How to Scale Big Data Applications

USING MYSQL SHARDING FRAMEWORKS

Introduction

- Author: Justin Swanhart

Principal Support Engineer @ Percona

Creator of Shard-Query and Flexviews

What is big data?

What is big data?

- Big data is an all-encompassing term for any collection of data sets so large and complex that it becomes difficult to process using traditional data processing applications.*
  - Is sometimes unstructured data but we will be talking about structured data
  - Basically when the database becomes too large for a single server you are entering big data

* Wikipedia definition

Machine Generated Data (cont)

- Usually data that has been generated by computer or by a sensor
- Data often generated 24/7
- Grows quickly but is usually not updated
- Bulk loading is common

Machine Generated Data (cont)

- Most often the query pattern is OLAP (analytical queries examine large amounts of data)
- Data format unlikely to change so large tables (no alter) are okay
- A column store like Infobright is very good for this kind of data
- Star schema very common:
  - One large table (fact table) for the machine generated data
  - Fact table may have many columns
  - Smaller lookup tables (dimension tables) are common

Machine generated data

- Call detail records
  - Cell phone billing
  - Internet telephone call analysis

- Large online retailer sales system
  - Itemized receipt is kept for each sale
  - Analyze top selling items, etc

Human Generated Data/Content

- Social network content
  - tweets
  - facebook posts
  - blog (livejournal, etc)

- Online auction data
  - auction history (bids, etc)

Human Generated Data/Content (cont)

- Data often generated 24/7
- Grows quickly and is also updated frequently
- Data model varies
- Most often query pattern is OLTP (small amounts of data are read or written constantly)

Too much for one server

- The datasets just described can quickly grow beyond the abilities of even a large server
  - For OLAP
    - the amount of data examined is related to the time it takes to run a query
    - Too much data means queries take too much time
    - Queries are single threaded – adding cores doesn't help single query performance
  - For OTLP
    - Indexes have to be kept in memory for good performance (thus data size must be kept small)
    - The number of connections required is often too high for a single server
    - Simply put: You can't run Facebook on a single MySQL server

Sharding Basics

Sharding – What is it?

- Sharding is a portmanteau of “Shared Nothing”
  - The database is split over many machines
  - Each machine can act alone to answer a query about the data involved in that node.
  - There is no shared state, or shared hardware between nodes.
- Involves splitting up the data
  - Data in the big table (or in some cases tables) is divided up
  - It is split up into identical tables on more than one server
- Lookup tables are not split up (sharded) but are duplicated on each node
  - These are called unsharded tables

Sharding – The shard key

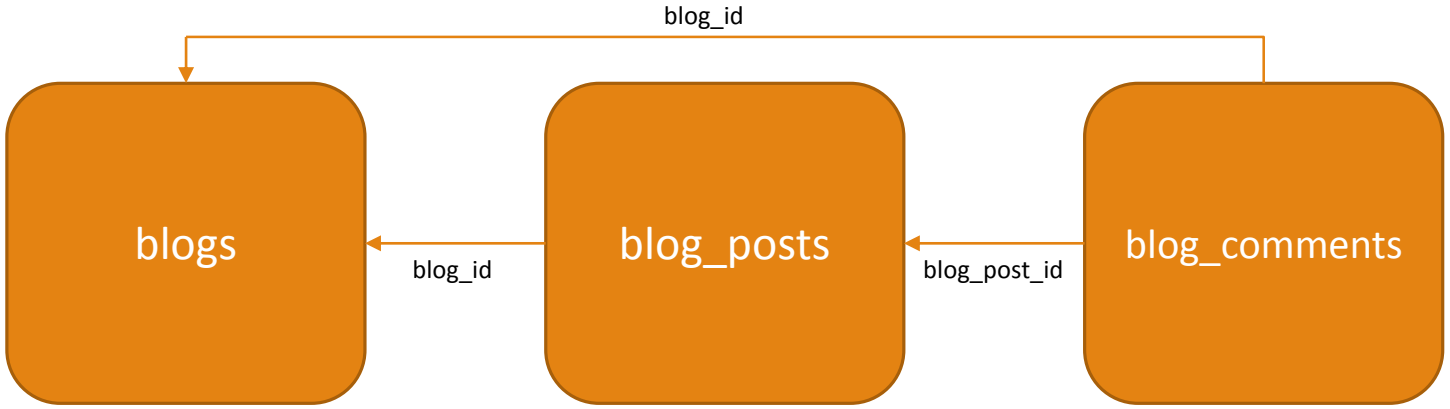
- The shard key is the column in a table that indicates how it is split up (sharded)
- For example, if the shard key is customer_id then
  - All rows for any particular customer key will be located on the same shard
  - The shard mapper will know which shard to go to given a particular customer_id

Sharding – The shard key

- Both MySQL Fabric and Shard-Query require shard keys but they work slightly differently in each framework
  - Fabric allows a per-table definition of the shard key
    - You must specify the shard key in a function call before running SQL (example later)
  - Shard-Query requires a shard key be declared in the configuration
    - Shard-Query automatically recognizes when the shard key is used and sends queries to the right shard

Shard Key example

- A blog site has a column called **blog_id** in the sharded tables
  - The table that lists all blogs (blogs)
  - The table that lists all blog posts (blog_posts)
  - The table that lists all blog comments (blog_comments)
    - Note that this denormalization is necessary
    - It keeps the blog_comments on the same shard as the posts and the blog itself



unsharded

Shard Key example: joins

You can join between sharded tables, but you must join using the shard key:

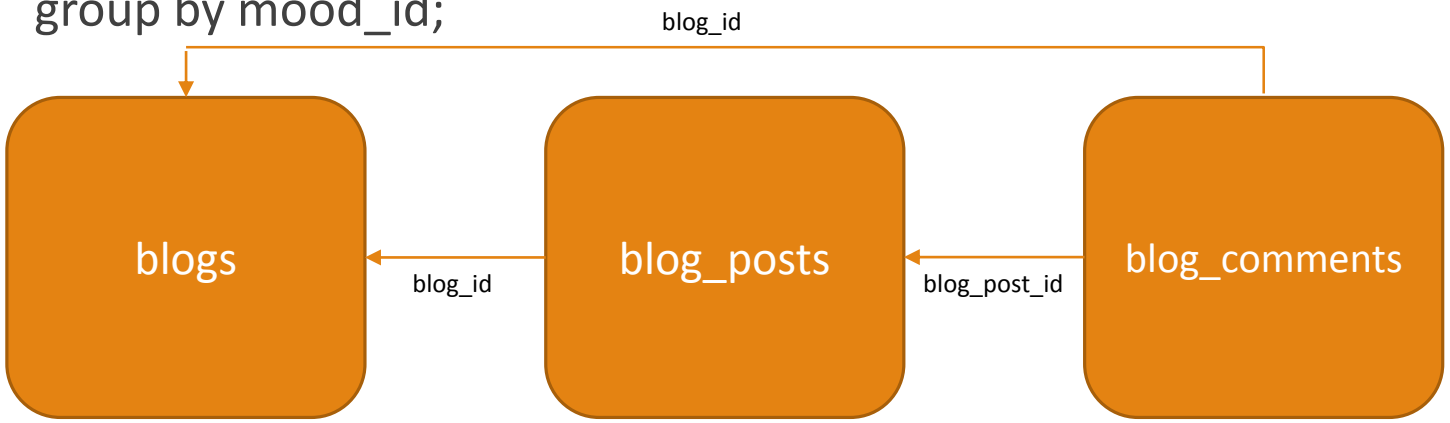
```
select mood.mood_id, count(*) cnt
 from blogs
 join blog_posts on blogs.blog_id = posts.blog_id
 join blog_comments on posts.post_id = blog_comments.post_id
 and blog_comments.blog_id = blogs.blog_id
 join moods on posts.mood_id = moods.mood_id
 where blogs.blog_id = 30
group by mood_id;
```

Shard key

moods

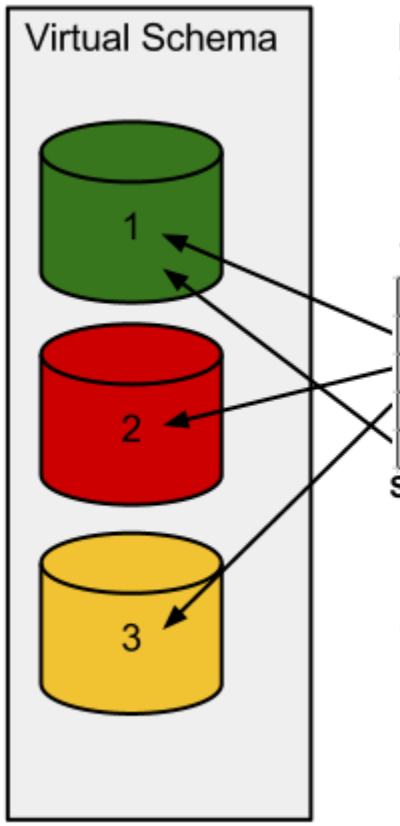
unsharded

Joins to unsharded tables are allowed too



Sharding – Fabric Mappers

- MySQL Fabric supports two mappers:
  - **Hash** mapper tries to evenly distribute rows around the cluster
  - **Range** mapper uses predefined ranges of column values (1 to 25 goes on server1, 26 to 50 goes on server2,etc)
  - Will send statements to the correct shard
    - The shard key must be specified in a function call before the operation is performed (example later)



Hash Mapper

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

Table: sales

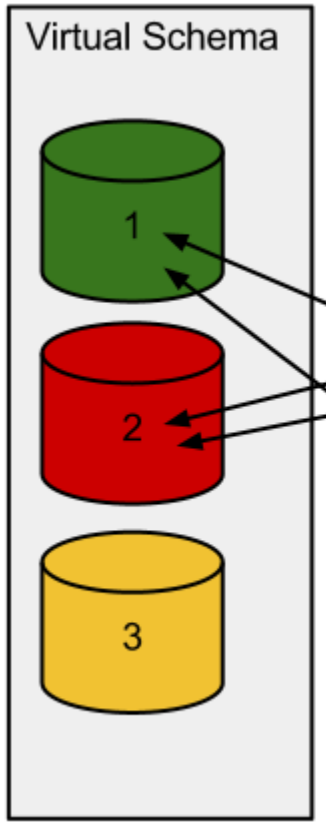
customer_id	order_date	item	qty	price
100	1/2/2013	phone	1	250
200	1/4/2013	netbook	2	1000
300	1/5/2013	mouse	20	10
400	1/3/2013	printer	1	175

Shard Column: customer_id

Hash function:
COLUMN_VALUE % NUM_SHARDS = SHARD_ID

Sharding – Shard-Query mappers

- Shard-Query supports two mappers as well:
  - **Hash** mapper, similar to MySQL Fabric
  - **Directory** mapper – maps a shard key to a particular shard using a lookup table (customer_id=30 is on shard2)
  - The Shard-Query loader understands the shard key and will send rows to the appropriate shard
  - Shard-Query supports INSERT (including bulk insert) and will send rows to the appropriate shard as well



Directory Mapper

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

Table: sales

customer_id	order_date	item	qty	price
100	1/2/2013	phone	1	250
200	1/4/2013	netbook	2	1000
300	1/5/2013	mouse	20	10
400	1/3/2013	printer	1	175

Shard Column: customer_id

Map Table

schema_name	column_name	column_value	shard_id
sales	customer_id	100	1
sales	customer_id	200	2
sales	customer_id	300	2
sales	customer_id	400	1

Sharding: Summary

- Both sharding frameworks have a shard key
- Both sharding frameworks have a shard mapper
  - Together they control data placement on each shard

Sharding Frameworks

MySQL Fabric

- MySQL Fabric
  - Written in Python, it is included as part of MySQL Utilities
  - Stores configuration information in a MySQL database
  - Driver must support MySQL Fabric (MySQL connector for Python, MySQL JDBC connector for Java, PHP)
  - Fabric is good for most OLTP workloads

MySQL Fabric - Terms

- **Group -** A collection of MySQL servers.
- **Global group -** Special groups that store updates that must be propagated to all shards.
- **Node -** A running python instance of MySQL Fabric. This is the XMLRPC server.
- **Shard -** A horizontal partition of data in a table. Usually assigned in ranges
- **Primary -** A group member that has been designated master (the writer).
- **Secondary -** A group member that is read only.

MySQL Fabric

- Create groups of servers
  - High availability can be achieved through managed replication
  - Requires GTID and --log-slave-updates
- Not suitable for OLAP
  - Queries are limited to running on a single shard
  - Therefore you can't do "select count(*) from sharded_table;"
  - You would have to connect to each shard, get the count, then add the counts together
- Management is via the command-line utility **mysqlfabric**

MySQL Fabric Requirements

- A global group with a PRIMARY node
- One group for each shard to reside in
- Configuration of the sharding range boundaries
- Modify application to set the shard key during execution

MySQL Fabric Example – add groups

```
[vagrant@store ~]$ mysqlfabric group create salaries-global
Procedure :
{ uuid = 390aa6c0-acda-40e2-ad52-8c0869613635,
finished = True,
success = True,
return = True,
activities =
}
[vagrant@store ~]$ for i in 1 2; do mysqlfabric group create salaries-$i; done
Procedure :
{ uuid = 274742a2-5e84-49b8-8446-5a8fc55f1899,
finished = True,
success = True,
return = True,
activities =
}
Procedure :
{ uuid = 408cfd6a-ff3a-493e-b39b-a3241d83fda6,
finished = True,
success = True,
return = True,
activities =
}
```

MySQL Fabric Example – add nodes

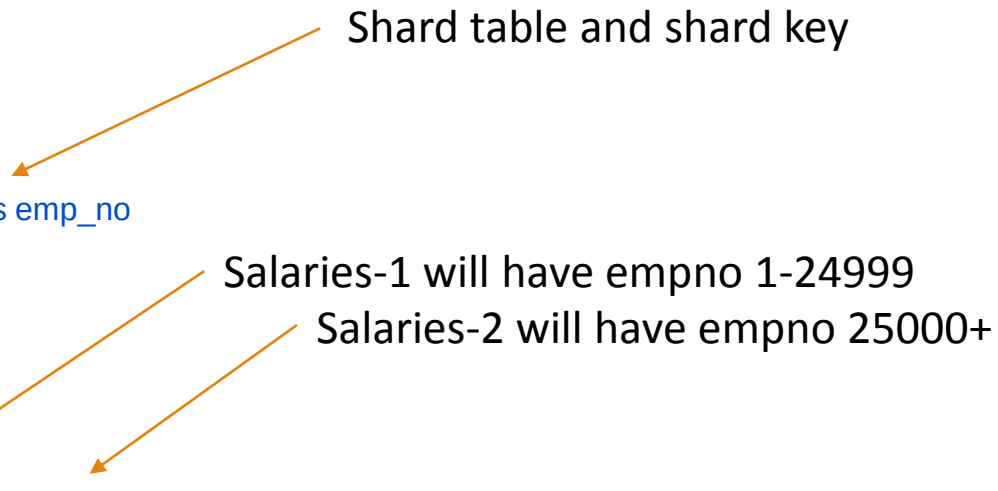
```
[vagrant@store ~]$ mysqlfabric group add salaries-global node1:3306
Procedure :
{ uuid = 0d0f657c-9304-4e3f-bf5b-a63a5e2e4390,
finished = True,
success = True,
return = True,
activities =
}
[vagrant@store ~]$ mysqlfabric group add salaries-1 node2:3306
Procedure :
{ uuid = b0ee9a52-49a2-416e-bdfd-eda9a384f308,
finished = True,
success = True,
return = True,
activities =
}
[vagrant@store ~]$ mysqlfabric group add salaries-2 node3:3306
Procedure :
{ uuid = ea5d8fc5-d4f9-48b1-b349-49520aa74e41,
finished = True,
success = True,
return = True,
activities =
}
```

MySQL Fabric Example – promotions

```
[vagrant@store ~]$ mysqlfabric group promote salaries-global
Procedure :
{ uuid = 5e764b97-281a-49f0-b486-25088a96d96b,
finished = True,
success = True,
return = True,
activities =
}
[vagrant@store ~]$ for i in 1 2; do mysqlfabric group promote salaries-$i; done
Procedure :
{ uuid = 7814e96f-71d7-4865-a278-cb6ed32a2d11,
finished = True,
success = True,
return = True,
activities =
}
Procedure :
{ uuid = cd30e9a9-b9ea-4b2d-a8ae-5e70f22363d6,
finished = True,
success = True,
return = True,
activities =
}
```

MySQL Fabric Example – add shards

```
[vagrant@store ~]$ mysqlfabric sharding create_definition RANGE salaries-global
Procedure :
{ uuid = fffcbb5f-24c6-47a2-9348-f1d810c8ef2f,
finished = True,
success = True,
return = 1,
activities =
}
[vagrant@store ~]$ mysqlfabric sharding add_table 1 employees.salaries emp_no
Procedure :
{ uuid = 8d0a3c51-d543-49a6-b47a-36a4ab499ab4,
finished = True,
success = True,
return = True,
activities =
}
[vagrant@store ~]$ mysqlfabric sharding add_shard 1 "salaries-1/1, salaries-2/25000" --state=ENABLED
Procedure :
{ uuid = 2585a5ea-a097-44a4-89fa-a948298d0595,
finished = True,
success = True,
return = True,
activities =
}
```



MySQL Fabric – Modify your app

- Must modify application to work with the framework
  - You must select a READ/WRITE or READ ONLY type of connection for example
  - You must specify the shard key in a function call before you run SQL
  - **fcnx.set_property**(**tables**=["employees.salaries"], **key**=emp_no, **mode**=fabric.MODE_READONLY)

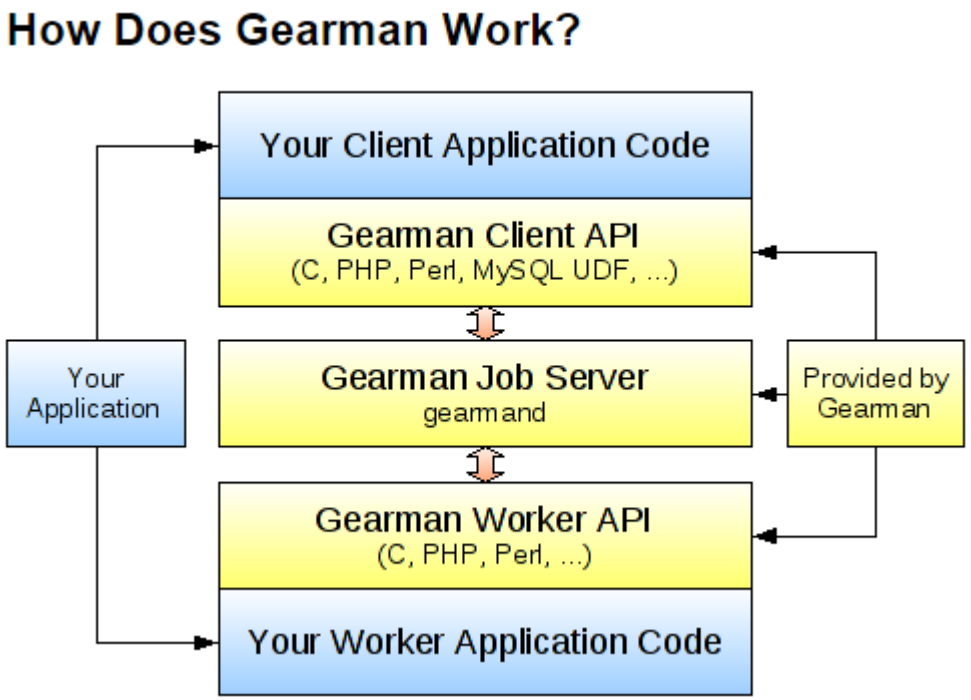
```
emp_no = random.randint(1,50000)
salary = random.randint(1,200000)
from_date = from_min + datetime.timedelta(seconds=random.randint(0, int((to_max - from_min).total_seconds())))
to_date = from_min + datetime.timedelta(seconds=random.randint(0, int((to_max - from_min).total_seconds())))
fcnx.set_property(tables=["employees.salaries"], key=emp_no, mode=fabric.MODE_READWRITE)
cur = fcnx.cursor()
cur.execute("insert into employees.salaries (emp_no,salary,from_date,to_date) values (%s, %s, %s, %s)",(emp_no,salary,from_date,to_date))
```

Shard-Query

- Shard-Query
  - Written in PHP, part of Swanhart- Tools
  - Good for most OLAP applications
  - Uses Gearman for parallelism
  - Stores configuration in a MySQL database schema
  - Works with any MySQL compatible database(MySQL 5.6 or MariaDB 10 recommended)

What is Gearman?

- Lightweight job manager
- Gearman is an anagram for manager
- Job server listens for jobs, hands them off to workers. Workers execute the job and optionally return results to the client.
- Shard-Query uses Gearman workers to execute SQL and insert the results into a coordination table
- Also includes workers for loading and for executing queries through the Gearman API



Shard-Query

- Designed for OLAP
  - It can run queries *in parallel*
  - Each machine can execute the query independently
  - Use MySQL partitioning + MySQL 5.6/MariaDB 10 for even more parallelism
  - Can be used on a single server with partitioning too

- But OLAP focus means it is too slow for OLTP
  - Has to make multiple round trips to answer a single query
  - Always has to use a temporary table
  - Response time better geared for analytics (OLAP) (.025 or so seconds minimum latency)

Shard-Query (cont)

- 4 Available interfaces
  - Transparent proxy LUA script for MySQL Proxy, which should be friendly to most apps
  - Also has an HTTP interface, a Gearman interface and a PHP OO interface (see run_query.php)
  - The HTTP Interface allows configuration of Shard-Query, or use command line tools
- Again, designed for OLAP
  - Can do select count(*) from sharded_table;
  - Creates a “virtual schema” from the databases defined in the configuration
  - Has basic support for window functions
  - Works great with star schema

How to set up Shard-Query

1. Create a bootstrap.ini file as a copy of example_bootstrap.ini
2. Specify the config options for Shard-Query and the list of shards in the .ini
3. Execute install_config_repo.php and follow on-screen prompts
4. Execute setup_virtual_schema.php –ini=bootstrap.ini
5. Make sure gearmand is running on the port you specified during setup
6. Start Shard-Query Gearman workers:
    1. cd swanhart-tools/shard-query/bin; ./start_workers

Shard-Query: Cross-shard aggregation

- Shard-Query supports treating the whole set of databases as one virtual schema
- Thus queries like select count(*) from sharded_table work!
  - Works by running a SELECT COUNT(*) from sharded_table on **EACH SHARD AUTOMATICALLY** *in parallel*
  - Each COUNT(*) is combined with SUM() to produce the final output
  - COUNT(DISTINCT …) and all other aggregation functions are supported too
  - Custom aggregate functions are possible too

Shard-Query Partition for parallelism

- Shard-Query examines the tables in use by the query and determines if they are partitioned
- If so, a parallel plan is created
- Each partition will be examined individually
- All tables must be partitioned identically

Shard-Query: “explain” plan

select count(distinct LO_OrderDateKey) from lineorder;
Shard-Query optimizer messages:
  * The following projections may be selected for a UNIQUE CHECK on the storage node operation:
expr_2679884247
SQL TO SEND TO SHARDS:
Array
(
   [0] => SELECT LO_OrderDateKey AS expr_2679884247
FROM lineorder PARTITION(p0) AS `lineorder` WHERE 1=1   GROUP BY expr_2679884247
…
   [6] => SELECT LO_OrderDateKey AS expr_2679884247
FROM lineorder PARTITION(p6) AS `lineorder` WHERE 1=1   GROUP BY expr_2679884247
   [7] => SELECT LO_OrderDateKey AS expr_2679884247
FROM lineorder PARTITION(p7) AS `lineorder` WHERE 1=1   GROUP BY expr_2679884247
)
SQL TO SEND TO COORDINATOR NODE:
SELECT COUNT(distinct expr_2679884247) AS ` count `
FROM `aggregation_tmp_95079490`

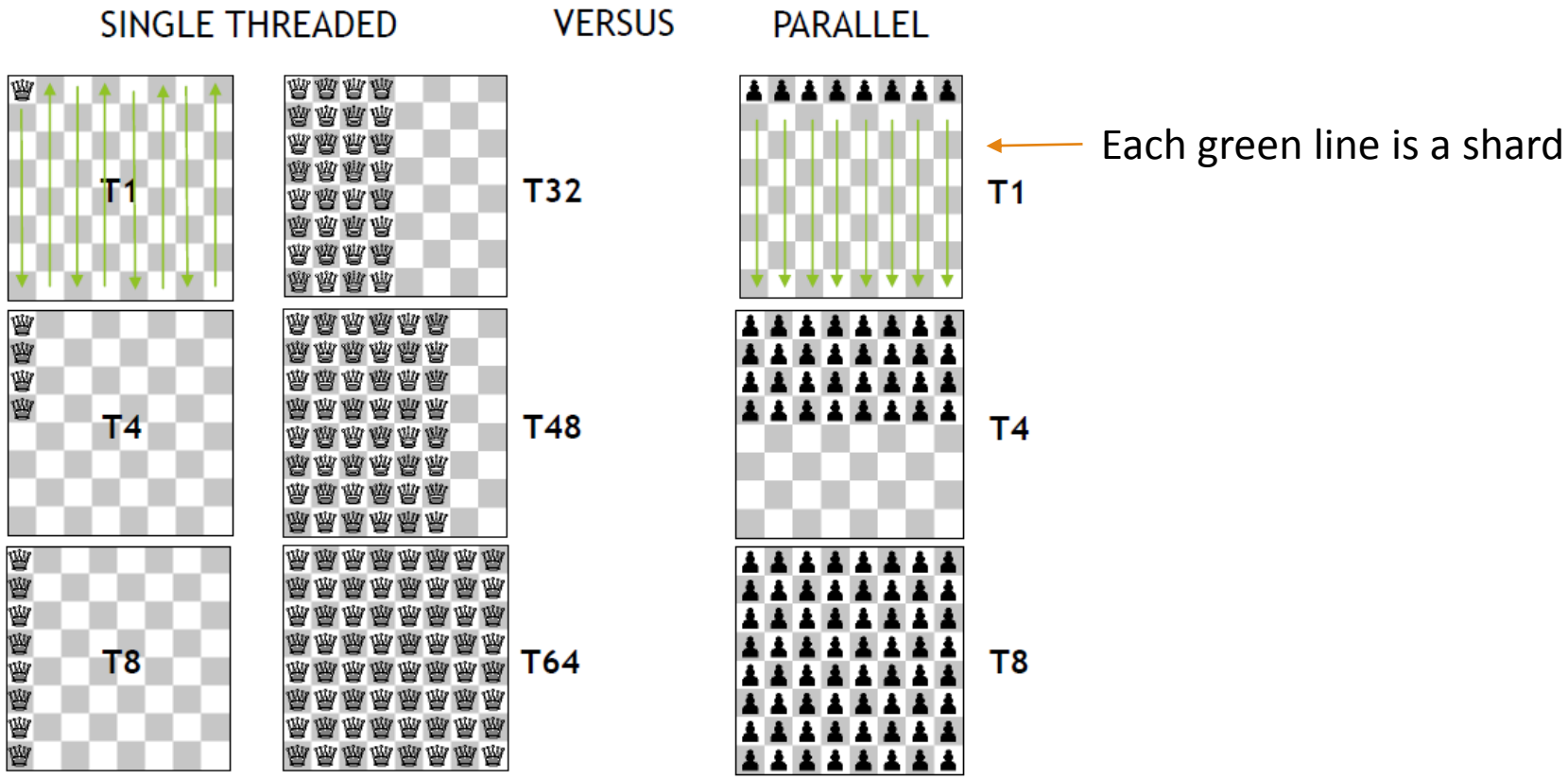
Array
(
   [count] => 2433
)
1 rows returned
Exec time: 1.2079381275177

One query per partition (8 total) (map)

“pushed” GROUP BY for COUNT(DISTINCT …)

Final aggregation (reduce)

Shard- Query : Query Parallelism



Shard-Query: Window Functions

- Window functions give a way for a query to access other rows in the resultset of the query
- This is like a self join
- Shard-Query supports window functions that stand alone
  - That is you can't multiply a window function by something unless you use a subquery in the from clause, eg:

```
SELECT ss*4 from (SELECT depname,salary, sum(salary) OVER (partition by depname order by salary) as ss FROM empsalary) sq;
```

  - Window functions are not allowed in the ORDER BY clause

Shard-Query: Window func running sum

```
[justin@localhost bin]$ php ./run_query < /tmp/test/test4.sql
Array
(
   [depname] => develop
   [salary] => 4200
   [ss] => 8400
)
Array
(
   [depname] => develop
   [salary] => 4200
   [ss] => 8400
)
Array
(
   [depname] => develop
   [salary] => 4500
   [ss] => 17400
)
Array
(
   [depname] => develop
   [salary] => 4500
   [ss] => 17400
)
...
20 rows returned
Exec time: 0.049021005630493
```

```
SELECT depname,salary, sum(salary) OVER (partition by depname order by salary) as ss
FROM empsalary;
```

Shard-Query: Parallel Loader

- Set up loader nodes to distribute the load over multiple machines
- Distributed load from a network filesystem, S3 or HTTP
- Simply use LOAD DATA INFILE to initiate the loader
  - If using S3 or HTTP, optionally change CHUNKSIZE to 128M

Tips for Sharding big data

- Try to stick to one big table for detailed measurements (fact table)
- Keep unsharded tables small if possible, as they must be duplicated on all shards
- Pick a shard key that is evenly distributable, like an invoice date or a customer number
- With Shard-Query, use the directory mapper if you want to split shards
- With Fabric use range mapper if you want to split shards
- Partition all your shards identically to take full advantage of Shard-Query parallelism