

17 Things Developers should know about Databases

Peter Zaitsev
CEO, Percona
Ryazan, Russia

November 5th, 2019



Who Are you ?

**More
Developer ?**

More OPS ?

Ops

Focused on Database Only

Generalist

Programming Language

- What Programming Languages does your team use ?

Devs vs Ops

DevOps suppose to have solved it but tension is still common between Devs and Ops

Especially with Databases which are often special snowflake

Especially with larger organizations

Large Organizations

- Ops vs Ops have conflict too

Devs vs Ops Conflict

- **Devs**

- Why is this stupid database always the problem.
- Why can't it just work and work fast

- **Ops**

- Why do not learn schema design
- Why do not you write optimized queries
- Why do not you think about capacity planning

Database Responsibility

- Shared Responsibility for Ultimate Success

Top Recommendations for Developers

Learn Database Basics

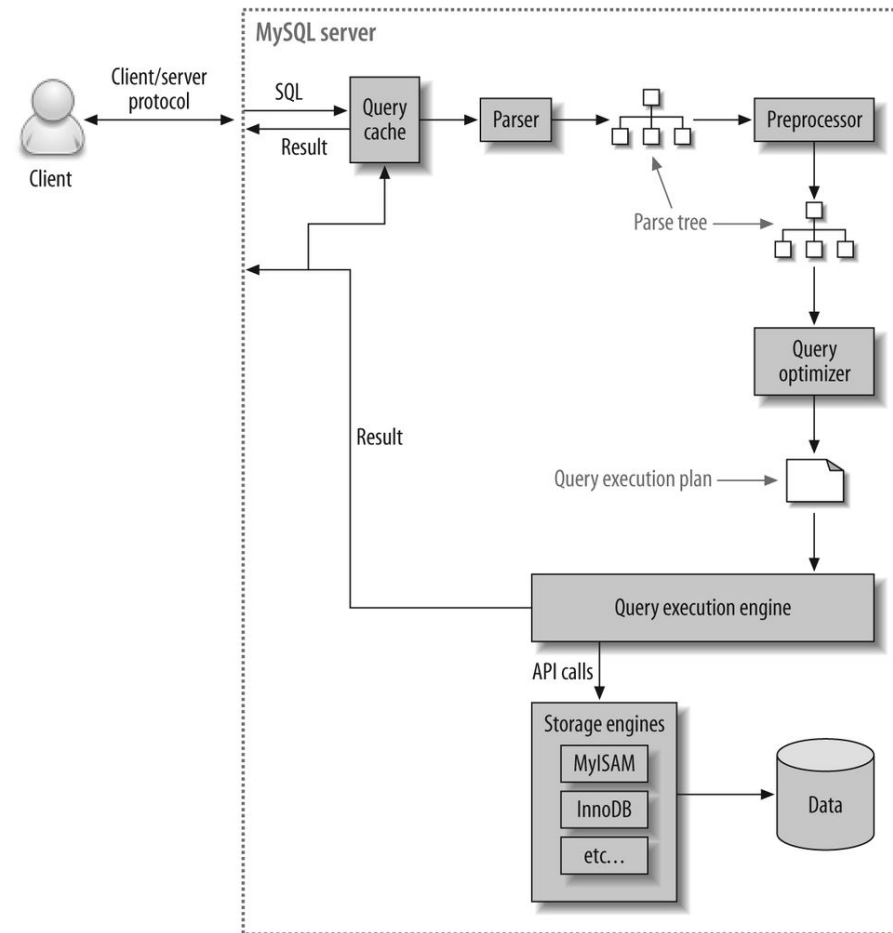
You can't build great database powered applications if you do not understand how databases work

Schema Design

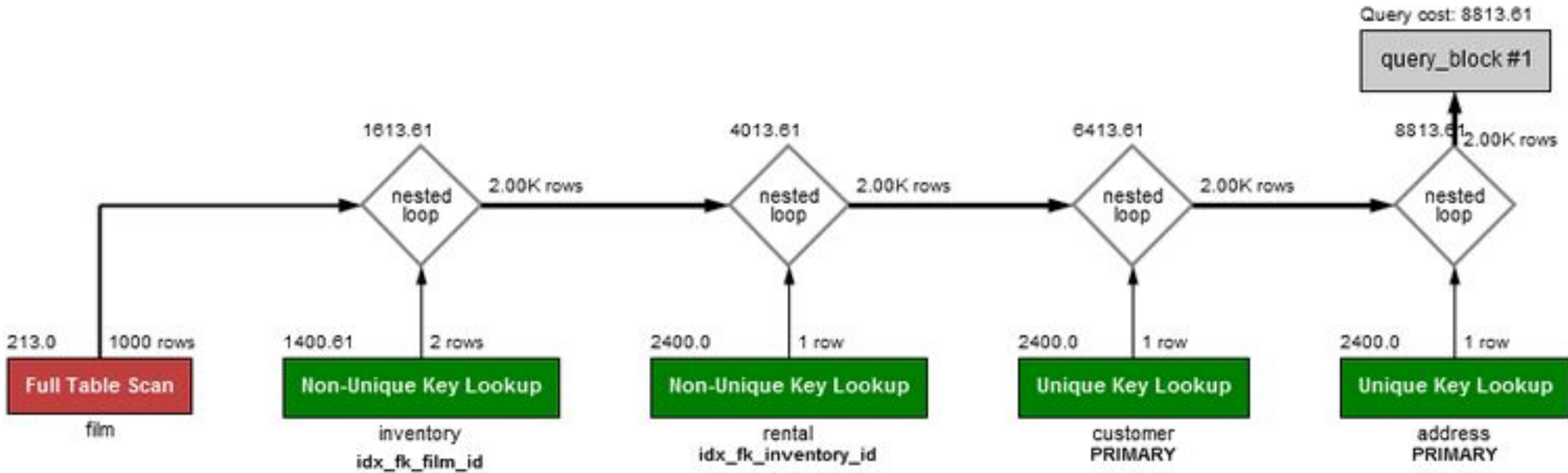
Power of the Database Language

How Database Executes the Query

Query Execution Diagram

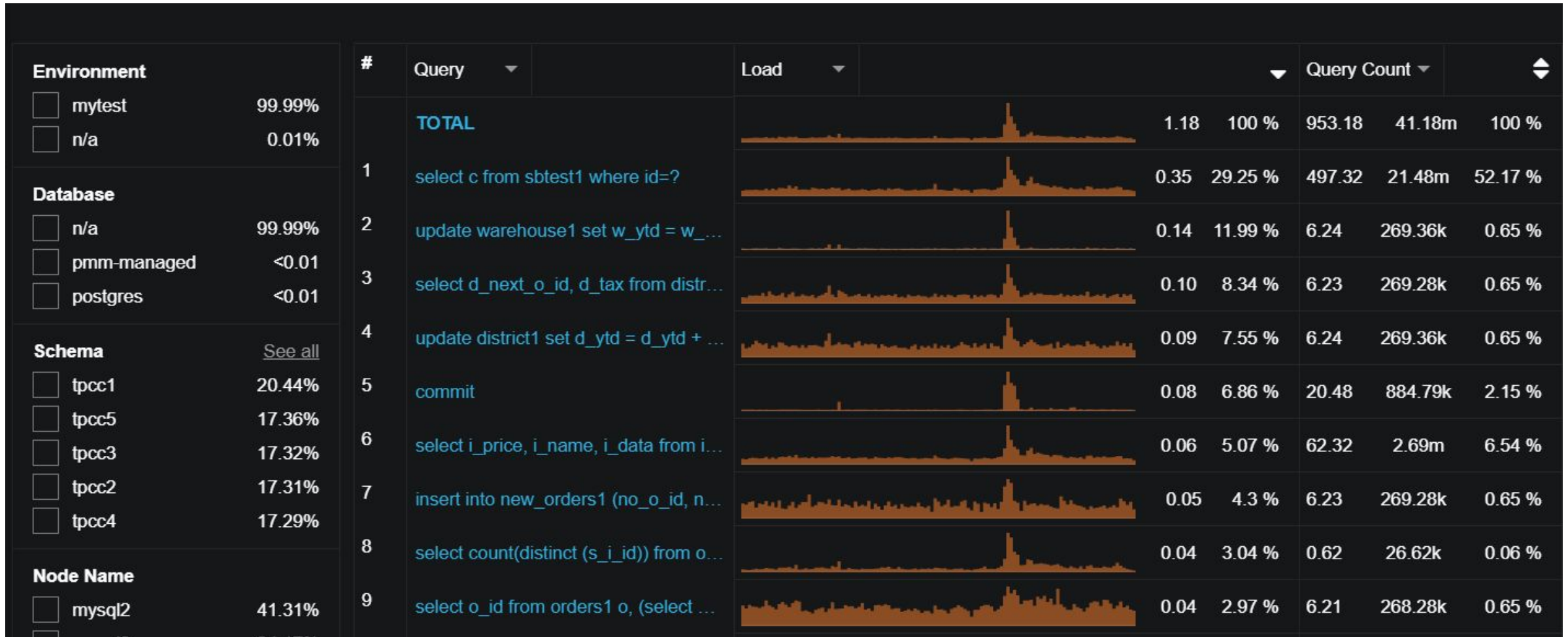


EXPLAIN

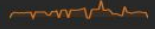




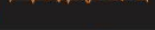
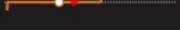
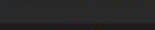
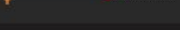
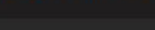
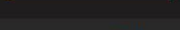

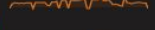

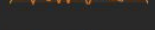

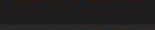
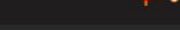
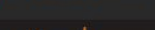
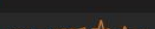



<https://dev.mysql.com/doc/refman/8.0/en/execution-plan-information.html>

Which Queries are Causing the Load



Why Are they Causing this Load

SELECT sbtest			737F39F04B198EF6	
Metrics			Query first seen: ☉ Aug 3, 2017 1:55 PM ☿ Last seen: ☉ Today at 9:46 AM	
Metrics	Rate/Sec	Sum	Per Query Stats	
Query Count	104.05 (per sec) 	374.58 k 4.27% of total		
Query Time	19.00 load 	18:59:56 29.73% of total	183.66 ms avg	
Lock Time	0.11 (avg load) 	0:06:42 1.35% of total 0.61% of query time	1.13 ms avg	
InnoDB IO Read Wait	0.61 (avg load) 	0:36:44 9.10% of total 3.38% of query time	6.20 ms avg	
InnoDB Read Ops	52.35 (per sec) 	188.45 k 7.62% of total	0.00 avg	
InnoDB Read Bytes	857.64 KB (per sec) 	3.09 GB 7.62% of total 16.38 KB avg io size	8.22 KB avg	
InnoDB Distinct Pages	-	-	4.69 avg	
Rows Sent	10.41 k (per sec) 	37.46 m 30.52% of total	100.00 avg	
Bytes Sent	1.30 MB (per sec) 	4.67 GB 30.78% of total 124.71 Bytes bytes/row	12.47 KB avg	
Rows Examined	1.14 m (per sec) 	4.11 b 39.17% of total 109.79 per row sent	10.47 k avg	
External Sorts (Filesort)	104.05 (per sec) 	374.58 k 49.93% of total 100.00% of queries	-	
Full Table Scans	0.01 (per sec) 	40.00 0.17% of total 0.01% of queries	-	
Queries Requiring Tmp Table In Memory	104.05 (per sec) 	374.58 k 95.17% of total 100.00% of queries	-	

How to Improve their Performance

Example

```
SELECT DISTINCT c
FROM sbtest1
WHERE id
      BETWEEN 5559
      AND 5658
ORDER BY c
```

CREATE

```
CREATE TABLE `sbtest1` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `k` int(10) unsigned NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB AUTO_INCREMENT=100000001 DEFAULT
```

JSON

Expand All

```
-{
  "query_block": -{
    "select_id": 1,
    "cost_info": +{...},
    "ordering_operation": -{
      "using_filesort": false,
      "duplicates_removal": -{
        "using_temporary_table": true,
        "using_filesort": true,
        "cost_info": +{...},
        "table": +{...}
      }
    }
  }
}
```

Check out PMM

<http://pmmdemo.percona.com>

PMM v 2 is now GA

How are Queries Executed ?

Single Threaded

Single Node

Distributed

Indexes

Indexes are Must

Indexes are Expensive

Capacity Planning

No Database can handle “unlimited scale”

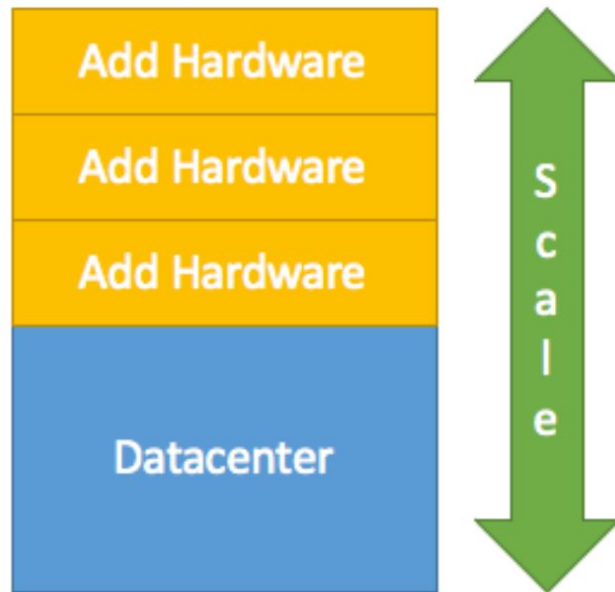
Scalability is very application dependent

Trust Measurements more than Promises

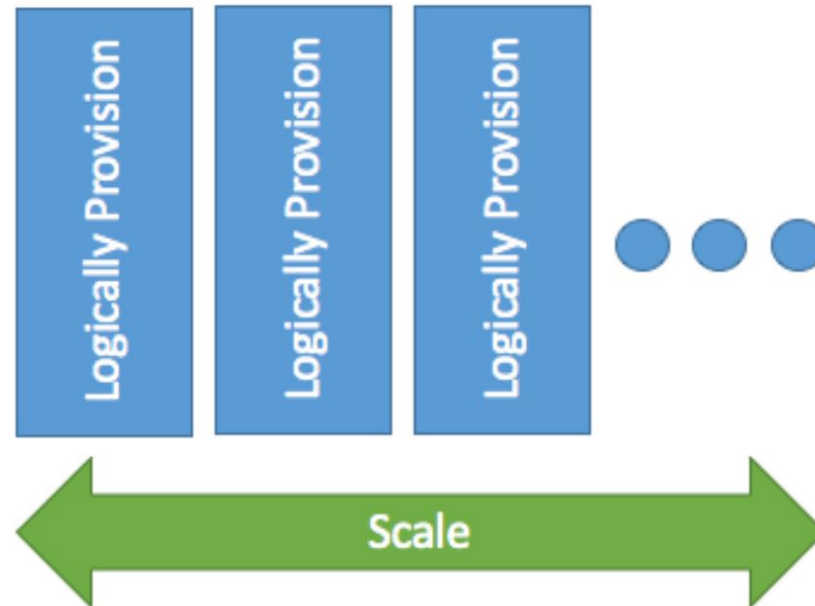
Can be done or can be done Efficiently ?

Vertical and Horizontal Scaling

Vertical Scaling



Horizontal Scaling



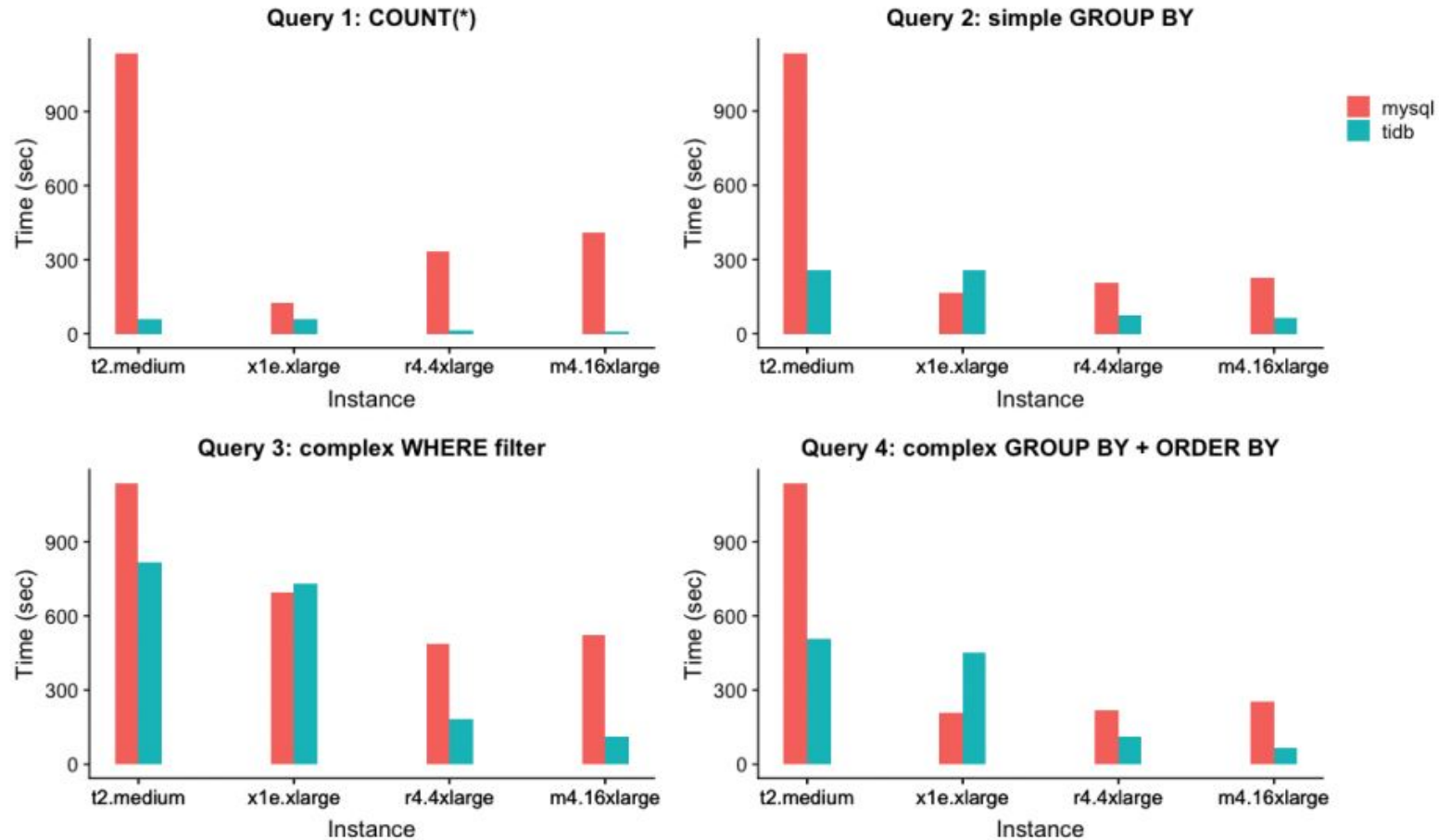
Scalable != Efficient

The Systems which promote a scalable can be less efficient

Hadoop, Cassandra, TiDB are great examples

By only the wrong thing you can get in trouble

TiDB Scalability (Single Node)



TiDB Efficiency

TiDB and MySQL - point selects - sysbench



Throughput != Latency

- If I tell you system can do 100.000 queries/sec would you say it is fast ?

Speed of Light Limitations

High Availability Design Choices

You want instant durable replication over wide geography or Performance ?

Understanding Difference between High Availability and Disaster Recovery protocols

Network Bandwidth is not the same as Latency

Also Understand

Connections to the database are expensive

Especially if doing TLS Handshake

Query Latency Tends to Add Up

Especially on real network and not your laptop

ORM (Object-Relational-Mapping)

Allows Developers to query the database without need to understand SQL

Can create SQL which is very inefficient

Learn SQL Generation “Hints”, Learn JPQL/HQL advanced features

Be ready to manually write SQL if there is no other choice

Do not Leave Transactions Open

Open Connection is rather inexpensive

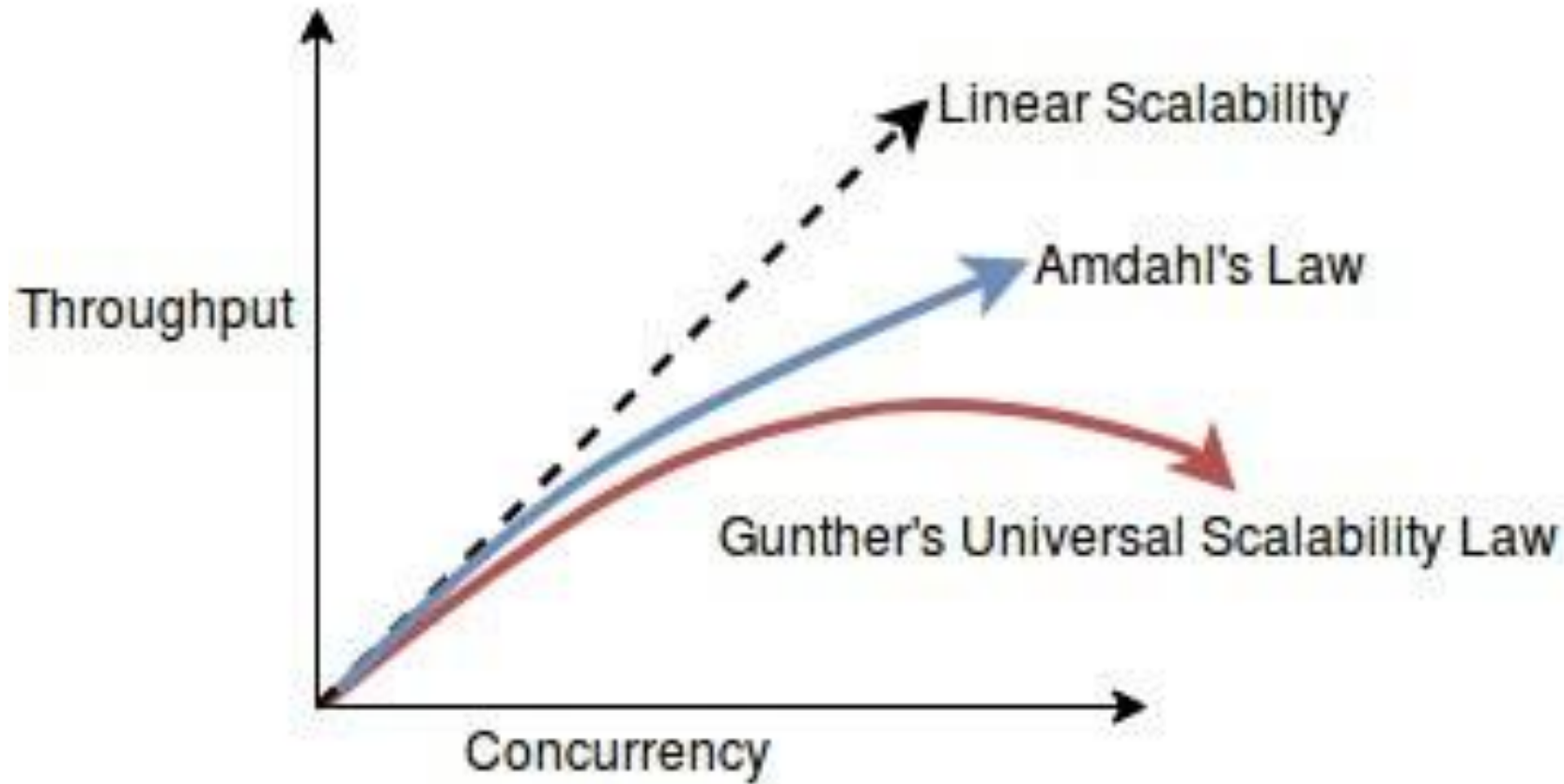
Transaction open for Long Time can get very expensive (even if it performed no writes)

Isolation Mode Matters

SET AUTOCOMMIT=0 - Any SELECT query will Open Transaction

COMMIT/ROLLBACK closes transaction

Understanding Optimal Concurrency



Queueing

Request Queueing is Normal

With requests coming at “Random Arrivals” some queueing will happen with any system scale

Should not happen too often or for very long

Queueing is “Cheaper” Close to the User

Benefits of Connection Pooling

**Avoiding
Connection
Overhead,
especially TLS/SSL**

**Avoiding using
Excessive Number
of Database
Connections**

**Multiplexing/Load
Management**

Configuring Connection Pool

Default and Maximum Connection Pool Size

Scaling Parameters

Combined Connection Pool Max Size should be smaller than number of connections database can support

Waiting for free connection to become available is OK

Law of Gravity

- Shitty Application at scale will bring down any Database

Scale Matters

Developing and Testing with Toy Database is risky

Queries Do not slow down linearly

The slowest query may slow down most rapidly

Memory or Disk

Data Accessed in memory is much faster than on disk

It is true even with modern SSDs

SSD accesses data in large blocks, memory does not

Fitting data in Working Set

Newer is not Always Faster

Upgrading to the new
Software/Hardware is not always
faster

Test it out

Defaults Change are often to blame

Upgrades are needed but not seamless

Major Database Upgrades often require application changes

Having Conversation on Application Lifecycle is a key

Character Sets

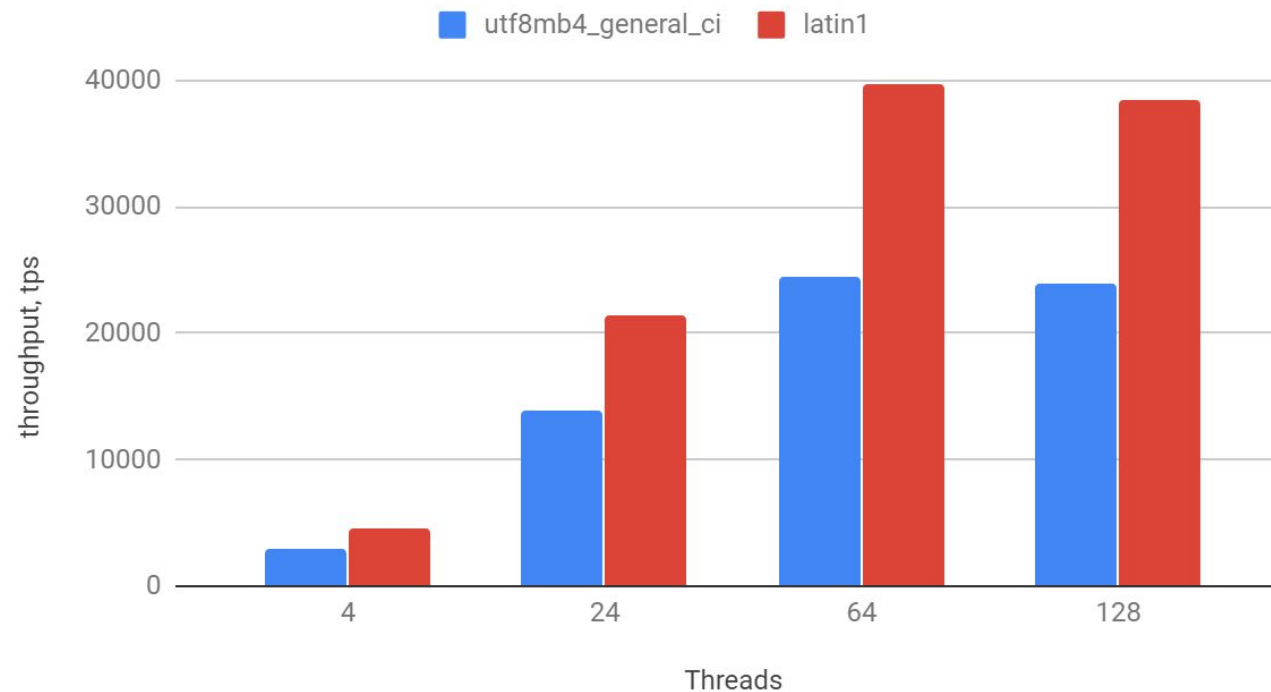
Performance Impact

Pain to Change

Wrong Character Set can cause Data Loss

Character Sets

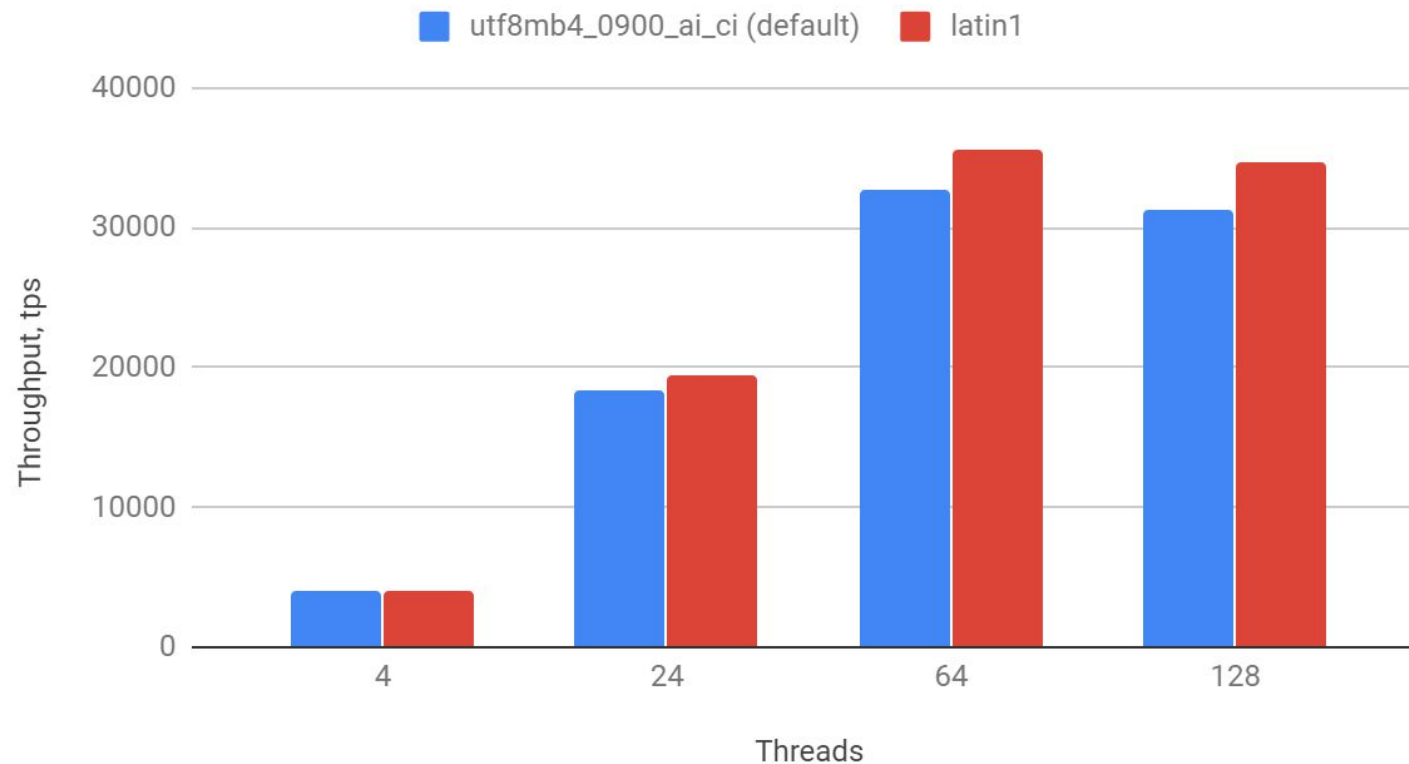
MySQL 5.7 utf8mb4_general_ci (default) and latin1



<https://per.co.na/MySQLCharsetImpact>

Less impact In MySQL 8

MySQL 8.0 utf8mb4_0900_ai_ci and latin1



Operational Overhead

Operations Take Time, Cost Money, Cause Overhead

10TB Database Backup ?

Adding The Index to Large Table ?

Distributed Systems

10x+ More Complicated

Better High Availability

Many Failure Scenarios

Test how application performs

Risks of Automation

Automation is Must

Mistakes can destroy database at scale

Security

Database is where the most sensitive data tends to live

Shared Devs and Ops Responsibility

What Else

- What Would you Add ?

Thank You!

Twitter: @percona @peterzaitsev
