Boosting Quasi-Asynchronous I/O for Better Responsiveness in Mobile Devices

Daeho Jeong*,+ Youngjae Lee*, Jin-Soo Kim*+

*Samsung Co., Suwon, Korea
*Sungkyunkwan University
Jeong, Daeho

No contacts
• CPU?
• Memory?
• I/O?
Linux Approaches in I/O Scheduling

• Block layer
  – Classify I/O into \{SYNC || ASYNC\}

• CFQ I/O scheduler
  – SYNC queues have larger time slices than ASYNC
  – A SYNC queue per a process
    (vs. An ASYNC queue is shared)
  – Set a limit for ASYNC requests that can be dispatched in a single time slice
  – A new SYNC req. preempts other ASYNC req.
Page Cache

Page
Page
Page
Page
Page
Page
Page

Apps

I/O scheduler
By *kworker*,

Especially, when the number of dirty pages exceeds the **background dirty ratio**
What if a process waits for the completion of *asynchronous I/O*?
Quasi-Asynchronous I/O (QASIO)

- Quasi-Asynchronous I/O
  - Issued *asynchronously*, but should be treated as *synchronous* I/O
  - Detected *at run time*
  - Causes a problem such as *priority inversion problem*
- Causes unexpected delay
  - Similar to priority inversion problem

**Processing Order**

```
I/O scheduler
Async I/O  Async I/O  Async I/O  Async I/O  Async I/O  Async I/O  Async I/O  Async I/O  QASIO
```

**Wait on Write Back!!**

Task A

---

Wait on Write Back!!

Task A

---

Wait on Write Back!!

Task A

---

Wait on Write Back!!

Task A
Outline

• Definition of QASIO
• Dependencies on QASIO
• Impact by QASIO
• Implementation Overview
• Evaluation
• Conclusions
Dependencies on QASIO

Process A

Indirect dependency

Direct dependency

I/O scheduler

Async I/O

QASIO

Async I/O

Async I/O
Types of Dependencies on QASIO

• Direct Dependencies
  – $D_{\text{meta}}$: When modifying a metadata page
  – $D_{\text{data}}$: When modifying a data page
  – $D_{\text{sync}}$: When guaranteeing data to be written back
  – $D_{\text{discard}}$: When completing discard commands

• Indirect Dependencies
  – $I_{\text{jhandle}}$: When unable to obtain a journal handle
  – $I_{\text{jcommit}}$: When unable to complete $\text{fsync}()$
Types of Dependencies on QASIO

Direct Dependencies
- $D_{\text{meta}}$: When modifying a metadata page
- $D_{\text{data}}$: When modifying a data page
- $D_{\text{sync}}$: When guaranteeing data to be written back
- $D_{\text{discard}}$: When completing discard commands

Indirect Dependencies
- $I_{\text{jhandle}}$: When unable to obtain a journal handle
- $I_{\text{jcommit}}$: When unable to complete $\text{fsync}()$
Types of Dependencies on QASIO

• Direct Dependencies
  - $D_{\text{meta}}$ : When modifying a metadata page
  - $D_{\text{data}}$ : When modifying a data page
  - $D_{\text{sync}}$ : When guaranteeing data to be written back
  - $D_{\text{discard}}$ : When completing discard commands

• Indirect Dependencies
  - $I_{\text{jhandle}}$ : When unable to obtain a journal handle
  - $I_{\text{jcommit}}$ : When unable to complete $\text{fsync()}$
Case Study of D_{meta}

rename(), write(), unlink(), chmod(), chown(), fsync()
Case Study of D_{meta}

1. async. bulky write
2. flush out

Page Cache

inode entries, directory entries, block bitmaps, inode bitmaps, group descriptors, super block

metadata buffer

3. COMMIT & make “dirty”

JBD2 Journaling

Apps

Page Cache

kworker

I/O Scheduler
Case Study of D_{meta}

1. async. bulky write
2. flush out
3. COMMIT & make “dirty”
4. flush out & lock buffer

Page Cache

Apps

Page Cache

metadata buffer

JBD2 Journaling

I/O Scheduler

Async I/O
Case Study of D_{meta}

1. async. bulky write
2. flush out
3. COMMIT & make “dirty”
4. flush out & lock buffer

rename(), write(), unlink(), chmod(), chown(), fsync()
Case Study of D\textsubscript{meta}

1. async. bulky write
2. flush out
3. COMMIT & make “dirty”
4. flush out & lock buffer
5. invoke file operations (lock\_buffer())

rename(), write(), unlink(), chmod(), chown(), fsync()
Case Study of $D_{sync}$

Normal Case

UI Task

write()

Page Cache

page

I/O Scheduler
Case Study of D\textsubscript{\text{sync}}

**Normal Case**

- UI Task
  - \texttt{fsync()} (flush & wait)
  - Sync. Req.
  - Page Cache
  - I/O Scheduler
- page
Case Study of D_{sync}

1. async. bulky write
Page Cache

2. flush out
kworker

I/O Scheduler

UI Task
Case Study of $D_{\text{sync}}$

1. async. bulky write

Apps

Page Cache

2. flush out

UI Task

3. async. write

write()

Page Cache

$kworker$

kworker

I/O Scheduler
Case Study of D_{\text{sync}}

1. async. bulky write
2. flush out
3. async. write
4. flush out

Page Cache

Apps

write()

UI Task

kworker

I/O Scheduler

Async I/O
Case Study of $D_{\text{sync}}$

1. async. bulky write
2. flush out
3. async. write
4. flush out
5. invoke fsync()
   (wait_on_page_writeback())
1. 1K partial write

Case Study of I\textit{handle}

**Apps**

<table>
<thead>
<tr>
<th>Page Cache</th>
<th>Task A</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>write()</td>
</tr>
</tbody>
</table>

**Page Cache**

**Page**

**I/O Scheduler**

**JBD2 Journaling**

Running Transaction

**handle**

**kworker**
Case Study of jhandle

Apps

Page Cache

1. 1K partial write

Task A

JBD2 Journaling
Running Transaction
handle

Page

write()

2. flush out

kworker

I/O Scheduler

Async I/O
Case Study of I\_jhandle

1. 1K partial write
2. flush out
3. 1K partial write

JBD2 Journaling
Running Transaction
handle

Page Cache
page
page

Apps

Task A

write()
Case Study of I$_j$handle

Task A

1. 1K partial write

2. flush out

Page Cache

Page

Page

kworker

I/O Scheduler

QASIO

JBD2 Journaling

Running Transaction

handle

write()

4. wait_on_page_writeback()
Case Study of jhandle

1. 1K partial write
2. flush out
3. 1K partial write
4. wait_on_page_writeback()
5. COMMIT & locked

Apps → Page Cache → Task A → Page → kworker → I/O Scheduler → QASIO

JBD2 Journaling
Running Transaction

handle

1K partial write

locked

D_data
Case Study of Ijhandle

1. 1K partial write
2. flush out
3. 1K partial write
4. wait_on_page_writeback
5. COMMIT & locked

JBD2 Journaling
Running Transaction

Other Tasks

Page Cache

Apps

Task A

Page

kworker

I/O Scheduler

QASIO
How Severe is the delay by QASIO?

- The delay by QASIOs depends on
  - The number of outstanding requests
  - The maximum number of requests
  - I/O performance of underlying storage device

- A file system call can be blocked for
  - Over 1 second on an MLC eMMC (S.W.: 57.4 MB/s)
  - Over 4 seconds on a TLC eMMC (S.W.: 26.0 MB/s)

*S.W. => Sequential Write Bandwidth*
Degradations by QASIOs in Real-Life Scenarios

- App start time is slowed down by 2.4x in the worst case.
- Shot count is decreased by 19%.
- App install time is increased by 35%.
How to Boost a QASIO

- Just focus on Direct Dependencies

- Two requirements
  - Req.(1): When a task is waiting for an asynchronous I/O’s completion, the kernel gives information about QASIO to the I/O scheduler

  - Req.(2): The I/O scheduler should prioritize them among asynchronous I/Os based on the hint
How to Boost a QASIO

• Just focus on Direct Dependencies

• Two requirements
  – **Req.(1):** When a task is waiting for an asynchronous I/O’s completion, the kernel gives information about QASIO to the I/O scheduler
    => VFS, MM, FS, Block Layer
  – **Req.(2):** The I/O scheduler should prioritize them among asynchronous I/Os based on the hint
    => Each I/O Scheduler
Implementation Overview

VFS Layer
lock_buffer();

Page Cache
wait_on_page_writeback();

EXT4 Filesystem
ext4_free_data_callback();

Detect $D_{data}$ & $D_{sync}$
Detect $D_{meta}$
Detect $D_{discard}$

Block Layer

elv_boost();

CFQ I/O Scheduler

CFQ Async. I/O Request Tree

next req.

adjacent distance

$\triangle$ : red-black tree
$\square$ : I/O request
Implementation Overview

VFS Layer
- lock_buffer();

Page Cache
- wait_on_page_writeback();

EXT4 Filesystem
- ext4_free_data_callback();

Detect $D_{data}$ & $D_{sync}$
Detect $D_{meta}$
Detect $D_{discard}$

Block Layer

CFQ I/O Scheduler

QASIO list
- 5 - 6 - 7

I/O Boosting

CFQ Async. I/O Request Tree

next req.

adjacent distance

$: red-black tree
$: I/O request
Evaluations

• **Samsung Galaxy S5**
  – Exynos 5422 (quad Cortex-A15 & quad Cortex-A7)
  – 2GB DRAM
  – 16GB eMMC storage (S.W.: 54.5MB/s, R.W.: 10.4MB/s)
  – Android platform version 4.4.2 (KitKat)
  – Linux kernel version 3.10.9

• Evaluation methodology
  – Microbenchmarks, Real-life scenarios, Android I/O benchmarks
Microbenchmarks

• Five microbenchmarks (M1 ~ M5)

• M1
  – Iterates the creation of a 4KB file, 500 times
  – performs fsync() to each created file
  – creat() -> write(4KB) -> fsync() -> close()
  – Mimics the I/O pattern of DB
Microbenchmarks

• The normalized total elapsed time

The total elapsed time is reduced by up to 83.1% ↓
**Microbenchmarks**

- The latency of key file system calls

<table>
<thead>
<tr>
<th>Opt</th>
<th>M1</th>
<th></th>
<th></th>
<th>M2</th>
<th></th>
<th></th>
<th>M3</th>
<th></th>
<th></th>
<th>M4</th>
<th></th>
<th></th>
<th>M5</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>creat()</td>
<td>fsync()</td>
<td></td>
<td>creat()</td>
<td>fsync()</td>
<td></td>
<td>write()</td>
<td></td>
<td></td>
<td>truncate()</td>
<td></td>
<td></td>
<td>fsync()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
<td>Max</td>
<td>Avg</td>
</tr>
<tr>
<td><strong>NONE</strong></td>
<td>119.57</td>
<td>1435.54</td>
<td>98.24</td>
<td>1119.62</td>
<td>109.01</td>
<td>1397.64</td>
<td>172.05</td>
<td>1417.82</td>
<td>0.19</td>
<td>1813.54</td>
<td>62.60</td>
<td>1632.15</td>
<td>13.27</td>
<td>13.87</td>
<td></td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>1.02</td>
<td>39.15</td>
<td>35.64</td>
<td>144.69</td>
<td>3.90</td>
<td>22.52</td>
<td>69.24</td>
<td>298.83</td>
<td>0.10</td>
<td>177.40</td>
<td>12.85</td>
<td>334.57</td>
<td>6.85</td>
<td>7.11</td>
<td></td>
</tr>
</tbody>
</table>

97.3%↓  98.4%↓  90.2%↓
Real-life scenarios

- Scenario A: Launching the “Contacts” App

The worst case launch time is reduced by up to 44.8%↓

The total wait time by $D_{meta}$ and $D_{discard}$ is reduced by 96.1%↓ and 87.4%↓
Real-life scenarios

• Scenario B: Burst Mode in the “Camera” App

The shot count is improved by up to 14.4%↑
The total wait time by $D_{data}$ is reduced by 98.4%↓
Real-life scenarios

• Scenario C: Installing the “Angry Birds” App

The app install time is improved by up to 11.5% ↓
Android I/O benchmarks

(a) Antutu benchmark

(b) RLBench benchmark

12.0%↑  10.2%↑

17.1%↓
Conclusions

• A new type of I/O, QASIO
  – Seemingly asynchronous, but has the synchronous property
  – Types of dependency on QASIO

• Novel scheme to detect and boost QASIO
  – The worst case latency of a file system call, 98.4%↓
  – The worst case “Contacts” app start time, 44.8%↓

• Future work
  – Analyze the impact of QASIO on other types of systems
  – Devise another solutions optimized for each type of QASIO
Thank You

Daeho Jeong (daeho43@gmail.com)
Youngjae Lee (yjlee@csl.skku.edu)
Jin-Soo Kim (jinsookim@skku.edu)