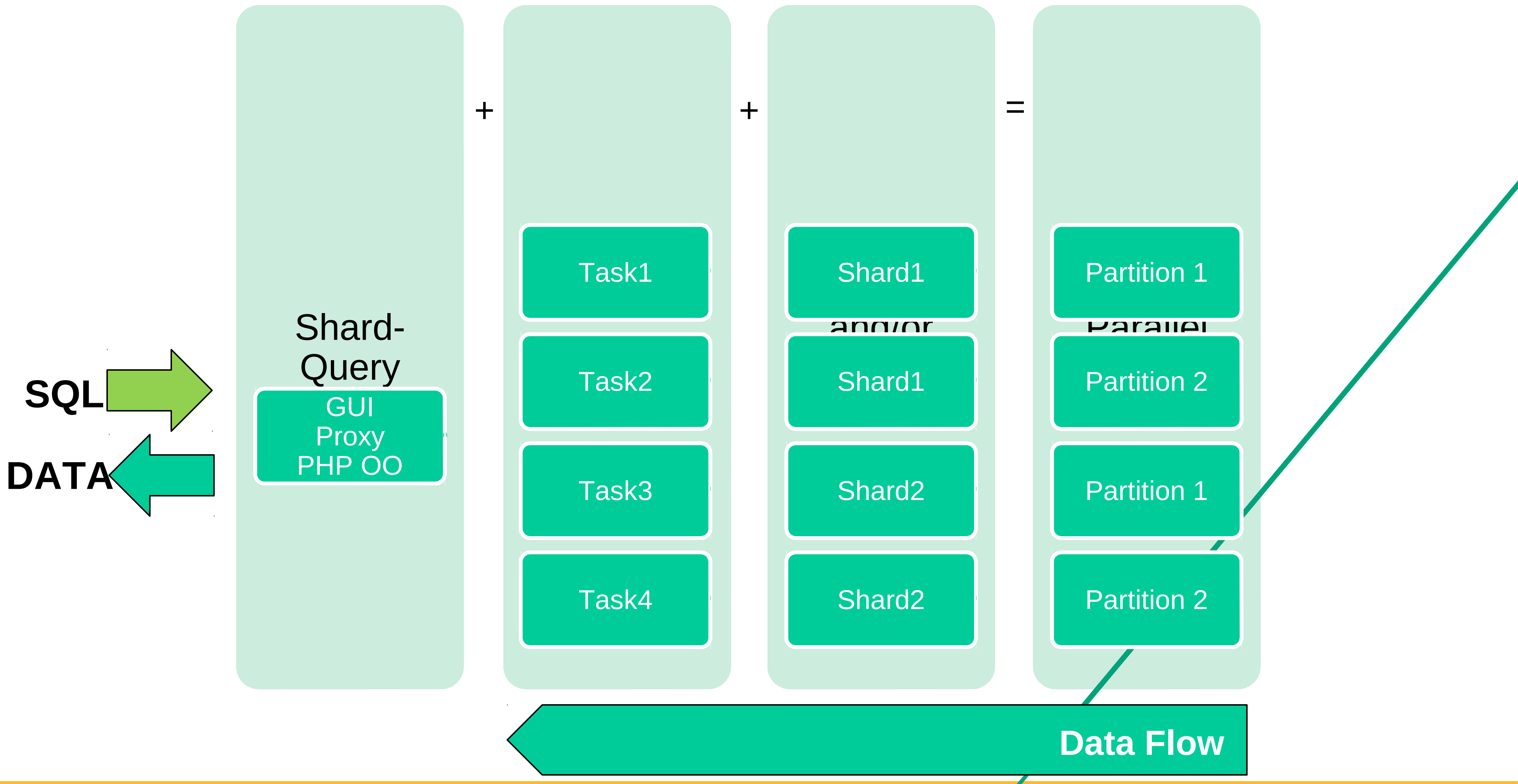
Shard-Query 2.0 Features

- **Automatic sharding and massively parallel loading**
  - Add new shards then spread new data over them automatically
- **Long query support**
  - Supports asynchronous jobs for running long queries
- **Complex query suppo**rt
  - GROUP BY, HAVING, LIMIT, ORDER BY, WITH ROLLUP, derived tables, UNION, etc

Shard-Query 2.0 Features

- **Automatic sharding and massively parallel loading**
  - Add new shards then spread new data over them automatically
- **Long query support**
  - Supports asynchronous jobs for running long queries

Shard-Query 2.0



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

One big server or many small ones

Big Data: Wikipedia Traffic Stats

- Wikipedia traffic stats
  - 2 months of data (Feb and Mar 2008)
  - 180 GB raw data (single table)
  - No PK
  - Partitioned by date, one partition per day
  - One index, on the wiki language ("en", etc)
  - 360 GB data directory after loading
    - 7 years of data is available (15TB)

http://dumps.wikimedia.org/other/pagecounts-raw/

Big Data: Wikipedia Traffic Stats

```
CREATE TABLE `fact` (
  `date_id` int(11) NOT NULL,
  `hour_num` tinyint(4) NOT NULL,
  `page` varchar(1024) NOT NULL,
  `project` char(3) NOT NULL,
  `language` varchar(12) NOT NULL,
  `cnt` bigint(20) NOT NULL,
  `bytes` bigint(20) NOT NULL,
  KEY `language` (`language`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
PARTITION BY RANGE (date_id)
(PARTITION p1 VALUES LESS THAN (20080101) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (20080102) ENGINE = InnoDB,
PARTITION p3 VALUES LESS THAN (20080103) ENGINE = InnoDB,
…,
PARTITION p91 VALUES LESS THAN (20080331) ENGINE = InnoDB,
PARTITION p92 VALUES LESS THAN (20080401) ENGINE = InnoDB,
PARTITION pmax VALUES LESS THAN (MAXVALUE) ENGINE = InnoDB,
)
```

Data set is sharded by hour_num to evenly distribute work between servers (each server has 1/8 of the data)

SQ can split tasks over partitions too, for multiple tasks per shard

EC2 Instance Sizes

Instance Sizes Compared			
Instance Size	***Feature***	***Measure***	***Price Per hour***
m2.xlarge The Pawn	**cores**	**2**	**0.41 USD**
	ecu	6.5	
	ecu/core	**3.25**	
	storage	420	
	memory	17.1	
hs1.8xlarge The King	**cores**	**16**	**3.1 USD**
	ecu	35	
	ecu/core	**2.25**	
	storage	2048	
	memory	60.5	

Scale Out				
Instance Size	***Feature***	***1x Machine***	***8x Machines***	***8X Price Per hour***
m2.xlarge The Pawn	**cores**	2	16	**3.28 USD**
	ecu	6.5	**52**	
	ecu/core	**3.25**	**3.25**	
	storage	420	3360	
	memory	17.1	**136.8**	
hs1.8xlarge The King	**cores**	16	128	
	ecu	**35**	280	
	ecu/core	**2.25**	**2.25**	
	storage	2048	16384	
	memory	60.5	484	

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

Working set fits into the ram of 8 servers

Anybody want to bet on the King?

Working Set

- The queries examine up to **15 days** of data
  - It is important that the working set be able to be kept in memory for best performance
  - Each partition contains approximately 8GB of data
  - Working set is 120GB of InnoDB pages (60GB raw)
    - Doesn’t fit on a single server

http://dumps.wikimedia.org/other/pagecounts-raw/

Don’t bet on the king…

Single-Threaded Query Performance the <u>unsharded</u> data set

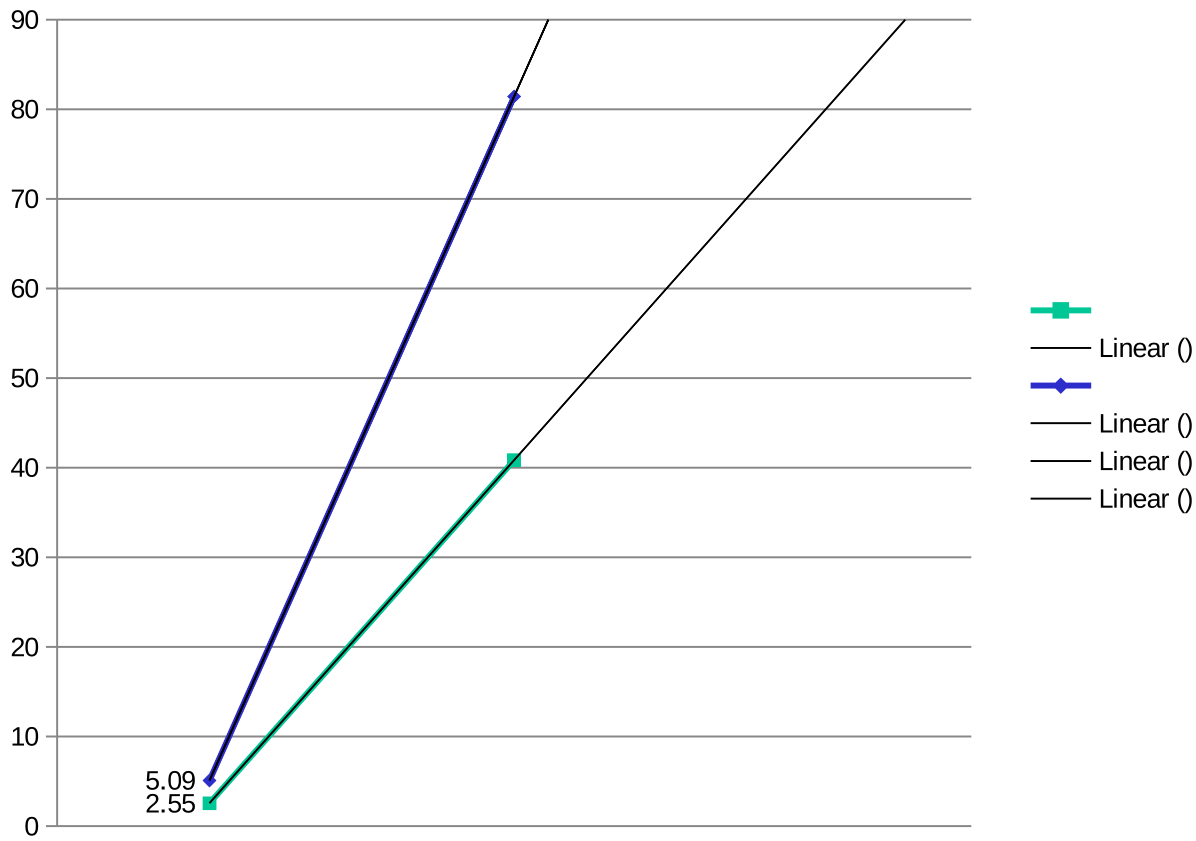
```
select count(*)
  from stats.fact
 where date_id = 20080214;
+----------+
| count(*) |
+----------+
| 84795020 |
+----------+
```

8 shards should
be able to do the work
In ~18s or ~2 seconds,
depending on where
the bottleneck is.

Instance Sizes Compared (again)

Instance Size	*Feature*	*Measure*	*Performance COLD / HOT*	*Price Per hour*
m2.xlarge The Pawn	**cores**	**2**		0.41 USD
	ecu	6.5		
	ecu/core	**3.25**	**2m22s / 33s**	**.21/core**
	storage	420		
	memory	17.1		
hs1.8xlarge The King	**cores**	**16**		3.1 USD
	ecu	35		
	ecu/core	**2.25**	**1m10s / 43s**	**.19/core**
	storage	2048		
	memory	60.5		

Simple In-Memory COUNT(*) query performance



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

Where to go from here

- Shard-Query is not the answer to every performance problem

- Flexviews can be used too
- This helps when the working set is too large to keep in memory even with many machines
- Cost of aggregation is amortized over time, and distributed over many machines if you still shard
- Scales reads, writes, aggregation

- But it is an excellent addition to your toolbox

Distributed row store w/ Galera

- Each shard is an Percona XtraDB or MariaDB cluster
  - Support massive ingestion rates via MPP loader
  - real time ad-hoc complex querying
  - All the components support HA
    - Galera, Gearman, Apache, PHP, MySQL proxy
  - Redundancy can be fully geographically distributed
  - Use partitioning at the table level too

Distributed document store

- Each shard is a MariaDB cluster
  - Use dynamic columns and anchor models for dynamic schema

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
  - SQL harder to parallelize for scale up

justin.swanhart@percona.com
@jswanhart

We're Hiring! www.percona.com/about-us/careers/

I'm also speaking at:

PERCONA LIVE

1. Detecting and preventing SQL injection with the Percona Toolkit
2. Divide and Conquer in the cloud: One big server or many small ones?
3. Introduction to open source column stores

PLMCE.com

April 2013