The Dangers and Complexities of SQLite Benchmarking

Dhathri Purohith, Jayashree Mohan and Vijay Chidambaram
Benchmarking SQLite is Non-trivial!

- Benchmarking complex systems in a repeatable fashion is error prone
- The main issues with benchmarking:
  - Inconsistency in the industrial benchmarking tools
  - Incorrect reporting of benchmarking results
• Benchmarking SQLite is hard
• Depends on several configuration parameters
• Current tools provide conflicting results (3X) for the same set of parameters
• Easy to show conflicting results by tuning parameters
• Right configuration can provide massive performance gains (28X)
Outline

● Overview of SQLite
● Motivation
● Existing tools to benchmark SQLite
● Parameters affecting performance of SQLite
● Conclusion
SQLite

- Lightweight, embedded, relational database popular in mobile systems
- Commonly used benchmark in many mobile applications to store their data
  - E.g. Twitter and Facebook
- Used as a benchmark for evaluating several systems
  - E.g. I/O scheduling frameworks (Yang et.al., SOSP ‘15), the Linux read-ahead mechanism (Olivier et.al., SIGBED ‘15)

Benchmarking SQLite is an important part of evaluating these systems.
SQLite architecture

User Space
Application

Cache

Disk

DB
SQLite architecture

User Space
Application

Cache

Disk

DB
SQLite architecture

User Space Application

Cache

Disk

Journal
SQLite architecture

User Space Application

Cache

Disk

fsync()

Journal

fsync()

fsync()
SQLite architecture

User Space
Application

Cache

Disk

DB

Journal
SQLite architecture
SQLite architecture

User Space Application

Cache

Disk

Journal

DB

Journal
SQLite architecture

User Space
Application

Cache

Disk

DB

Journal

Journal
Outline

● Overview of SQLite
● Motivation
● Existing tools to benchmark SQLite
● Parameters affecting performance of SQLite
● Conclusion
Motivation: A Case Study of SQLite

Benchmarking SQLite is tricky - Its performance varies greatly based on configuration parameters.

- **Default:** Delete journal mode, FULL synchronization mode on Ext4 in Android.
- **Workload:** 1 trial = 30K transactions (10K inserts, followed by updates and deletes of 10K)
Motivation: A Case Study of SQLite

Benchmarking SQLite is tricky - It’s performance varies greatly based on configuration parameters.

- **Default:** Delete journal mode, FULL synchronization mode on Ext4 in Android.
- **Workload:** 1 trial = 30K transactions (10K inserts, followed by updates and deletes of 10K)
- **Custom:** WAL journal mode with 1MB journal size and NORMAL synchronization mode on F2FS
Motivation: A Case Study of SQLite

Benchmarking SQLite is tricky - It’s performance varies greatly based on configuration parameters.

- **Default:** Delete journal mode, FULL synchronization mode on Ext4 in Android.
- **Workload:** 1 trial = 30K transactions (10 K inserts, followed by updates and deletes of 10K)
- **Custom:** WAL journal mode with 1MB journal size and NORMAL synchronization mode on F2FS

![Chart showing performance comparison between Default and Custom configurations]
Are we reporting it right?
Incomplete specification of benchmarking results

- 16 papers from the past couple of years, used SQLite to evaluate performance.

None of them reported all the parameters required to meaningfully compare results.
Outline

● Overview of SQLite
● Motivation
● Existing tools to benchmark SQLite
● Parameters affecting performance of SQLite
● Conclusion
Inconsistency in existing benchmarking tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Default TPS</th>
<th>Custom TPS</th>
<th>Papers that use</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobiBench</td>
<td>20</td>
<td>57</td>
<td>7</td>
</tr>
<tr>
<td>RL Bench</td>
<td>30</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>AndroBench</td>
<td>29</td>
<td>150</td>
<td>3</td>
</tr>
</tbody>
</table>

- Results between the tools differ by 50% in their default setting.
- Differ by 3X when a single parameter is changed.

Misleading and meaningless to compare, if parameters are not reported!
Outline

● Overview of SQLite
● Motivation
● Existing tools to benchmark SQLite
● Parameters affecting performance of SQLite
● Conclusion
Parameters affecting SQLite Performance

1. Filesystem
2. Journaling Mode
3. Pre-population of database
4. Synchronization Mode
5. Journal Size
Hardware Setup for experimentation

- Experiments performed on Samsung Galaxy Nexus S on 32GB internal storage.
- Controlled experimental setup: Vary one parameter, while keeping all others constant.
Workload

● 1 trial = 3000 transactions (1000 inserts, followed by 1000 updates and 1000 deletes)
● Database prepopulated with 100K rows.
● Results reported as throughput (transactions/sec)
● Default Configuration:
  ○ DELETE journal mode
  ○ FULL synchronization mode
  ○ Ext4 filesystem in ordered mode.
Outline

- Overview of SQLite
- Motivation
- Existing tools to benchmark SQLite
- Parameters affecting performance of SQLite
  - Filesystem
  - Journal Mode
  - Pre-population of the database
  - Synchronization mode
  - Journal Size
- Conclusion
1. Filesystem

- Application writes are transformed into block level operations by filesystem.
1. Filesystem

DELETE - Normal
1. Filesystem
1. Filesystem

![Graphs showing transactions per second for different filesystem operations: Insert, Update, Delete. The graphs compare Ext4 and F2FS under DELETE and WAL modes.](image)
1. Filesystem
1. Filesystem

[Graphs showing transactions per second for different filesystem operations (Insert, Update, Delete) under Normal and Full modes.]
1. Filesystem

- Depending on the parameters chosen, we can show either one performing better.
- F2fs paper evaluates only WAL mode: claims better performance than ext4.
Outline

● Overview of SQLite
● Motivation
● Existing tools to benchmark SQLite
● Parameters affecting performance of SQLite
  ○ Filesystem
  ○ Journal Mode
  ○ Pre-population of the database
  ○ Synchronization mode
  ○ Journal Size
● Conclusion
2. Journaling mode

- Defines the type of SQLite journal used.
  - **DELETE**: Default mode
    - Uses traditional rollback journaling mechanism: contents of the database is written on to the journal and the changes are written to the database file directly.
DELETE Journal mode revisited
2. Journaling mode

- Defines the type of SQLite journal used.
  - DELETE: Default mode
    - Uses traditional rollback journaling mechanism: contents of the database is written on to the journal and the changes are written to the database file directly.
  - WAL:
    - Write-ahead log, in which the changes to the database are written to the journal and is committed to the database when user explicitly triggers it.
WAL journal mode

User Space
Application

Cache

Disk
WAL journal mode

User Space Application

Tx : 1

Cache

Tx : 1

WAL

COMMIT

Disk
WAL journal mode

User Space Application

Cache

Disk

WAL

Tx : 1
WAL journal mode

User Space
Application

Cache

Disk

WAL

Tx : 2

Tx : 1

Tx : 2
WAL journal mode - checkpointing

User Space Application

Cache

Tx: 2

Disk

WAL

Tx: 1  Tx: 2

Checkpoint
2. Journaling mode

- OFF:
  - No Rollback journal
  - Likely corruption on crash
2. Journaling mode

- **X-axis**: Journaling mode
- **Y-axis**: Results reported in transactions/sec
2. Journaling mode

- **DELETE**: Max TPS of 30 achieved
2. Journaling mode

- WAL:
  Max TPS of 270 achieved
2. Journaling mode

- WAL 10X better than DELETE
- Journal deleted after each commit in DELETE mode.
- For 1000 SQLite inserts,
  - WAL : 1000 fsync()
  - DELETE : 5000 fsync()
Outline

● Overview of SQLite
● Motivation
● Existing tools to benchmark SQLite
● Parameters affecting performance of SQLite
  ○ Filesystem
  ○ Journal Mode
  ○ Pre-population of the database
  ○ Synchronization mode
  ○ Journal Size
● Conclusion
3. Pre-population of database

- Necessary to ensure realistic performance estimates.
3. Pre-population of database

- Necessary to ensure realistic performance estimates.
- Almost 2X performance difference
- Benchmarking tools don’t prepopulate. Unrealistic numbers.
Outline

● Overview of SQLite
● Motivation
● Existing tools to benchmark SQLite
● Parameters affecting performance of SQLite
  ○ Filesystem
  ○ Journal Mode
  ○ Pre-population of the database
  ○ Synchronization mode
  ○ Journal Size
● Conclusion
4. Synchronization Mode

- Controls the frequency of fsync() issued by SQLite library.
  - **FULL**: Writes to database(calls fsync()) on each commit.
FULL Synchronization in WAL

User Space Application

Cache

Disk

WAL

User Space Application

Tx : 1

Cache

Disk

WAL

Tx : 1

Checkpoint
4. Synchronization Mode

- Controls the frequency of fsync() issued by SQLite library.
  - **FULL**: Writes to database(calls fsync()) on each commit.
  - **NORMAL**: Writes to log on each commit.
NORMAL Synchronization in WAL

User Space
Application

Cache

Disk

Checkpoint

Tx : 1

Tx : 100

Tx : 1

Tx : 100

COMMIT

COMMIT

SYNC
4. Synchronization Mode

- Controls the frequency of fsync() issued by SQLite library.
  - **FULL**: Writes to database (calls fsync()) on each commit.
  - **NORMAL**: Writes to log on each commit.
  - **OFF**: Consistency mechanism left to the OS.
4. Synchronization Mode

- X-axis: Synchronization mode
- Y-axis: Results reported in transactions/sec
4. Synchronization Mode

- **FULL**: Max TPS: 30
4. Synchronization Mode

- **NORMAL**: Max TPS: 45
4. Synchronization Mode

- NORMAL : 1.5X better than FULL.
- To strike balance between durability and performance, use WAL+NORMAL.
Outline

- Overview of SQLite
- Motivation
- Existing tools to benchmark SQLite
- Parameters affecting performance of SQLite
  - Filesystem
  - Journal Mode
  - Pre-population of the database
  - Synchronization mode
  - Journal Size
- Conclusion
5. Journal Size

- In WAL mode, journal can grow unbounded
- Potentially affects read performance.
5. Journal Size

- In WAL mode, journal can grow unbounded
- Potentially affects read performance.
5. Journal Size

- In WAL mode, journal can grow unbounded
- Potentially affects read performance.

- Performance improves with increase in journal size
5. Journal Size

- In WAL mode, journal can grow unbounded
- Potentially affects read performance.

- Performance improves with increase in journal size
- When WAL is full - triggers checkpoint.
- Smaller WAL => more checkpointing
5. Journal Size

- In WAL mode, journal can grow unbounded
- Potentially affects read performance.

![Graph showing performance improvements with journal size]

- Performance improves with increase in journal size
- When WAL is full - triggers checkpoint.
- Smaller WAL => more checkpointing
- Saturates beyond a point
Outline

- Overview of SQLite
- Motivation
- Existing tools to benchmark SQLite
- Parameters affecting performance of SQLite
- Conclusion
Conclusion

● The Systems community has discussed in the past, how tricky benchmarking can be.
● But in practice, we have shown that industrial benchmarking tools are inconsistent, and academic reporting of results is incomplete.

● Draw attention to:
  ○ Developers and researchers must understand the impact of various parameters on SQLite performance.
  ○ To ensure repeatable and comparable results, reporting configuration parameters is vital.
THANK YOU..

Questions ?

Jayashree Mohan
jaya@cs.utexas.edu
Hardware Setup for experimentation

- Experiments performed on Samsung Galaxy Nexus S
- Controlled experimental setup: Vary one parameter, while keeping all others constant.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Dual Core 1.2GHz Cortex A9</td>
</tr>
<tr>
<td>Memory</td>
<td>32GB internal, 1GB RAM</td>
</tr>
<tr>
<td>Android</td>
<td>6.0.1(cyanogenmod 13)</td>
</tr>
<tr>
<td>Kernel</td>
<td>3.0.101 (F2FS enabled)</td>
</tr>
<tr>
<td>Battery</td>
<td>3.7V, 1850mAh</td>
</tr>
</tbody>
</table>