Pocket Data
The Case for TPC-MOBILE

Oliver Kennedy, Jerry Ajay, Geoff Challen, Lukasz Ziarek

http://odin.cse.buffalo.edu/research/astral
Big Data!
Big Data!
Big Data!

• GB, TB or PB of data!

• Hundreds of thousands of updates per second

• Thousands of nodes computing together!

• “Virtually” infinite resources!
Big Data!
Big Data!

TPC-C  TPC-H  SSB
TPC-E  TPC-DI  TPC-DS
YCSB
Big Data!

What about other types of databases?
The average smartphone processes almost 180 thousand queries per day.

That’s about **2 queries per second**.
2 Queries per Second
2 Queries per Second

• Is this Big Data? **No!**
2 Queries per Second

• Is this Big Data? No!

• Is this Important?
2 Queries per Second

- Is this Big Data? **No!**
- Is this Important?
  
  - **Multi-Tenancy**: The phone is more than just a DB.
2 Queries per Second

- Is this Big Data? No!
- Is this Important?
  - **Multi-Tenancy**: The phone is more than just a DB.
  - **Power**: 1-2 days of battery life under ideal circumstances.
2 Queries per Second

• Is this Big Data? No!

• Is this Important?
  
  • **Multi-Tenancy**: The phone is more than just a DB.
  
  • **Power**: 1-2 days of battery life under ideal circumstances.
  
  • **It's Everywhere**: Odds are that your phone is running some queries right now!
2 Queries per Second

• Is this Big Data? No!

• Is this Important? YES!

• **Multi-Tenancy**: The phone is more than just a DB.

• **Power**: 1-2 days of battery life under ideal circumstances.

• **It’s Everywhere**: Odds are that your phone is running some queries right now!
We need to better understand pocket-scale data
SQLite

- **Embedded**: SQLite is a library
- **Un-shared**: SQLite DBs are specific to one client “app”.
- **Lightweight**: Entire SQLite DB is backed to one file.
- **Universal**: SQLite client library is available by default in nearly all major OSes.
- **“Easy”**: Duck Typing, Relaxed SQL Syntax, One Big Lock (file)
How do developers and users use Pocket Scale Data?
PhoneLab
A Smartphone Platform Testbed

~200 UB students, faculty, and staff using instrumented LG Nexus 5 smartphones in exchange for discounted service.
PhoneLab

A Smartphone Platform Testbed

- **Preliminary Trial**: 11 phones for ~1 month (254 phone/days)
- Instrumented SQLite logs **all** statements (~45 mil statements)
  - ~33.5 million `SELECT` statements
  - ~9.4 million `INSERT` statements
  - ~1 million `UPDATE` statements
  - ~1.2 million `DELETE` statements
  - 179 distinct ‘apps’ issuing statements

https://phone-lab.org/experiment/request
• SELECT Complexity
• ORM Effects
• Function Usage
• Read/Write Ratios
• Query Periodicity
SELECT Complexity

- ORM Effects
- Function Usage
- Read/Write Ratios
- Query Periodicity
SELECT Complexity

Number of SELECT Queries vs. Number of Tables Accessed

Number of SELECT Queries vs. Maximum Nesting Depth
SELECT Complexity

30 million simple “SPA” queries

Number of SELECT Queries

Number of Tables Accessed

1000000
100000
10000
1000
100
10
1
0.1

1
2
3
4
5
6
7
8

1
2
3
4

1000000
100000
10000
1000
100
10
1

The ODIn Lab @
SELECT Complexity

30 million simple “SPA” queries

Infrequent, but extremely complex queries
SELECT Complexity
(by app)

% of SELECT Queries That Are Key-Value Queries

CDF

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SELECT Complexity (by app)

CDF

% of SELECT Queries That Are Key-Value Queries

24 / 179 apps using SQLite only as a K/V Store
SELECT Complexity

INSERT OR REPLACE INTO properties(property_key,property_value)
VALUES (?,?)

SELECT property_value
FROM properties
WHERE property_key=?;

(These are actual real queries from the trace)
SELECT Complexity

![CDF graph showing the cumulative distribution function of returned row count](image-url)
SELECT Complexity

80% of SELECTs return one row
SELECT Complexity

80% of SELECTs return one row

Small % of SELECTs return 100s of rows
SELECT Complexity
(by app)
SELECT Complexity
(by app)

CDF

Returned Row Count

0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

10^0 10^1 10^2 10^3 10^4

Google+, GMail, Facebook, Contacts
• SELECT Complexity

ORM Effects
• Function Usage
• Read/Write Ratios
• Query Periodicity
Object-Relational Mapper
pers = Persons.get(10)
name = pers.firstName()
```python
pers = Persons.get(10)
name = pers.firstName()
```

```sql
SELECT first_name
FROM Persons
WHERE id = 10;
```
Object-Relational Mapper

SQL DB used for persisting objects

```python
pers = Persons.get(10)
name = pers.firstName()
```

```sql
SELECT first_name
FROM Persons
WHERE id = 10;
```
pers = Persons.get(10)
org = pers.employer()
name = org.name()
pers = Persons.get(10)
org = pers.employer()
name = org.name()

SELECT employer_id
FROM Persons
WHERE id = 10;

SELECT name
FROM Organizations
WHERE id = ?;
pers = Persons.get(10)
org = pers.employer()
name = org.name()

SELECT employer_id
FROM Persons
WHERE id = 10;

SELECT name
FROM Organizations
WHERE id = ?;

ORMs are not always efficient
pers = Persons.get(10)
pers.setSalary(pers.salary() * 1.1)
pers = Persons.get(10)
pers.setSalary(
    pers.salary() * 1.1
)

SELECT salary
FROM Persons
WHERE id = 10;

UPDATE Persons
SET salary = ?
WHERE id = 10;
pers = Persons.get(10)
pers.setSalary(
    pers.salary() * 1.1
)

SELECT salary
FROM Persons
WHERE id = 10;

UPDATE Persons
SET salary = ?
WHERE id = 10;

We saw NO update value computations in SQL
pers = Persons.get(10)
pers.setSalary(pers.salary() * 1.1)

SELECT salary
FROM Persons
WHERE id = 10;

INSERT OR REPLACE INTO Persons(id, salary)
VALUES (?, 10);
Object-Relational Mapper

pers = Persons.get(10)
pers.setSalary(
    pers.salary() * 1.1
)

SELECT salary
FROM Persons
WHERE id = 10;

INSERT OR REPLACE INTO
    Persons(id, salary)
VALUES (?, 10);

**Insert or Replace used very frequently**
• SELECT Complexity
• ORM Effects
Function Usage
• Read/Write Ratios
• Query Periodicity
# Aggregates

<table>
<thead>
<tr>
<th>Function</th>
<th>Call Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_CONCAT</td>
<td>583,474</td>
</tr>
<tr>
<td>SUM</td>
<td>321,387</td>
</tr>
<tr>
<td>MAX</td>
<td>314,970</td>
</tr>
<tr>
<td>COUNT</td>
<td>173,031</td>
</tr>
<tr>
<td>MIN</td>
<td>19,566</td>
</tr>
<tr>
<td>AVG</td>
<td>15</td>
</tr>
</tbody>
</table>
## Aggregates

<table>
<thead>
<tr>
<th>Function</th>
<th>Call Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_CONCAT</td>
<td>583,474</td>
</tr>
<tr>
<td>SUM</td>
<td>321,387</td>
</tr>
<tr>
<td>MAX</td>
<td>314,970</td>
</tr>
<tr>
<td>COUNT</td>
<td>173,031</td>
</tr>
<tr>
<td>MIN</td>
<td>19,566</td>
</tr>
<tr>
<td>AVG</td>
<td>15</td>
</tr>
</tbody>
</table>

Aggregates most common function type
## Aggregates

<table>
<thead>
<tr>
<th>Function</th>
<th>Call Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP CONCAT</td>
<td>583,474</td>
</tr>
<tr>
<td>SUM</td>
<td>321,387</td>
</tr>
<tr>
<td>MAX</td>
<td>314,970</td>
</tr>
<tr>
<td>COUNT</td>
<td>173,031</td>
</tr>
<tr>
<td>MIN</td>
<td>19,566</td>
</tr>
<tr>
<td>AVG</td>
<td>15</td>
</tr>
</tbody>
</table>

**Concatenate all strings in a column: Non-algebraic**
Other Functions
Other Functions

- Mostly string manipulation (length, substr)
Other Functions

• Mostly string manipulation (length, substring)

• Some Android-Specific (phone_numbers_equal)
Other Functions

• Mostly string manipulation (length, substr)
• Some Android-Specific (phone_numbers_equal)
• NO UDFs at all
• SELECT Complexity
• ORM Effects
• Function Usage
• Read/Write Ratios
• Query Periodicity
Reads vs Writes

CDF

Read/Write Ratio (100% = All Reads)
~15% of apps write more frequently than they read.
Reads vs Writes

CDF

Read/Write Ratio (100% = All Reads)

~15% of apps write more frequently than they read

~15% of apps do not perform a single write!
Read-Only Workloads
Read-Only Workloads

• JuiceSSH, Key Chain

• Credential store, infrequent writes
Read-Only Workloads

- JuiceSSH, Key Chain
  - Credential store, infrequent writes
- Google Play Newsstand, Eventbrite, …
  - Frequent queries over changing data
  - Data bulk updated by copying entire SQLite DB
• SELECT Complexity
• ORM Effects
• Function Usage
• Read/Write Ratios

Query Periodicity
Query Arrival Frequency

CDF

Previous Query Arrival Time (μs)

[Graph showing the cumulative distribution function (CDF) of query arrival times.]
Query Arrival Frequency

15-20% of queries arrive ~10ms after last query
By Query Type

CDF

Next Query Arrival Time (µs)

CDF

Previous Query Arrival Time (µs)

- UPDATE
- SELECT
- INSERT
- DELETE
By Query Type

70% of inserts come less than 0.1 ms before another query.
By Query Type

70% of inserts come less than 0.1 ms before another query

Most sequences consist of INSERTs and SELECTs

CDF

Next Query Arrival Time (μs)

Previous Query Arrival Time (μs)
By App

CDF

Google Play services
Media Storage
Gmail
Google+
Facebook
Hangouts
Android System
Messenger
Calendar Storage
Contacts Storage

Previous Query Arrival Time (μs)
By App

Google Play Services and Media Storage are VERY bursty.
By App

Google Play Services and Media Storage are VERY bursty.

Same 10 ms periodicity evident across all apps.
A Call to Action!

• Mobile phones process ~2 queries/second
  • DB performance important for power, latency, …
• Embedded DBs used differently than Server DBs.
  • We need to understand these access patterns before we can optimize for them.

We need a TPC-MOBILE for pocket-scale data!