I/O Stack Optimization for Smartphones

Sooman Jeong¹, Kisung Lee², Seongjin Lee¹, Seoungbum Son², and Youjip Won¹

¹ Dept. of Electronics and Computer Engineering, Hanyang University
² Samsung Electronics
Outline

- Motivation
- Background
- Analysis of the Android I/O Stack
- Optimizations of the Android I/O Stack
  - Using the optimal journaling mode in SQLite
  - Alternative Filesystems
  - Eliminating unnecessary metadata flushes
  - External journaling
  - Using polling based I/O
- Evaluations
- Demo
Motivation

Smartphone is everywhere!

% of Chinese Internet Users Accessing the Web via Desktop PCs vs. via Mobile Phones, 6/07 – 12/12

[KPCB Internet Trends 2013]
Storage I/O is the performance bottleneck in Android.
Android Platform

Application Framework

- Window
- Package
- Telephony
- Contact

Libraries

- SQLite
- libc
- ...

Android Runtime

- Core lib
- Dalvik VM

Linux Kernel

- Display
- WiFi
- Filesystem
- Power MM
- Audio
- Block Device driver
- Key PAD

Storage
I/O stack of Android Platform

SQLite

EXT4

Block Device Driver (CFQ, Interrupt Driven IO)

Insert/update/delete

Read/write
I/O characteristics of Android Apps (GS3, ICS)

File Types

Block Types

I/O Modes

Locality

I/O Size

IRQs

Sooman Jeong et al.
USENIX ATC’13, SAN JOSE, CA, June 26~28, 2013
I/O characteristics of Android Apps (GS3, ICS)

SQLite > 90%

Metadata & Journal > 40%

Synchronous > 70%

File Types

Block Types

I/O Modes

Random > 80%

4KB I/O > 64%

IRQ for eMMC > 18%

Locality

I/O Size

IRQs
SQLite > 90% !!!

Metadata & Journal > 40% !

Synchronous Write > 70% !!!
Journaling in SQLite (Delete Mode)

1. Insert a database entry
2. Create journal.
3. Record the data to journal.
4. Put commit mark to journal.
5. Insert entry to DB
6. fsync()
7. fsync()
8. Delete journal.
write(fd, )
SQLite and EXT4

- write SQLite journal to storage.
- write SQLite DB to storage.
- write EXT4 journal (descriptor, metadata) to storage.
- write EXT4 journal (commit) to storage.
Summary

9 random writes to eMMC!!!
9 random writes to eMMC!!!
SQLite maintains DB journal. + EXT4 maintains filesystem journal. = EXT4 journals SQLite journal file.
SQLite maintains DB journal.

EXT4 journals SQLite journaling activity.
70% of the writes are purely for managerial purpose!

EXT4 journals SQLite journal file.
EXT4 journals SQLite journal file.

SQLite maintains DB journal.

EXT4 maintains filesystem journal.

EXT4 journaling activity.

70% of the writes are purely for managerial purpose!
Optimize Android I/O stack!
SQLite Journaling mode

DELETE

WAL

TRUNCATE

PERSIST

SQLite
Eliminating unnecessary metadata flushes

SQLite

fsync() vs. fdatasync()

EXT4
Alternative Filesystems

- XFS
- EXT4
- NILFS2
- F2FS
- BTRFS
Interrupt vs. Polling

interrupt vs. polling
External Journaling

EXT4

VS.
SQLite Journaling Modes
Delete (GS3, ICS)

2 \texttt{fsync()} and 9 \texttt{writes} for one insert()!
Truncate (GS3, ICS)

2 `fsync()` and 8 `writes`.
Persist (GS3, ICS)

The worst mode!

3 `fsync()` and 12 `writes`. The worst mode!
WAL Mode (GS3, ICS)

Only 1 `fsync()` and 3 writes. The best mode!
## Summary

<table>
<thead>
<tr>
<th>SQLite Journaling Mode</th>
<th>DELETE</th>
<th>TRUNCATE</th>
<th>PERSIST</th>
<th>WAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of fsync() calls</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Number of IOs</td>
<td>9</td>
<td>8</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>EXT4 Journal size (metadata)</td>
<td>24 KB</td>
<td>16 KB</td>
<td>8 KB</td>
<td>16 KB</td>
</tr>
<tr>
<td>Total IO Volume</td>
<td>72 KB</td>
<td>64 KB</td>
<td>72 KB</td>
<td>36 KB</td>
</tr>
</tbody>
</table>
Filesystems
"write() followed by fsync()" is the essence of the Android I/O.
4 KB `write()` followed by `fsync()`
F2FS (Flash Friendly Filesystem)
write() followed by fsync()

Summary

BTRFS

NILFS2

XFS

F2FS
fsync() vs. fdatasync()
Eliminating Unnecessary Metadata Flushes

fsync(fd0)   fsync(fd1)

Page cache

data  size  data  mtime  atime

Disk

data  size  data  mtime  atime

fdatasync(fd0)  fdatasync(fd1)
Eliminating Unnecessary Metadata Flushes

\[ \text{fsync}(fd0) \quad \text{fsync}(fd1) \]

Page cache

\[ \text{fdatasync}(fd0) \quad \text{fdatasync}(fd1) \]

Disk

\begin{align*}
\text{data} & \quad \text{size} \\
\text{data} & \quad \text{mtime} \quad \text{atime}
\end{align*}

\begin{align*}
\text{data} & \quad \text{size} \\
\text{data}
\end{align*}
External Journaling
4K random write() followed by fsync()
External journaling

Journal on separate partition
→ FTL can exploit the locality of I/O!

sequential
Interrupt driven I/O vs. Polling based I/O
Hardware trend

The number of CPU cores ↑

Multi-core on smartphones

I/O latency of eMMC ↓

Performance of mobile flash storage

Sooman Jeong et al. USENIX ATC'13, SAN JOSE, CA, June 26~28, 2013
Interrupt driven I/O

- mmcqd
- Send I/O request
- Sleep()
- IRQ handler
- Complete I/O request

Polling based I/O

- mmcqd
- Send I/O request
- Busy wait
- Complete I/O request

Polling can reduce context switching overhead!
Experiment
## Galaxy S3 (ICS 4.0.4, Linux 3.0.15)

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Exynos 4412 1.4 GHz Quad-core</td>
</tr>
<tr>
<td>RAM</td>
<td>2 GB</td>
</tr>
<tr>
<td>Internal Storage</td>
<td>32 GB eMMC</td>
</tr>
<tr>
<td>External Storage</td>
<td>16 GB Transcend u-SD Card</td>
</tr>
</tbody>
</table>
SQLite performance: journaling modes

SQLite Insert

- TRUNCATE (default) → WAL: 116% up
- TRUNCATE, EXT4 (default) → WAL, F2FS: 281% up
SQLite performance: journaling modes

SQLite Update

- TRUNCATE (default) → WAL: 232% up
- TRUNCATE, EXT4 (default) → WAL, F2FS: 348% up
**fsync() vs. fdatasync()**

SQLite Insert

- `fsync()` → `fdatasync()` : 17% up
- `fsync()` → `fdatasync()` and F2FS : 126% up
**fsync() vs. fdatasync()**

**SQLite Update**

- $\text{fsync()} \rightarrow \text{fdatasync()} : 53\% \text{ up}$
- $\text{fsync()} \rightarrow \text{fdatasync()}$ and F2FS : 250% up
External journaling

SQLite Insert

<table>
<thead>
<tr>
<th>File System</th>
<th>Insert / sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base EXT4</td>
<td>30%</td>
</tr>
<tr>
<td>Ext-Journal</td>
<td>37%</td>
</tr>
<tr>
<td>Base XFS</td>
<td>20%</td>
</tr>
<tr>
<td>Ext-Journal</td>
<td>30%</td>
</tr>
</tbody>
</table>

SQLite Update

<table>
<thead>
<tr>
<th>File System</th>
<th>Update / sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base EXT4</td>
<td>39%</td>
</tr>
<tr>
<td>Ext-Journal</td>
<td>20%</td>
</tr>
<tr>
<td>Base XFS</td>
<td>20%</td>
</tr>
<tr>
<td>Ext-Journal</td>
<td>39%</td>
</tr>
</tbody>
</table>
### 4 KB random write+fsync()

<table>
<thead>
<tr>
<th># of thread</th>
<th>Scenario</th>
<th>Idle base</th>
<th>Idle poll</th>
<th>HD Record base</th>
<th>HD Record poll</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>KIOPS</td>
<td>1002</td>
<td>981</td>
<td>667</td>
<td>756</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>7.5</td>
<td>10.9</td>
<td>26.4</td>
<td>30.2</td>
</tr>
<tr>
<td>10</td>
<td>KIOPS</td>
<td>2609</td>
<td>2705</td>
<td>2136</td>
<td>2351</td>
</tr>
<tr>
<td></td>
<td>CPU (%)</td>
<td>11.1</td>
<td>12.9</td>
<td>30.1</td>
<td>33.1</td>
</tr>
</tbody>
</table>

- Marginal gain (1~2%) when CPU is IDLE.
- 13% gain when we record HD video in background.
Real Workload

Replay Twitter and Facebook by Mobigen

Twitter

Facebook

-71%

-58%
Combining All the Improvements

- B: Base
- P: Polling
- E: External Journaling
- F: `fdatasync()`
- W: WAL mode

SQLite Insert

<table>
<thead>
<tr>
<th>B</th>
<th>P</th>
<th>E</th>
<th>F</th>
<th>FEP</th>
<th>FEPW</th>
<th>FEP</th>
<th>FEPW</th>
<th>FP</th>
<th>FPW</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>160</td>
<td>140</td>
<td>120</td>
<td>100</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

300% and 134% improvements with fdatasync(), Ext. J, Polling, WAL.
Finally,

Polling

WAL

300% up !!
39 → 157 ins/sec

F2FS

Ext. J

fdatasync

Block Device Driver

SQLite
Finally,

Polling

WAL

F2FS

Ext. J

atasync
• Android IO stack is collection of unorchestrated layers.
• Journaling of Journal (JOJ) lies at the core of the problem.
• We optimize Android I/O stack with WAL mode in SQLite, F2FS, fdatasync(), External journaling, polling based I/O.

What we achieved is...

• With legacy EXT4, SQLite performance improves by 134%.
• With F2FS, SQLite performance improves by 300%

solely via software modification on existing smartphone!
Thank you…

77smart@hanyang.ac.kr