Engineering Robust Server Software

Scalability
Other Scalability Issues

- Database
- Load Testing
Databases

• Most server applications use databases
  • Very complex pieces of software
  • Designed for scalability
• …but how well depends on what you are doing with them…
Databases and Concurrency

- How do databases handle concurrency?
  - Could use locks, but… what we learned about those?
Databases and Concurrency

• How do databases handle concurrency?
  • Could use locks, but… what we learned about those?
• Postgres (and many others): MVCC
  • Multi-version concurrency control
  • Basically, the DB keeps multiple versions
  • Ensures **consistency** based on **transaction isolation level**
Serializability

- In 650, you learned about serializability…
  - Who can remind us what it is?
  - What are its benefits?
Serializability

- In 650, you learned about serializability…
  - Who can remind us what it is?
  - What are its benefits?
- Does this sound similar to any other ideas we have learned recently?
  - If so, what conclusions might you draw about performance?
  - What do you think we might do?
Isolation Levels

- Serializable
  - As in 650
  - Nothing unexpected
- Repeatable Read
  - Can have **phantom reads**
- Read Committed (default in Postgres)
  - Can have **non-repeatable reads** (+phantoms)
- Read Uncommitted
  - Can have **dirty reads** (+non-repeatable +phantoms)
## Non-Repeable Read

<table>
<thead>
<tr>
<th>id</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>66</td>
</tr>
<tr>
<td>67</td>
<td>128</td>
</tr>
<tr>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td>456</td>
<td>1</td>
</tr>
</tbody>
</table>

Values within a row change between reads

```sql
SELECT count FROM tbl WHERE id = 42;
```

99

```sql
UPDATE tbl SET count = 66 WHERE id = 42;
```

COMMIT;

```sql
SELECT count FROM tbl WHERE id = 42;
```

66
Phantom Read

<table>
<thead>
<tr>
<th>id</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>66</td>
</tr>
<tr>
<td>67</td>
<td>128</td>
</tr>
<tr>
<td>99</td>
<td>32</td>
</tr>
<tr>
<td>456</td>
<td>1</td>
</tr>
</tbody>
</table>

Set of rows in a query change between reads

SELECT * from tbl WHERE count < 10;

(99,0)
(456,1)

UPDATE tbl SET count = 32 WHERE id = 99;
COMMIT;

SELECT * from tbl WHERE count < 10;

(456,1)
### Dirty Read

<table>
<thead>
<tr>
<th>id</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>66</td>
</tr>
<tr>
<td>67</td>
<td>128</td>
</tr>
<tr>
<td>99</td>
<td>32</td>
</tr>
<tr>
<td>456</td>
<td>77</td>
</tr>
</tbody>
</table>

Read from uncommitted transaction

UPDATE tbl SET count = 77 WHERE id = 456;

SELECT count from tbl WHERE id = 456;

77

ROLLBACK;
Isolation Levels: Postgres

- **Serializable**  
  - As in 650  
  - Nothing unexpected
- **Repeatable Read**  
  - Can have **phantom reads**
- **Read Committed** (default in Postgres)  
  - Can have **non-repeatable reads** (+phantoms)
- **Read Uncommitted**  
  - **Not actually available: upgraded to Read Committed**  
  - Can have **dirty reads** (+non-repeatable +phantoms)

Can throw exns for violations
More On Isolation

- For more on isolation in Postgres, see
  - [https://www.postgresql.org/docs/9.5/static/transaction-isolation.html](https://www.postgresql.org/docs/9.5/static/transaction-isolation.html)
Query Performance

- Many things can affect query performance
  - Complicated topic..
- But how can you gain insight into what is going on?
- Can you do anything to improve it?
Explain

explain select * from grades where grade < 62;

QUERY PLAN

Seq Scan on grades  (cost=0.00..494.80 rows=55 width=35)
Filter: (grade < 62)

Want to know how your query is going to be executed?

- Ask Postgres to EXPLAIN it
- https://www.postgresql.org/docs/9.5/static/sql-explain.html
Seq Scan?

• Sequential Scan = linearly examine each element.
  • Sound good?
Seq Scan?

- Sequential Scan = linearly examine each element.
  - Sound good?
- No! We can do better..
- How?
  - Ask postgres to build an index

```sql
CREATE index ON grades (grade);
explain select * from grades where grade <62;
```

---

**QUERY PLAN**

```
Bitmap Heap Scan on grades  (cost=4.71..128.27 rows=55 width=35)
  Recheck Cond: (grade < 62)
  ->  Bitmap Index Scan on grades_grade_idx  (cost=0.00..4.70 rows=55 width=0)
      Index Cond: (grade < 62)
```
Indexes

• Why not index everything?
  • Cost to maintain index
  • Building=expensive: do before deploying
• Build indexes that are useful for the queries you need
• See
  • https://www.postgresql.org/docs/9.5/static/sql-createindex.html
Load Testing

- All of this discussion of scalability..
  - How do we know how well we are doing?
  - Well, then again, how do you know how you are doing for anything?
Load Testing

- All of this discussion of scalability..
  - How do we know how well we are doing?
  - Well, then again, how do you know how you are doing for anything?
- Test your code!
  - What is the purpose of testing?
  - What is a successful test case?
Load Testing

- All of this discussion of scalability..
  - How do we know how well we are doing?
  - Well, then again, how do you know how you are doing for anything?
- Test your code!
  - What is the purpose of testing? Discover problems
  - What is a successful test case? One that shows a problem
- So for load testing, what is our criteria for success?
Load Testing

- All of this discussion of scalability..
  - How do we know how well we are doing?
  - Well, then again, how do you know how you are doing for anything?
- Test your code!
  - What is the purpose of testing? **Discover problems**
  - What is a successful test case? **One that shows a problem**
- So for load testing, what is our criteria for success?
  - Identify performance/scalability problems
Load Testing

• Rule 1: generate a lot of load
  • Sending one request, then another serially? Not enough
  • Need multiple programs/threads/systems generating load
Load Testing

• Rule 1: generate a lot of load
  • Sending one request, then another serially? Not enough
  • Need multiple programs/threads/systems generating load

• Rule 2: system needs significant data to start
  • Why?
Load Testing

- Rule 1: generate a lot of load
  - Sending one request, then another serially? Not enough
  - Need multiple programs/threads/systems generating load
- Rule 2: system needs significant data to start
Data Size Matters

- Suppose you have 10 rows in a table
  - Does indexing matter?
  - What level of the memory hierarchy do you hit?
Data Size Matters

• Suppose you have 10 rows in a table
  - Does indexing matter? **No (probably minorly counter productive)**
  - What level of the memory hierarchy do you hit? **L1 cache**

• Suppose you have 10,000,000,000 (10B) rows in a table
  - Does indexing matter?
  - What level of the memory hierarchy do you hit?
Data Size Matters

- Suppose you have 10 rows in a table
  - Does indexing matter? No (probably minorly counter productive)
  - What level of the memory hierarchy do you hit? L1 cache
- Suppose you have 10,000,000 (10M) rows in a table
  - Does indexing matter? Yes
  - What level of the memory hierarchy do you hit? Disk
Data Size Matters

- Suppose you have 10 rows in a table
  - Does indexing matter? No (probably minorly counter productive)
  - What level of the memory hierarchy do you hit? L1 cache
- Suppose you have 10,000,000 (10M) rows in a table
  - Does indexing matter? Yes
  - What level of the memory hierarchy do you hit? Disk
- How different are these performance characteristics?
  - Bandwidth?
  - Latency?
Load Testing

• Rule 1: generate a lot of load
  • Sending one request, then another serially? Not enough
  • Need multiple programs/threads/systems generating load

• Rule 2: system needs significant data to start
  • Performance characteristics depend on size

• Rule 3: data needs to have reasonable characteristics
  • Match values/conditions on values of real data
  • Why?
Data Must Be Realistic

- Suppose you run the query
  - SELECT * from whatever WHERE x < 100 AND x > 50;
  - You have only 5 in that range in your test data
  - Your real data ends up with 5,000,000 in that range
- How similar will your performance characteristics be?
Load Testing

- Rule 1: generate a lot of load
  - Sending one request, then another serially? Not enough
  - Need multiple programs/threads/systems generating load
- Rule 2: system needs significant data to start
  - Performance characteristics depend on size
- Rule 3: data needs to have reasonable characteristics
  - Match values/conditions on values of real data
- Rule 4: mix and match many combinations of operations in parallel
  - Why?
Mix and Match Operation

- Obvious: we want to ensure each operation done at least once
  - Just like statement coverage.
- But why mix and match them?
Mix and Match Operation

- Obvious: we want to ensure each operation done at least once
  - Just like statement coverage.
- But why mix and match them?
  - Different resource usage: cache, bandwidth, ….
  - Different pairings = different resource contention
    - And different DB contention
    - Read by itself vs waiting for a write to commit
What Is "Passing"?

- Ok, so you follow all my rules…
- Make your test cases…
- Run them….
- How do you know if you "passed" the test?
What Is "Passing"?

- Ok, so you follow all my rules…
- Make your test cases…
- Run them….
- How do you know if you "passed" the test?
  - …It depends…. (oh man, I love that answer).
Different Goals

• We might have different goals:
  • Can our system handle the demand from X users?
    • e.g., can DukeHub handle registration?
  • Did we just make it better?
    • e.g., we think we optimized the code, did it really improve?
  • Does our system scale sufficiently with more hardware?
    • Note: requires definition of "sufficiently"
  • Does our system degrade gracefully with more load?
    • Note: requires definition of "gracefully"
Can We Handle Demand of X Users

- Load test with loads that try to mimic X Users
  - May not be hitting system as hard as you possibly can
  - Probably want to add some margin for error
- Measure latencies of requests
  - See how many are within tolerable range
    - Define tolerable?
  - Quite possibly in terms of % guarantees
    - e.g., 99% of requests took less than 500 usec.
Did We Make It Better?

• You do something to your code to improve scalability
  • (Add an index, replace a locked DS with a LF one, …)
  • How do you know it is actually better?
    • Side note: how do you convince your boss that
      • (a) it was worth your time
      • (b) he/she should give you a raise for your hardcore hacking?
Did We Make It Better?

• You do something to your code to improve scalability
  • (Add an index, replace a locked DS with a LF one, …)
  • How do you know it is actually better?
  • Run the old, run the new, measure performance -> see which wins
    • Is it that simple?
Did We Make It Better

Time (lower = better)

- Test1
- Test2
- Test3
- Test4
- Test5

Old
New
Did We Make It Better?

• You do something to your code to improve scalability
  • (Add an index, replace a locked DS with a LF one, …)
  • How do you know it is actually better?
  • Run the old, run the new, measure performance -> see which wins
    • Different tests may show different results
    • Different metrics may show different results
      • E.g., slower with this hw, but more scalable with more hw
Is Our System Scalable "Enough"?

- What is scalable enough?
  - That also depends…
Is Our System Scalable "Enough"?

- What is scalable enough?
  - That also depends…

- How much hardware do we need to add for X more users?
  - Combines two notions of scalability we saw earlier
  - Why does this make business sense?
Is Our System Scalable "Enough"?

- What is scalable enough?
  - That also depends...

- How much hardware do we need to add for X more users?
  - Combines two notions of scalability we saw earlier
  - Why does this make business sense?
    - Compute costs money, users bring money -> profitable?
    - Think Cloud Computing
Wrap Up

• Today
  • Databases
  • Load Testing

• Next Time:
  • Tyler will come talk about IO Scalability

• Next Week:
  • Lock Free Data Structures
  • Then guest lectures.