Performance of popular open source databases
for monitoring applications

D.Kovalskyi, I.Sfiligoi, F.Wuerthwein, A.Yagil
University of California, San Diego

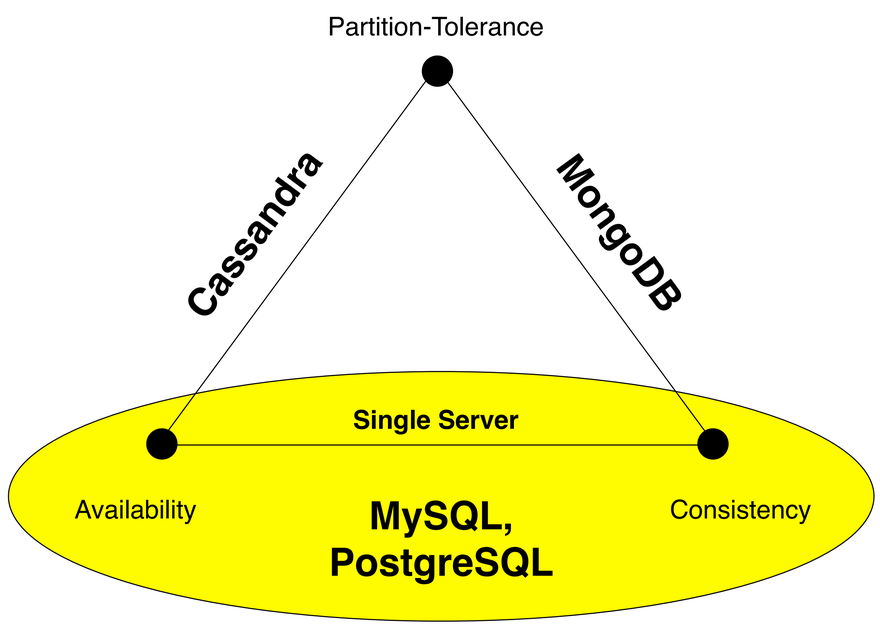
The Question

Problem: we need a database, but we don't know how much resources it will need to provide a reliable solution

Concern: can a single server handle the load or we have to plan for horizontal scaling?

Question: how well popular open source DBs perform on a single server. If it's likely that at some point we need to adopt a more scalable solution, should we start with it right away or we can get more done with traditional systems?

Overview



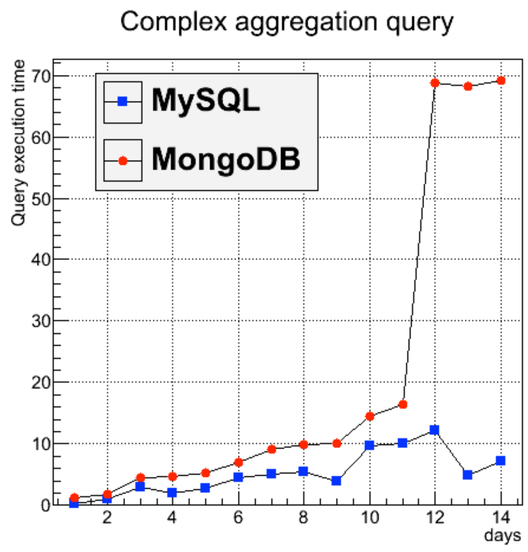
CAP Theorem: it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- **Consistency** (all nodes see the same data at the same time)
- **Availability** (a guarantee that every request receives a response about whether it was successful or failed)
- **Partition tolerance** (the system continues to operate despite arbitrary message loss or failure of part of the system)

Databases tested:

Relational	Non-relational
MySQL v5.5.32	MongoDB v2.2.4
PostgreSQL v9.2.4	Apache Cassandra v2.0

Results



First Test

- Results (flashed I/O cache)
  - MySQL (server1): 72 sec
    - MySQL (server2): 73 sec
  - PostgreSQL (server1): 62 sec
  - MongoDB (server1): 290 sec
    - MongoDB uses ~x2 more disk space and can be penalized by worse caching
    - Clearly there is a difference in performance though
  - Apache Cassandra failed
- Performance strongly depends on schema and data structure in use
  - All reasonable optimizations were used (indexing on many variables)

Hardware role

- Caching in RAM and SSD give the largest performance gain
- Total execution time with OS caching of I/O:
  - MySQL: 27 sec
  - PostgreSQL: 37 sec
  - MongoDB: 260 sec
- In-memory database solutions are becoming popular
  - IMDB allows for a better/more efficient design
  - Using MySQL **Memory** engine we haven't observed any performance gain compared with MyISAM + RAM based caching by OS
- CPU is mostly not an issue - more RAM and faster data-storage are more critical

Second Test	Total execution time	One query (median)	One query (95%)
MySQL (2 threads, server 1)	27.36 sec	0.01 sec	0.24 sec
MySQL (10 threads, server 1)	26.87 sec	0.07 sec	1.21 sec
MySQL (2 threads, server 2, remote connection)	47.63 sec	0.03 sec	0.41 sec
MySQL (10 threads, server 2, remote connection)	25.91 sec	0.08 sec	1.21 sec
PostgreSQL (2 threads, server 1)	15.17 sec	0.02 sec	0.10 sec
PostgreSQL (10 threads, server 1)	14.13 sec	0.10 sec	0.13 sec
MongoDB (2 threads, server 1)	77.78 sec	0.05 sec	0.80 sec
MongoDB (10 threads, server 1)	59.14 sec	0.12 sec	2.47 sec
Apache Cassandra	Failed	Failed	Failed

Apache Cassandra Issues

- Cassandra is not well suitable for dynamic aggregation operations
- A simple query to count number of records takes enormous amount of time due to need to scan all the "columns"
  - By default this operation is limited in CQL to the first 10000 entries selected
- Dynamic filtering is not expected either - one needs to allow filtering explicitly
- Selection must be formed with at least one "equal" statement on an indexed column

Test Procedure

Production System

- GlideMon - user job monitoring for distributed condor job submission system
- We have a working system based on a single MySQL server

Test Setup

- Snap-shot of the GlideMon database:
  - Total number of jobs: ~14 M
  - Last two weeks since the date DB was frozen:
    - Number of jobs: 2.2M
    - Number of tasks: 5 K
    - Number of users: 251
  - Database size: ~10-20GB

Hardware

1. Intel Core 2 Duo E8500 @ 3.17 GHz, 2 cores, 8GB RAM, RAID5, Fedora 18
2. Intel Xeon X5472 @ 3.00 GHz, 8 cores, 12GB RAM, HDD, Scientific Linux 5 (remote)

Test Queries

1. Complex single threaded aggregation query
   - Count number of analysis jobs in each state for each user within last N days
   - Slow query - materialized view in production
2. Simple select query for a number of simultanous users
   - List of tasks for a given user with number of jobs summary
   - Dynamic query - run it without caching

Conclusions

- Often it is not easy to predict if a database management system (DMS) for a new application would fit in a single server or a cluster maybe needed
  - DMS that performs well in both single node and multi nodes modes may be a good choice to avoid extra work for migration
    - **New generation of DMS promise good scalability, but how well they perform before we need to scale?**
- **Traditional relational** DMS perform the best in a single node mode
  - Lots of tools and documentation is available
  - PostgreSQL and MySQL show similar performance in most tests
- **New non-relational** DMS are designed with multi node mode in mind
  - **MongoDB** performed well in a single-node test.
    - The system is well spread, but tools are lacking
    - Comparable performance with RDMS, but in general is slower
      - **Can be a reasonable compromise when there is a reasonable chance that one server is not enough, but a cluster will be needed sooner than with RDMS**
  - Apache **Cassandra** failed to deliver what we need
    - Better matching of DMS to application would help (documentation is deficient)
      - Only after implementing a realistic database we found that the tool doesn't fit our needs
- The smaller the database - the easier is to find a proper tool and achieve adequate performance