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 do 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 can we 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:

<table>
<thead>
<tr>
<th>Relational</th>
<th>Non-relational</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL v9.5.32</td>
<td>MongoDB v2.2.4</td>
</tr>
<tr>
<td>PostgreSQL v9.2.4</td>
<td>Apache Cassandra v2.0</td>
</tr>
</tbody>
</table>

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
- Snapshot of 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 E8800 @ 3.17 GHz, 2 cores, 8 GB RAM, RAID5, Fedora 18
2. Intel Xeon X5472 @ 3.00 GHz, 8 cores, 12 GB RAM, H75, Scientific Linux 5 (x86_64)

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

Results

Complex aggregation query

Hardware role
- Caching in RAM and SSD give the largest performance gain
- Total execution time with OS caching of I/O:
  - MySQL (server2): 72 sec
  - MySQL (server2): 73 sec
  - PostgreSQL (server2): 62 sec
  - MongoDB (server2): 290 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

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 SQL to the first 10000 entries selected
- Dynamic filtering is not expected either - one needs to allow filtering explicitly
- Selection must be formatted with at least one “equal” statement on an indexed column

Second Test

<table>
<thead>
<tr>
<th>Database</th>
<th>Total execution time</th>
<th>One query (median)</th>
<th>One query (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL (2 threads, server 1)</td>
<td>27.36 sec</td>
<td>0.01 sec</td>
<td>0.24 sec</td>
</tr>
<tr>
<td>MySQL (10 threads, server 1)</td>
<td>26.87 sec</td>
<td>0.07 sec</td>
<td>1.21 sec</td>
</tr>
<tr>
<td>MySQL (2 threads, server 2, remote connection)</td>
<td>47.63 sec</td>
<td>0.05 sec</td>
<td>0.41 sec</td>
</tr>
<tr>
<td>MySQL (10 threads, server 2, remote connection)</td>
<td>25.91 sec</td>
<td>0.08 sec</td>
<td>1.21 sec</td>
</tr>
<tr>
<td>PostgreSQL (2 threads, server 1)</td>
<td>15.17 sec</td>
<td>0.02 sec</td>
<td>0.10 sec</td>
</tr>
<tr>
<td>PostgreSQL (10 threads, server 1)</td>
<td>14.13 sec</td>
<td>0.10 sec</td>
<td>1.13 sec</td>
</tr>
<tr>
<td>MongoDB (2 threads, server 1)</td>
<td>77.78 sec</td>
<td>0.05 sec</td>
<td>0.80 sec</td>
</tr>
<tr>
<td>MongoDB (10 threads, server 1)</td>
<td>59.14 sec</td>
<td>0.12 sec</td>
<td>2.47 sec</td>
</tr>
<tr>
<td>Apache Cassandra</td>
<td>Failed</td>
<td>Failed</td>
<td>Failed</td>
</tr>
</tbody>
</table>

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