Optimizing Storage in Your App

Better performance and efficiency

Kai Kaahaaina, CoreOS
Alejandro Lucena, Developer Tools
Optimizing Storage in Your App
Better performance and efficiency
Optimizing Storage in Your App
Better performance and efficiency

Battery life
Optimizing Storage in Your App
Better performance and efficiency

Battery life

Performance
Optimizing Storage in Your App
Better performance and efficiency

Battery life
Performance
Reduce footprint
Optimizing Storage in Your App
Better performance and efficiency

Battery life
Performance
Reduce footprint
Device health
Efficient image assets
File system metadata
Syncing to disk
Serialized data files
Core Data
SQLite
Efficient Image Assets
Image Assets
Growing in size

Increasing screen resolutions
• iPhone 6s — 1334 x 750
• iPhone Xs — 2436 x 1125

Higher resolution cameras

Harder to manage
Image Assets
Growing in size
Image Assets
Growing in size
JPEG Size: 24.6MB
HEIC
A more efficient and capable alternative to JPEG
HEIC
A more efficient and capable alternative to JPEG

~50% smaller files than JPEG at comparable quality
HEIC
A more efficient and capable alternative to JPEG

~50% smaller files than JPEG at comparable quality
Smaller on disk footprint
HEIC
A more efficient and capable alternative to JPEG

~50% smaller files than JPEG at comparable quality

Smaller on disk footprint

Faster network downloads and uploads
HEIC
A more efficient and capable alternative to JPEG

~50% smaller files than JPEG at comparable quality
Smaller on disk footprint
Faster network downloads and uploads
Faster to load and save to disk
HEIC
A more efficient and capable alternative to JPEG

Store auxiliary images (depth, disparity, and so on)
HEIC
A more efficient and capable alternative to JPEG

Store auxiliary images (depth, disparity, and so on)

Supports alpha
HEIC
A more efficient and capable alternative to JPEG

Store auxiliary images (depth, disparity, and so on)

Supports alpha

Lossless compression
HEIC
A more efficient and capable alternative to JPEG

Store auxiliary images (depth, disparity, and so on)

Supports alpha

Lossless compression

Multiple images in a single container
HEIC
A more efficient and capable alternative to JPEG

Available starting with
• iOS 11
• macOS High Sierra
Image Assets
Growing in size
JPEG Size: 24.6MB
Image Assets
Growing in size
JPEG Size: 24.6MB
HEIC Size: 17.9MB
Asset Catalog
Simple app resource management
Asset Catalog
Simple app resource management

App icon
Asset Catalog
Simple app resource management

App icon

Device and scale variants
Asset Catalog
Simple app resource management

App icon
Device and scale variants
On demand resources
Asset Catalog
Storage and performance benefits

Storage efficiency
Asset Catalog
Storage and performance benefits

Storage efficiency
• On-disk footprint
Asset Catalog

Storage and performance benefits

Storage efficiency

• On-disk footprint
• App slicing (iOS)
Asset Catalog

Storage and performance benefits

Storage efficiency
• On-disk footprint
• App slicing (iOS)

Performance
Asset Catalog
Storage and performance benefits

Storage efficiency
• On-disk foot print
• App slicing (iOS)

Performance
• Image loading
Asset Catalog
Storage and performance benefits

Storage efficiency
• On-disk foot print
• App slicing (iOS)

Performance
• Image loading
• App launch
Asset Catalog
Storage and performance benefits

Storage efficiency
• On-disk footprint
• App slicing (iOS)

Performance
• Image loading
• App launch (up to 10% faster!)
Asset Catalog
Easy GPU based compression

Lossless by default
Asset Catalog
Easy GPU based compression

Lossless by default

Lossy image compression available
Asset Catalog

Easy GPU based compression

Lossless by default

Lossy image compression available

• Hardware accelerated decompression
Asset Catalog
Easy GPU based compression

Lossless by default

Lossy image compression available
  • Hardware accelerated decompression
  • Lower memory footprint
Image Assets
Growing in size
JPEG Size: 24.6MB
HEIC Size: 17.9MB
Image Assets
Growing in size
JPEG Size: 24.6MB
HEIC Size: 17.9MB
AssetCatalog: 14.9MB
Use HEIC and Asset Catalogs to reduce footprint
File System Metadata
File System Metadata
Modifying a plist

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>last_app_launch</key>
    <date>2019-06-07T07:26:49Z</date>
</dict>
</plist>
File System Metadata
Modifying a plist

On launch our demo app reads and updates this plist
File System Metadata
Modifying a plist

On launch our demo app reads and updates this plist
• One read
File System Metadata
Modifying a plist

On launch our demo app reads and updates this plist
• One read
• Three writes
File System Metadata
Modifying a plist

On launch our demo app reads and updates this plist
• One read
• Three writes
• fsync
File System Metadata

Why three writes instead of one?

File system metadata writes happen as a result of

• Creating a file
• Deleting a file
• Renaming a file
• Updating a file
• And other file system operations
File System Metadata

What is file system metadata?

Name
File System Metadata
What is file system metadata?

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**File System Metadata**

What is file system metadata?

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
File System Metadata

What is file system metadata?

Name
Size
Location
Dates
File System Metadata
What is file system metadata?

Name
Size
Location
Dates
And more
File System Metadata
Writing a 240 byte NSDictionary to a file
File System Metadata
Writing a 240 byte NSDictionary to a file

Update file system node
File System Metadata
Writing a 240 byte NSDictionary to a file
File System Metadata
Writing a 240 byte NSDictionary to a file
File System Metadata
Writing a 240 byte NSDictionary to a file

File System Tree

Original

Transaction ID: 1
Node: 123

New

New

Transaction ID: 2
Node: 123
File System Metadata
Writing a 240 byte NSDictionary to a file

Update file system node
File System Metadata
Writing a 240 byte NSDictionary to a file

Update file system node
Update object map
File System Metadata
Writing a 240 byte NSDictionary to a file

Update file system node
Update object map
Total Metadata: 8K
File System Metadata

Writing a 240 byte NSDictionary to a file

Update file system node (4K)

Update object map

Total Metadata: 8K
File System Metadata
Writing a 240 byte NSDictionary to a file

Update file system node (4K)
Update object map (4K)
Total Metadata: 8K
File System Metadata
Writing a 240 byte NSDictionary to a file

Update file system node (4K)
Update object map (4K)
Total Metadata: 8K
File itself
File System Metadata
Writing a 240 byte NSDictionary to a file

Update file system node (4K)
Update object map (4K)
Total Metadata: 8K
File itself (4K)
File System Metadata

Writing a 240 byte NSDictionary to a file

Update file system node (4K)
Update object map (4K)
Total Metadata: 8K
File itself (4K)
Total data: 12K
File System Metadata

Book keeping workload for common operations

Create a file: 8K
File System Metadata
Book keeping workload for common operations

Create a file: 8K
Delete a file: 8K
File System Metadata
Book keeping workload for common operations

Create a file: 8K
Delete a file: 8K
Rename a file: 16K
File System Metadata
Book keeping workload for common operations

Create a file: 8K
Delete a file: 8K
Rename a file: 16K
Modifying a file: 8K
File system metadata updates are not free.
File System Metadata
Efficient non-persistent files

Create a file
File System Metadata
Efficient non-persistent files

Create a file
Keep it open and unlinked
File System Metadata
Efficient non-persistent files

Create a file

Keep it open and unlinked

Do not call `fsync()`
File System Metadata

More information on APFS

Syncing to Disk
flushes cached data to disk

fsync()

App

OS Cache

Disk Cache

Permanent Storage
fsync()

Flushes cached data to disk
- Moves data from the OS cache to the disk cache
Flushes cached data to disk

- Moves data from the OS cache to the disk cache
- Data may not be written to permanent storage immediately
Flushes cached data to disk
• Moves data from the OS cache to the disk cache
• Data may not be written to permanent storage immediately
• Does not guarantee write ordering
flushes cached data to disk

- Moves data from the OS cache to the disk cache
- Data may not be written to permanent storage immediately
- Does not guarantee write ordering
- Expensive

fsync()
Flushes cached data to disk

- Moves data from the OS cache to the disk cache
- Data may not be written to permanent storage immediately
- Does not guarantee write ordering
- Expensive
- Done periodically by the OS
Drive flush its cache to disk

F_FULLFSYNC

App

OS Cache

Disk Cache

Permanent Storage
Drive flush its cache to disk
• Causes all data in disk cache to be flushed
Drive flush its cache to disk
• Causes all data in disk cache to be flushed
• Expensive
Drive flush its cache to disk
• Causes all data in disk cache to be flushed
• Expensive
• Done periodically by OS
F_BARRIERFSYNC

Enforces I/O ordering
F_BARRIERFSYNC

Enforces I/O ordering

- `fsync()` with a barrier
F_BARRIERFSYNC

Enforces I/O ordering
• \texttt{fsync()} with a barrier

Much less expensive than \texttt{F_FULLFSYNC}
Enforcing I/O Ordering

- **F_BARRIERFSYNC**: Ensures I/O ordering for the specified data

- **F_FULLFSYNC**: Indiscriminately commits all disk cache data to permanent storage
Serialized Data Files
Plists, XML, and JSON

The good

• Convenient
Plists, XML, and JSON

The good
• Convenient
• Excellent for infrequently written data
Plists, XML, and JSON

The good

• Convenient
• Excellent for infrequently written data
• Easy to parse
Plists, XML, and JSON

The trade off
Plists, XML, and JSON

The trade off

• Entire file must be rewritten for every change
Plists, XML, and JSON

The trade off

• Entire file must be rewritten for every change
• Scales poorly
Plists, XML, and JSON

The trade off

• Entire file must be rewritten for every change
• Scales poorly
• Easy to misuse
Plists, XML, and JSON

The trade off
• Entire file must be rewritten for every change
• Scales poorly
• Easy to misuse
• Metadata intensive
Plists, XML, and JSON

The trade off

• Entire file must be rewritten for every change
• Scales poorly
• Easy to misuse
• Metadata intensive

Not meant to be a database
**NSDictionary**

Create, read, and modify example

<table>
<thead>
<tr>
<th>Start-</th>
<th>Duration</th>
<th>Process</th>
<th>Thread</th>
<th>Narrative</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:02:814.992</td>
<td>166.06 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed open on path plist_example.plist</td>
</tr>
<tr>
<td>00:02:815.177</td>
<td>153.14 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed write on fd 3 with a requested size of 484.46 KiB. Amount of bytes actually processed: 484.46 KiB</td>
</tr>
<tr>
<td>00:02:815.332</td>
<td>474.19 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed fsync on fd 3 and path (plist_example.plist)</td>
</tr>
<tr>
<td>00:02:815.810</td>
<td>15.83 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed close on fd 3 (plist_example.plist)</td>
</tr>
<tr>
<td>00:02:816.009</td>
<td>25.57 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed open on path plist_example.plist</td>
</tr>
<tr>
<td>00:02:816.537</td>
<td>2.00 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed fstat64 on fd 3 and path (plist_example.plist)</td>
</tr>
<tr>
<td>00:02:816.561</td>
<td>296.53 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed read on fd 3 with a requested size of 484.46 KiB. Amount of bytes actually processed: 484.46 KiB</td>
</tr>
<tr>
<td>00:02:816.859</td>
<td>5.23 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed close on fd 3 (plist_example.plist)</td>
</tr>
<tr>
<td>00:02:817.305</td>
<td>164.45 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed open on path plist_example.plist</td>
</tr>
<tr>
<td>00:02:817.473</td>
<td>135.80 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed write on fd 3 with a requested size of 484.46 KiB. Amount of bytes actually processed: 484.46 KiB</td>
</tr>
<tr>
<td>00:02:817.811</td>
<td>284.92 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed fsync on fd 3 and path (plist_example.plist)</td>
</tr>
<tr>
<td>00:02:817.897</td>
<td>9.04 µs</td>
<td>plist_example (31544)</td>
<td>plist_example (pid: 31544, tid: 0x606...)</td>
<td>plist_example (31544) performed close on fd 3 (plist_example.plist)</td>
</tr>
</tbody>
</table>
### NSDictionary

**Modify example**

<table>
<thead>
<tr>
<th>Start</th>
<th>Total Latency</th>
<th>Process</th>
<th>Thread</th>
<th>Operation</th>
<th>Path</th>
<th>Extra Flags</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:02.815.378</td>
<td>409.48 µs</td>
<td>plist_example (31544)</td>
<td>Main Thread 0x60662ef</td>
<td>Data Write</td>
<td>plist_e...ple.plist</td>
<td>ASYNC</td>
<td>488.00 Kib</td>
</tr>
<tr>
<td>00:02.877.647</td>
<td>239.84 µs</td>
<td>plist_example (31544)</td>
<td>Main Thread 0x60662ef</td>
<td>Data Write</td>
<td>plist_e...ple.plist</td>
<td>ASYNC</td>
<td>488.00 Kib</td>
</tr>
</tbody>
</table>

Backtrace:
- fsync
- _NSWriteDataToWriteDataWithExtendedAttributes
- `[NSDictionary dictionaryWithContentsOfFile:atomically:]`
- `writeDictToFile(dict:fileName:)`
- `main`
- `start`
NSDictionary
Modify example
Core Data
Core Data

Core Data management
• Built on SQLite
Core Data

Core Data management
• Built on SQLite
• Manages object graphs and relationships
Core Data

Core Data management
• Built on SQLite
• Manages object graphs and relationships
• Change tracking and notifications
Core Data

Core Data management
• Built on SQLite
• Manages object graphs and relationships
• Change tracking and notifications
• Automatic version tracking and multi-writer conflict resolution
  - Automatic connection pooling
Core Data

Core Data management
• Built on SQLite
• Manages object graphs and relationships
• Change tracking and notifications
• Automatic version tracking and multi-writer conflict resolution
  - Automatic connection pooling
• CloudKit integration
Core Data

Core Data management
• Built on SQLite
• Manages object graphs and relationships
• Change tracking and notifications
• Automatic version tracking and multi-writer conflict resolution
  - Automatic connection pooling
• CloudKit integration (new with iOS 13)
Core Data

Core Data management
• Built on SQLite
• Manages object graphs and relationships
• Change tracking and notifications
• Automatic version tracking and multi-writer conflict resolution
  - Automatic connection pooling
• CloudKit integration (new with iOS 13)
• Live queries
Core Data

Core Data management
• Automatic memory management
Core Data

Core Data management

• Automatic memory management
• Statement aggregation in transactions
Core Data

Core Data management
• Automatic memory management
• Statement aggregation in transactions
• Schema migrations
Core Data

Core Data management
• Automatic memory management
• Statement aggregation in transactions
• Schema migrations
• Denormalization
Core Data

Core Data management
- Automatic memory management
- Statement aggregation in transactions
- Schema migrations
- Denormalization (new with iOS 13)
Core Data

Core Data management

• Automatic memory management
• Statement aggregation in transactions
• Schema migrations
• Denormalization (new with iOS 13)
• And so much more!
50–70%

Reduction in code to support the model layer
SQLite
Connections
Journaling
Transactions
File size and privacy
Partial indexes
Connections
Opening and Closing Connections

Can cause expensive operations!

• Consistency checking
• Journal recovery
• Journal checkpointing
Opening and Closing Connections

Can cause expensive operations!

- Consistency checking
- Journal recovery
- Journal checkpointing

Recommended usage model

- Keep connections open as long as possible
- Close only when necessary
- Pool connections on multi-threaded processes
Journaling
Delete Mode Journaling

Default SQLite journaling mode
How Does Delete Mode Journaling Work?
How Does Delete Mode Journaling Work?

Database

Journal
How Does Delete Mode Journaling Work?
How Does Delete Mode Journaling Work?
WAL Mode Journaling Reduces Writes

PRAGMA journal_mode=WAL;

Write Ahead Logging
Combines multiple writes to the same page
Uses fewer barriers
Supports multiple readers at the same time as a writer
Supports snapshots
How Does WAL Mode Journaling Work?

Database
How Does WAL Mode Journaling Work?

Database

Write Ahead Log
How Does WAL Mode Journaling Work?

Database

Write Ahead Log
How Does WAL Mode Journaling Work?

Database

Write Ahead Log
How Does WAL Mode Journaling Work?
How Does WAL Mode Journaling Work?

Database

Write Ahead Log
How Does WAL Mode Journaling Work?

Database

Write Ahead Log
WAL mode is more efficient for most use cases.
Transactions
Transactions Help Reduce Writes

Use for multiple INSERT, UPDATE, and DELETE statements
Transactions Help Reduce Writes

Use for multiple `INSERT`, `UPDATE`, and `DELETE` statements.

Pages that are changed by multiple statements are only written once.
Multiple Single Statement Transactions

Transaction #1

Transaction #2

Transaction #3
Multiple Statement Transaction

Transaction #1 with multiple statements
Transactions Help Reduce Writes

Use for multiple INSERT, UPDATE, and DELETE statements

Pages that are changed by multiple statements are only written once

Useful for aggregating changes over time!
File Size and Privacy
File Size and Privacy

What happens when we delete data from a database?

Space containing the deleted data is marked as free

While no longer part of the database the deleted data is still on disk
File Size and Privacy

How do we securely delete sensitive data?
File Size and Privacy

PRAGMA schema.secure_delete=FAST;

How do we securely delete sensitive data?
How do we securely delete sensitive data?

• Automatically zeros deleted data
• No cost for data within the same page as the header
• Default behavior starting with iOS 13
File Size and Privacy

Don't use VACUUM
File Size and Privacy

Don't use VACUUM

• VACUUM is a slow I/O intensive operation
Why Is VACUUM Expensive?
Why Is **VACUUM** Expensive?

Database

Journal
Why Is VACUUM Expensive?

Database

Journal
Why Is VACUUM Expensive?
Why Is **VACUUM** Expensive?

Database
File Size and Privacy

Don't use `VACUUM`

- `VACUUM` is a slow I/O intensive operation
- All valid data gets written at least twice!
PRAGMA schema.auto_vacuum=INCREMENTAL;

PRAGMA schema.incremental_vacuum=(N);
// 'N' is the number pages to be vacuumed
How Is Incremental Auto Vacuum More Efficient?

Database

Write Ahead Log
How Is Incremental Auto Vacuum More Efficient?

Database

Write Ahead Log
How Is Incremental Auto Vacuum More Efficient?

Database

Write Ahead Log

Updated parent node

Migrated page

Migrated page

Truncated Database
Use incremental auto vacuum and fast secure delete to manage both file size and privacy.
Partial Indexes
Indexes

Faster ORDER BY, GROUP BY, and WHERE clauses

Each index adds additional I/O when writing to the database
Partial Indexes

Indexes with a `WHERE` clause

Partial indexes only store data which matches the `WHERE` clause

Queries that do not match the `WHERE` clause cannot use the index
SQLite Summary
SQLite Summary
Best practices

Keep database connections open
SQLite Summary
Best practices

Keep database connections open

Use WAL mode
SQLite Summary

Best practices

Keep database connections open

Use WAL mode

Multiple statements per transaction
SQLite Summary
Best practices

Keep database connections open
Use WAL mode
Multiple statements per transaction
Fast secure delete and auto vacuum incremental
SQLite Summary

Best practices

Keep database connections open

Use WAL mode

Multiple statements per transaction

Fast secure delete and auto vacuum incremental

Partial indexes
File Activity Instrument

Alejandro Lucena, Developer Tools
File Activity Instrument
File Activity Instrument

Supports all Apple devices
File Activity Instrument

Supports all Apple devices

Support for tracing single processes and all processes
File Activity Instrument

Supports all Apple devices

Support for tracing single processes and all processes

Obtains both logical and physical I/O information
File Activity Instrument

- Supports all Apple devices
- Support for tracing single processes and all processes
- Obtains both logical and physical I/O information
- Offers automated reasoning
Automated Reasoning
Automated Reasoning

Excessive physical writes
Automated Reasoning

Excessive physical writes

Failed I/O related calls
Automated Reasoning

- Excessive physical writes
- Failed I/O related calls
- Suboptimal caching
File Activity Instrument

This template records and analyzes filesystem and disk I/O activity. Recommendations are generated if the template detects behavior that is symptomatic of general I/O anti-patterns.
File Activity Instrument
Open and Close per Operation
Disk Usage

- **Filesystem Suggestions**
  - Instrument
  - High
  - Moderate
  - Low

- **Filesystem Activity**
  - Logical Write
  - Logical Read

- **Disk Usage**
  - Physical Write
  - Physical Read

- **Disk I/O Latency**
  - Max Latency

- **ImageViewer**
  - Process: 62943
  - Max Write...
  - Max Read...

**Disk Usage > Disk I/O Statistics**

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Latency</th>
<th>Avg Latency</th>
<th>Std Dev Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>27.28 ms</td>
<td>18.34 µs</td>
<td>27.23 µs</td>
<td>21.14 µs</td>
<td>281.45 µs</td>
</tr>
<tr>
<td>Data Write</td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>27.28 ms</td>
<td>18.34 µs</td>
<td>27.23 µs</td>
<td>21.14 µs</td>
<td>281.45 µs</td>
</tr>
</tbody>
</table>
Open and Close per Operation

Disk Usage

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Latency</th>
<th>Avg Latency</th>
<th>Std Dev Time</th>
<th>Max Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>27.28 ms</td>
<td>18.34 μs</td>
<td>27.23 μs</td>
<td>21.14 μs</td>
<td>281.45 μs</td>
</tr>
<tr>
<td>Data Write</td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>27.28 ms</td>
<td>18.34 μs</td>
<td>27.23 μs</td>
<td>21.14 μs</td>
<td>281.45 μs</td>
</tr>
</tbody>
</table>
### Open and Close per Operation

**Disk Usage — Detail view**

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Lanten</th>
<th>Min Total Lanten</th>
<th>Avg Total Lanten</th>
<th>Std Dev Total Lanten</th>
<th>Max Total Lanten</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="arrow_icon" alt="Expand All" /> <strong>All</strong></td>
<td>1,003</td>
<td>5.84 MiB</td>
<td>25.71 ms</td>
<td>18.42 µs</td>
<td>25.63 µs</td>
<td>22.17 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td><img src="arrow_icon" alt="Expand Data Write" /> Data Write</td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>25.52 ms</td>
<td>18.42 µs</td>
<td>25.47 µs</td>
<td>21.59 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td><img src="arrow_icon" alt="Expand ImageViewer (61960)" /> ImageViewer (61960)</td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>25.52 ms</td>
<td>18.42 µs</td>
<td>25.47 µs</td>
<td>21.59 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td>/ImageViewer/open_close_imageviewer.db-journal</td>
<td>747</td>
<td>3.89 MiB</td>
<td>19.71 ms</td>
<td>18.42 µs</td>
<td>26.39 µs</td>
<td>24.86 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td>/ImageViewer/open_close_imageviewer.db</td>
<td>255</td>
<td>1.95 MiB</td>
<td>5.81 ms</td>
<td>19.26 µs</td>
<td>22.79 µs</td>
<td>3.50 µs</td>
<td>39.47 µs</td>
</tr>
<tr>
<td><img src="arrow_icon" alt="Expand Data Read" /> Data Read</td>
<td>1</td>
<td>4.00 KiB</td>
<td>186.51 µs</td>
<td>186.51 µs</td>
<td>186.51 µs</td>
<td>n/a</td>
<td>186.51 µs</td>
</tr>
</tbody>
</table>
## Open and Close per Operation
### Disk Usage — Detail view

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total Latency</th>
<th>Avg Total Latency</th>
<th>Std Dev Total Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>*** All ***</td>
<td>1,003</td>
<td>5.84 MiB</td>
<td>25.71 ms</td>
<td>18.42 µs</td>
<td>25.63 µs</td>
<td>22.17 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td>Data Write</td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>25.52 ms</td>
<td>18.42 µs</td>
<td>25.47 µs</td>
<td>21.59 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td><strong>ImageViewer (61960)</strong></td>
<td>1,002</td>
<td>5.84 MiB</td>
<td>25.52 ms</td>
<td>18.42 µs</td>
<td>25.47 µs</td>
<td>21.59 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td>/ImageViewer/open_close_imageviewer.db-journal</td>
<td>747</td>
<td>3.89 MiB</td>
<td>19.71 ms</td>
<td>18.42 µs</td>
<td>26.39 µs</td>
<td>24.86 µs</td>
<td>302.64 µs</td>
</tr>
<tr>
<td>/ImageViewer/open_close_imageviewer.db</td>
<td>255</td>
<td>1.95 MiB</td>
<td>5.81 ms</td>
<td>19.26 µs</td>
<td>22.79 µs</td>
<td>3.50 µs</td>
<td>39.47 µs</td>
</tr>
<tr>
<td>Data Read</td>
<td>1</td>
<td>4.00 KiB</td>
<td>186.51 µs</td>
<td>186.51 µs</td>
<td>186.51 µs</td>
<td>n/a</td>
<td>186.51 µs</td>
</tr>
</tbody>
</table>
Open and Close per Operation

Filesystem Suggestions

Filesystem Suggestions

Filesystem Activity

Disk Usage

Disk I/O Latency

ImageViewer

Significance / Type / Process / Description | Count
--- | ---
* All * | 12
Moderate | 12
Open and Close per Operation

Filesystem Suggestions

Filesystem Suggestions

Filesystem Activity

Disk Usage

Disk I/O Latency

ImageViewer

Filesystem Suggestions

Significance / Type / Process / Description | Count
--- | ---
* All * | 12
* Moderate * | 12
## Open and Close per Operation

### Filesystem Suggestions — Detail view

<table>
<thead>
<tr>
<th>Significance / Type / Process / Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moderate</td>
<td>12</td>
</tr>
<tr>
<td>Excessive Writes</td>
<td>12</td>
</tr>
<tr>
<td>ImageViewer (61960)</td>
<td>12</td>
</tr>
</tbody>
</table>

- ImageViewer (61960) performed 83 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 84 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 81 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 82 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 79 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 88 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 99 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 64 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 77 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 90 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 102 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
- ImageViewer (61960) performed 61 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
Open and Close per Operation

Filesystem Activity

Filesystem Activity

- Logical Writes
- Logical Reads

Disk Usage

- Physical Writes
- Physical Reads

Disk I/O Latency

- Physical I/O C...

Filesystem Activity

- Filesystem Calltree
- Filesystem Events
- File Descriptor History
- File Descriptor Time Slice

<table>
<thead>
<tr>
<th>Filesystem / Process</th>
<th>Count</th>
<th>Duration</th>
<th>Min Duration</th>
<th>Avg Duration</th>
<th>Std Dev...</th>
<th>Max Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>10,782</td>
<td>268.77 ms</td>
<td>299 ns</td>
<td>24.00 µs</td>
<td>61.35</td>
<td>762.76 µs</td>
</tr>
<tr>
<td>fonti</td>
<td>3,099</td>
<td>4.15 ms</td>
<td>299 ns</td>
<td>1.34 µs</td>
<td>772 ns</td>
<td>19.75 µs</td>
</tr>
<tr>
<td>pwrite</td>
<td>2,490</td>
<td>34.80 ms</td>
<td>3.10 µs</td>
<td>13.96 µs</td>
<td>17.82</td>
<td>377.81 µs</td>
</tr>
<tr>
<td>write_no_cancel</td>
<td>1,044</td>
<td>2.88 ms</td>
<td>538 ns</td>
<td>2.79 µs</td>
<td>3.36 µs</td>
<td>29.00 µs</td>
</tr>
<tr>
<td>stat64</td>
<td>1,044</td>
<td>18.91 ms</td>
<td>3.18 µs</td>
<td>18.11 µs</td>
<td>68.95</td>
<td>123.17 µs</td>
</tr>
<tr>
<td>fsync</td>
<td>998</td>
<td>123.29 ms</td>
<td>589 ns</td>
<td>123.73 µs</td>
<td>155.92</td>
<td>389.50 µs</td>
</tr>
<tr>
<td>fsync</td>
<td>786</td>
<td>1.07 ms</td>
<td>451 ns</td>
<td>1.36 µs</td>
<td>958 ns</td>
<td>15.77 µs</td>
</tr>
<tr>
<td>pread</td>
<td>537</td>
<td>1.55 ms</td>
<td>1.23 µs</td>
<td>2.80 µs</td>
<td>1.44 µs</td>
<td>16.87 µs</td>
</tr>
<tr>
<td>fstatfs4</td>
<td>255</td>
<td>609.26 µs</td>
<td>1.45 µs</td>
<td>2.39 µs</td>
<td>663 ns</td>
<td>5.69 µs</td>
</tr>
<tr>
<td>lstat64</td>
<td>252</td>
<td>1.13 ms</td>
<td>2.29 µs</td>
<td>4.48 µs</td>
<td>4.46 µs</td>
<td>39.78 µs</td>
</tr>
<tr>
<td>unlink</td>
<td>249</td>
<td>66.71 ms</td>
<td>172.32 µs</td>
<td>275.96 µs</td>
<td>78.97</td>
<td>762.76 µs</td>
</tr>
<tr>
<td>close_no_cancel</td>
<td>6</td>
<td>22.60 µs</td>
<td>2.73 µs</td>
<td>3.78 µs</td>
<td>587 ns</td>
<td>4.49 µs</td>
</tr>
<tr>
<td>fcntl_no_cancel</td>
<td>6</td>
<td>12.70 µs</td>
<td>1.74 µs</td>
<td>2.13 µs</td>
<td>287 ns</td>
<td>2.38 µs</td>
</tr>
<tr>
<td>open_no_cancel</td>
<td>6</td>
<td>940.37 µs</td>
<td>138.33 µs</td>
<td>165.73 µs</td>
<td>19.72</td>
<td>192.59 µs</td>
</tr>
<tr>
<td>stats64</td>
<td>6</td>
<td>984.71 µs</td>
<td>11.89 µs</td>
<td>16.41 µs</td>
<td>3.49 µs</td>
<td>22.61 µs</td>
</tr>
<tr>
<td>getattrlist</td>
<td>6</td>
<td>600.83 µs</td>
<td>85.57 µs</td>
<td>100.14 µs</td>
<td>12.97</td>
<td>122.39 µs</td>
</tr>
</tbody>
</table>
Open and Close as Needed

Disk Usage
## Disk Usage — Detail view

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total Latency</th>
<th>Avg Total Latency</th>
<th>Std Dev Total Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>54</td>
<td>288.00 Kib</td>
<td>3.28 ms</td>
<td>18.71 µs</td>
<td>60.71 µs</td>
<td>75.84 µs</td>
<td>275.92 µs</td>
</tr>
<tr>
<td>Data Write</td>
<td>54</td>
<td>288.00 Kib</td>
<td>3.28 ms</td>
<td>18.71 µs</td>
<td>60.71 µs</td>
<td>75.84 µs</td>
<td>275.92 µs</td>
</tr>
<tr>
<td>ImageViewer (62282)</td>
<td>54</td>
<td>288.00 Kib</td>
<td>3.28 ms</td>
<td>18.71 µs</td>
<td>60.71 µs</td>
<td>75.84 µs</td>
<td>275.92 µs</td>
</tr>
<tr>
<td>/ImageViewer/open_close_as_needed_imageviewer.db-journal</td>
<td>36</td>
<td>192.00 Kib</td>
<td>2.79 ms</td>
<td>18.71 µs</td>
<td>77.46 µs</td>
<td>88.50 µs</td>
<td>275.92 µs</td>
</tr>
<tr>
<td>/Imageviewer/open_close_as_needed_imageviewer.db</td>
<td>18</td>
<td>96.00 Kib</td>
<td>489.55 µs</td>
<td>19.98 µs</td>
<td>27.20 µs</td>
<td>5.02 µs</td>
<td>35.20 µs</td>
</tr>
</tbody>
</table>
# Disk Usage — Detail view

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total Latency</th>
<th>Avg Total Latency</th>
<th>Std Dev Total Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>▼ * All *</td>
<td>54</td>
<td>288.00 KIB</td>
<td>3.28 ms</td>
<td>18.71 μs</td>
<td>60.71 μs</td>
<td>75.84 μs</td>
<td>275.92 μs</td>
</tr>
<tr>
<td>▼ Data Write</td>
<td>54</td>
<td>288.00 KIB</td>
<td>3.28 ms</td>
<td>18.71 μs</td>
<td>60.71 μs</td>
<td>75.84 μs</td>
<td>275.92 μs</td>
</tr>
<tr>
<td>▼ ImageViewer (82282)</td>
<td>54</td>
<td>288.00 KIB</td>
<td>3.28 ms</td>
<td>18.71 μs</td>
<td>60.71 μs</td>
<td>75.84 μs</td>
<td>275.92 μs</td>
</tr>
<tr>
<td>/ImageViewer/open_close_as_needed_imageviewer.db-journal</td>
<td>36</td>
<td>192.00 KIB</td>
<td>2.79 ms</td>
<td>18.71 μs</td>
<td>77.46 μs</td>
<td>88.50 μs</td>
<td>275.92 μs</td>
</tr>
<tr>
<td>/ImageViewer/open_close_as_needed_imageviewer.db</td>
<td>18</td>
<td>96.00 KIB</td>
<td>489.55 μs</td>
<td>19.98 μs</td>
<td>27.20 μs</td>
<td>5.02 μs</td>
<td>35.20 μs</td>
</tr>
</tbody>
</table>
Open and Close as Needed

Filesystem Suggestions

Filesystem Suggestions

Significance / Type / Process / Description | Count
--- | ---
* All * | 3
Moderate | 3
Excessive Writes | 3
Open and Close as Needed

Filesystem Suggestions — Detail view

<table>
<thead>
<tr>
<th>Significance / Type / Process / Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All</td>
<td>3</td>
</tr>
<tr>
<td>▼ Moderate</td>
<td>3</td>
</tr>
<tr>
<td>▼ Excessive Writes</td>
<td>3</td>
</tr>
<tr>
<td>▼ ImageViewer (62282)</td>
<td>3</td>
</tr>
</tbody>
</table>

**ImageViewer (62282)** performed 17 physical writes within 1 second. This can put excessive strain on physical storage media and hinder application performance.
Delete Mode Journaling
## Delete Mode Journaling

### Disk Usage & Disk I/O Statistics

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total Latency</th>
<th>Avg Total Latency</th>
<th>Std Dev Total Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>▼ * All *</td>
<td>54</td>
<td>288.00 KiB</td>
<td>2.40 ms</td>
<td>18.98 µs</td>
<td>44.42 µs</td>
<td>57.65 µs</td>
<td>275.53 µs</td>
</tr>
<tr>
<td>▼ Data Write</td>
<td>54</td>
<td>288.00 KiB</td>
<td>2.40 ms</td>
<td>18.98 µs</td>
<td>44.42 µs</td>
<td>57.65 µs</td>
<td>275.53 µs</td>
</tr>
<tr>
<td>▼ ImageViewer (63410)</td>
<td>54</td>
<td>288.00 KiB</td>
<td>2.40 ms</td>
<td>18.98 µs</td>
<td>44.42 µs</td>
<td>57.65 µs</td>
<td>275.53 µs</td>
</tr>
<tr>
<td>/ImageViewer/delete_mode.db-journal</td>
<td>36</td>
<td>192.00 KiB</td>
<td>1.96 ms</td>
<td>18.98 µs</td>
<td>54.33 µs</td>
<td>68.71 µs</td>
<td>275.53 µs</td>
</tr>
<tr>
<td>/ImageViewer/delete_mode.db</td>
<td>18</td>
<td>96.00 KiB</td>
<td>443.16 µs</td>
<td>19.35 µs</td>
<td>24.62 µs</td>
<td>4.24 µs</td>
<td>35.66 µs</td>
</tr>
</tbody>
</table>
WAL Mode Journaling
WAL Mode Journaling

Filesystem Suggestions
- High
- Moderate
- Low

Filesystem Activity
- Logical Write
- Logical Read

Disk Usage
- Physical Write
- Physical Read

Disk I/O Latency
- Physical I/O
Journaling Comparison

- WAL Journaling
- Delete Mode Journaling
## Journaling Mode Comparison

### WAL Journaling

#### Filesystem Activity

<table>
<thead>
<tr>
<th>Syscall / Process</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>151</td>
</tr>
<tr>
<td>fsync</td>
<td>1</td>
</tr>
<tr>
<td>ftruncate</td>
<td>2</td>
</tr>
<tr>
<td>stat64</td>
<td>2</td>
</tr>
<tr>
<td>pread</td>
<td>5</td>
</tr>
<tr>
<td>fstat64</td>
<td>5</td>
</tr>
<tr>
<td>pwrite</td>
<td>9</td>
</tr>
<tr>
<td>write_nocancel</td>
<td>62</td>
</tr>
<tr>
<td>fcntl</td>
<td>65</td>
</tr>
</tbody>
</table>

### Delete Mode Journaling

#### Filesystem Activity

<table>
<thead>
<tr>
<th>Syscall / Process</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>240</td>
</tr>
<tr>
<td>fstatfs64</td>
<td>4</td>
</tr>
<tr>
<td>unlink</td>
<td>4</td>
</tr>
<tr>
<td>pread</td>
<td>12</td>
</tr>
<tr>
<td>fstat64</td>
<td>16</td>
</tr>
<tr>
<td>fsync</td>
<td>16</td>
</tr>
<tr>
<td>stat64</td>
<td>24</td>
</tr>
<tr>
<td>pwrite</td>
<td>40</td>
</tr>
<tr>
<td>write_nocancel</td>
<td>40</td>
</tr>
<tr>
<td>fcntl</td>
<td>84</td>
</tr>
</tbody>
</table>
Single Statement Transactions
### Single Statement Transactions

#### Filesystem Suggestions
- **Instrument**
- **High**
- **Moderate**
- **Low**

#### Filesystem Activity
- **Logical Write (LW)**
- **Logical Read (LR)**

#### Disk Usage
- **Physical Write (PW)**
- **Physical Read (PR)**

#### Disk I/O Latency
- **Physical I/O**

#### Filesystem Activity & Filesystem Statistics

<table>
<thead>
<tr>
<th>Syscall / Process</th>
<th>Count</th>
<th>Bytes</th>
<th>Duration</th>
<th>Min Duration</th>
<th>Avg Duration</th>
<th>Std Dev Duration</th>
<th>Max Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>111</td>
<td>49.65 KB</td>
<td>3.63 ms</td>
<td>377 ns</td>
<td>32.72 μs</td>
<td>84.99 μs</td>
<td>474.66 μs</td>
</tr>
<tr>
<td>fontl</td>
<td>36</td>
<td>0 Bytes</td>
<td>53.25 μs</td>
<td>377 ns</td>
<td>1.48 μs</td>
<td>1.01 μs</td>
<td>5.15 μs</td>
</tr>
<tr>
<td>fwrite</td>
<td>30</td>
<td>40.56 KB</td>
<td>371.09 μs</td>
<td>3.80 μs</td>
<td>12.37 μs</td>
<td>10.06 μs</td>
<td>43.81 μs</td>
</tr>
<tr>
<td>barfile</td>
<td>83</td>
<td>0 Bytes</td>
<td>1.65 μs</td>
<td>1.06 μs</td>
<td>157.75 μs</td>
<td>110.45 μs</td>
<td>224.88 μs</td>
</tr>
<tr>
<td>statfs4</td>
<td>12</td>
<td>0 Bytes</td>
<td>249.94 μs</td>
<td>12.36 μs</td>
<td>20.83 μs</td>
<td>8.82 μs</td>
<td>40.45 μs</td>
</tr>
<tr>
<td>fstatfs4</td>
<td>9</td>
<td>0 Bytes</td>
<td>12.77 μs</td>
<td>616 ns</td>
<td>1.42 μs</td>
<td>871 ns</td>
<td>2.84 μs</td>
</tr>
<tr>
<td>pread</td>
<td>6</td>
<td>72 Bytes</td>
<td>19.17 μs</td>
<td>1.56 μs</td>
<td>3.19 μs</td>
<td>1.94 μs</td>
<td>6.26 μs</td>
</tr>
<tr>
<td>fstatfs4</td>
<td>3</td>
<td>0 Bytes</td>
<td>7.17 μs</td>
<td>1.72 μs</td>
<td>2.39 μs</td>
<td>0.26 μs</td>
<td>2.95 μs</td>
</tr>
<tr>
<td>unlimit</td>
<td>3</td>
<td>0 Bytes</td>
<td>1.26 ms</td>
<td>389.39 ms</td>
<td>421.66 ms</td>
<td>46.26 ms</td>
<td>474.66 ms</td>
</tr>
</tbody>
</table>
Single Statement Transactions

Filesystem Suggestions
- Instrument
- High
- Moderate
- Low

Filesystem Activity
- Instrument
- Logical Write
- Logical Read

Disk Usage
- Instrument
- Physical Write
- Physical Read

Disk I/O Latency
- Instrument
- Physical I/O

Disk Usage & Disk I/O Statistics

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Duration</th>
<th>Min Duration</th>
<th>Avg Duration</th>
<th>Std Dev Dur</th>
<th>Max Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>12</td>
<td>72.00 Kib</td>
<td>1.58 ms</td>
<td>35.52 μs</td>
<td>131.40 μs</td>
<td>73.94 μs</td>
<td>227.57 μs</td>
</tr>
<tr>
<td>Data Write</td>
<td>12</td>
<td>72.00 Kib</td>
<td>1.58 ms</td>
<td>35.52 μs</td>
<td>131.40 μs</td>
<td>73.94 μs</td>
<td>227.57 μs</td>
</tr>
<tr>
<td>ImageViewer (64384)</td>
<td>12</td>
<td>72.00 Kib</td>
<td>1.58 ms</td>
<td>35.52 μs</td>
<td>131.40 μs</td>
<td>73.94 μs</td>
<td>227.57 μs</td>
</tr>
</tbody>
</table>
## Multiple Statement Transaction

### Files System Suggestion
- High
- Moderate
- Low

### File System Activity
- Logical Writes
- Logical Reads

### Disk Usage
- Physical Writes
- Physical Reads

### Disk I/O Latency
- Physical I/O...

### File System Activity > System Statistics

<table>
<thead>
<tr>
<th>Syscall / Process</th>
<th>Count</th>
<th>Bytes</th>
<th>Duration</th>
<th>Min Duration</th>
<th>Avg Duration</th>
<th>Std Dev</th>
<th>Duration</th>
<th>Max Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>All *</td>
<td>37</td>
<td>16.55 KiB</td>
<td>945.08 μs</td>
<td>332 ns</td>
<td>25.54 μs</td>
<td>58.86 μs</td>
<td>262.89 μs</td>
<td></td>
</tr>
<tr>
<td>fcntl</td>
<td>12</td>
<td>0 Bytes</td>
<td>18.87 μs</td>
<td>332 ns</td>
<td>1.57 μs</td>
<td>1.56 μs</td>
<td>6.20 μs</td>
<td></td>
</tr>
<tr>
<td>fwrite</td>
<td>10</td>
<td>16.53 KiB</td>
<td>111.09 μs</td>
<td>3.12 μs</td>
<td>11.11 μs</td>
<td>10.16 μs</td>
<td>37.37 μs</td>
<td></td>
</tr>
<tr>
<td>fsync</td>
<td>4</td>
<td>0 Bytes</td>
<td>419.17 μs</td>
<td>731 ns</td>
<td>104.79 μs</td>
<td>97.07 μs</td>
<td>199.16 μs</td>
<td></td>
</tr>
<tr>
<td>stat64</td>
<td>4</td>
<td>0 Bytes</td>
<td>117.42 μs</td>
<td>13.51 μs</td>
<td>29.36 μs</td>
<td>18.04 μs</td>
<td>54.62 μs</td>
<td></td>
</tr>
<tr>
<td>fstat64</td>
<td>3</td>
<td>0 Bytes</td>
<td>4.90 μs</td>
<td>518 ns</td>
<td>1.63 μs</td>
<td>1.29 μs</td>
<td>3.04 μs</td>
<td></td>
</tr>
<tr>
<td>pread</td>
<td>2</td>
<td>24 Bytes</td>
<td>8.17 μs</td>
<td>1.55 μs</td>
<td>4.08 μs</td>
<td>3.58 μs</td>
<td>6.62 μs</td>
<td></td>
</tr>
<tr>
<td>unlink</td>
<td>1</td>
<td>0 Bytes</td>
<td>262.89 μs</td>
<td>262.89 μs</td>
<td>262.89 μs</td>
<td>n/a</td>
<td>262.89 μs</td>
<td></td>
</tr>
<tr>
<td>fstatfs64</td>
<td>1</td>
<td>0 Bytes</td>
<td>2.58 μs</td>
<td>2.58 μs</td>
<td>2.58 μs</td>
<td>n/a</td>
<td>2.58 μs</td>
<td></td>
</tr>
</tbody>
</table>
## Multiple Statement Transaction

### Filesystem Activity

<table>
<thead>
<tr>
<th>Filesystem Suggestions</th>
<th>High</th>
<th>Moderate</th>
<th>Low</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Filesystem Activity</th>
<th>Logical Writes</th>
<th>Logical Reads</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Disk Usage</th>
<th>Physical Writes</th>
<th>Physical Reads</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Disk I/O Latency</th>
<th>Physical I/O...</th>
</tr>
</thead>
</table>

### Filesystem Statistics

<table>
<thead>
<tr>
<th>Syscall / Process</th>
<th>Count</th>
<th>Bytes</th>
<th>Duration</th>
<th>Min Duration</th>
<th>Avg Duration</th>
<th>Std Dev Duration</th>
<th>Max Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>37</td>
<td>16.55 KB</td>
<td>945.08 µs</td>
<td>332 ns</td>
<td>25.54 µs</td>
<td>58.86 µs</td>
<td>262.89 µs</td>
</tr>
<tr>
<td><code>fcnt1</code></td>
<td>12</td>
<td>0 Bytes</td>
<td>18.87 µs</td>
<td>332 ns</td>
<td>1.57 µs</td>
<td>1.56 µs</td>
<td>6.20 µs</td>
</tr>
<tr>
<td><code>pwrite</code></td>
<td>10</td>
<td>16.53 KB</td>
<td>111.09 µs</td>
<td>3.12 µs</td>
<td>11.1 µs</td>
<td>10.16 µs</td>
<td>37.37 µs</td>
</tr>
<tr>
<td><code>fsync</code></td>
<td>4</td>
<td>0 Bytes</td>
<td>419.17 µs</td>
<td>731 ns</td>
<td>104.79 µs</td>
<td>97.07 µs</td>
<td>199.16 µs</td>
</tr>
<tr>
<td><code>stat64</code></td>
<td>4</td>
<td>0 Bytes</td>
<td>117.42 µs</td>
<td>13.51 µs</td>
<td>29.36 µs</td>
<td>18.04 µs</td>
<td>54.62 µs</td>
</tr>
<tr>
<td><code>fstat64</code></td>
<td>3</td>
<td>0 Bytes</td>
<td>4.90 µs</td>
<td>518 ns</td>
<td>1.63 µs</td>
<td>1.29 µs</td>
<td>3.04 µs</td>
</tr>
<tr>
<td><code>pread</code></td>
<td>2</td>
<td>24 Bytes</td>
<td>8.17 µs</td>
<td>1.55 µs</td>
<td>4.08 µs</td>
<td>3.58 µs</td>
<td>6.62 µs</td>
</tr>
<tr>
<td><code>unlink</code></td>
<td>1</td>
<td>0 Bytes</td>
<td>262.89 µs</td>
<td>262.89 µs</td>
<td>262.89 µs</td>
<td>n/a</td>
<td>262.89 µs</td>
</tr>
<tr>
<td><code>fstatfs64</code></td>
<td>1</td>
<td>0 Bytes</td>
<td>2.58 µs</td>
<td>2.58 µs</td>
<td>2.58 µs</td>
<td>n/a</td>
<td>2.58 µs</td>
</tr>
</tbody>
</table>
Multiple Statement Transaction
Multiple Statement Transaction
### Transactions Comparison

#### Multiple Statement

<table>
<thead>
<tr>
<th>Syscall / Process</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>37</td>
</tr>
<tr>
<td>fcntl</td>
<td>12</td>
</tr>
<tr>
<td>pwrite</td>
<td>10</td>
</tr>
<tr>
<td>fsync</td>
<td>4</td>
</tr>
<tr>
<td>stat64</td>
<td>4</td>
</tr>
<tr>
<td>fstat64</td>
<td>3</td>
</tr>
<tr>
<td>pread</td>
<td>2</td>
</tr>
<tr>
<td>unlink</td>
<td>1</td>
</tr>
<tr>
<td>fstatfs64</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Single Statement

<table>
<thead>
<tr>
<th>Syscall / Process</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>111</td>
</tr>
<tr>
<td>fcntl</td>
<td>36</td>
</tr>
<tr>
<td>pwrite</td>
<td>30</td>
</tr>
<tr>
<td>fsync</td>
<td>12</td>
</tr>
<tr>
<td>stat64</td>
<td>12</td>
</tr>
<tr>
<td>fstat64</td>
<td>9</td>
</tr>
<tr>
<td>pread</td>
<td>6</td>
</tr>
<tr>
<td>fstatfs64</td>
<td>3</td>
</tr>
<tr>
<td>unlink</td>
<td>3</td>
</tr>
</tbody>
</table>
## Disk Usage

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total Latency</th>
<th>Avg Total Latency</th>
<th>Std Dev Total Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>27</td>
<td>168.00 KiB</td>
<td>1.16 ms</td>
<td>18.91 μs</td>
<td>43.12 μs</td>
<td>57.25 μs</td>
<td>276.55 μs</td>
</tr>
<tr>
<td>▼ Data Write</td>
<td>27</td>
<td>168.00 KiB</td>
<td>1.16 ms</td>
<td>18.91 μs</td>
<td>43.12 μs</td>
<td>57.25 μs</td>
<td>276.55 μs</td>
</tr>
<tr>
<td>▼ ImageViewer (65819)</td>
<td>27</td>
<td>168.00 KiB</td>
<td>1.16 ms</td>
<td>18.91 μs</td>
<td>43.12 μs</td>
<td>57.25 μs</td>
<td>276.55 μs</td>
</tr>
<tr>
<td>/ImageViewer/full_vacuum.db-journal</td>
<td>18</td>
<td>108.00 KiB</td>
<td>932.60 μs</td>
<td>18.93 μs</td>
<td>51.81 μs</td>
<td>68.99 μs</td>
<td>276.55 μs</td>
</tr>
<tr>
<td>/ImageViewer/full_vacuum.db</td>
<td>9</td>
<td>60.00 KiB</td>
<td>231.75 μs</td>
<td>18.91 μs</td>
<td>25.75 μs</td>
<td>5.52 μs</td>
<td>35.35 μs</td>
</tr>
</tbody>
</table>
### Disk I/O Statistics

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total</th>
<th>Avg Total</th>
<th>Std Dev Total</th>
<th>Max Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All</strong></td>
<td>27</td>
<td>168.00 KiB</td>
<td>1.16 ms</td>
<td>18.91 µs</td>
<td>43.12 µs</td>
<td>57.25 µs</td>
<td>276.55 µs</td>
</tr>
<tr>
<td>Data Write</td>
<td>27</td>
<td>168.00 KiB</td>
<td>1.16 ms</td>
<td>18.91 µs</td>
<td>43.12 µs</td>
<td>57.25 µs</td>
<td>276.55 µs</td>
</tr>
<tr>
<td>ImageViewer (65819)</td>
<td>27</td>
<td>168.00 KiB</td>
<td>1.16 ms</td>
<td>18.91 µs</td>
<td>43.12 µs</td>
<td>57.25 µs</td>
<td>276.55 µs</td>
</tr>
<tr>
<td>/Image Viewer/full_vacuum.db-journal</td>
<td>18</td>
<td>108.00 KiB</td>
<td>51.81 µs</td>
<td>18.93 µs</td>
<td>68.99 µs</td>
<td>276.55 µs</td>
<td></td>
</tr>
<tr>
<td>/Image Viewer/full_vacuum.db</td>
<td>9</td>
<td>60.00 KiB</td>
<td>25.75 µs</td>
<td>18.91 µs</td>
<td>5.52 µs</td>
<td>35.35 µs</td>
<td></td>
</tr>
</tbody>
</table>
## Incremental Vacuum

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total Latency</th>
<th>Avg Total Latency</th>
<th>Std Dev Total Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>12</td>
<td>72.00 KiB</td>
<td>882.60 μs</td>
<td>23.22 μs</td>
<td>73.55 μs</td>
<td>88.51 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>Data Write</td>
<td>12</td>
<td>72.00 KiB</td>
<td>882.60 μs</td>
<td>23.22 μs</td>
<td>73.55 μs</td>
<td>88.51 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>ImageViewer (66075)</td>
<td>12</td>
<td>72.00 KiB</td>
<td>882.60 μs</td>
<td>23.22 μs</td>
<td>73.55 μs</td>
<td>88.51 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>/ImageViewer/incremental_vacuum.db-journal</td>
<td>9</td>
<td>48.00 KiB</td>
<td>785.72 μs</td>
<td>23.22 μs</td>
<td>87.30 μs</td>
<td>99.60 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>/ImageViewer/incremental_vacuum.db</td>
<td>3</td>
<td>24.00 KiB</td>
<td>96.88 μs</td>
<td>31.62 μs</td>
<td>32.29 μs</td>
<td>805 ns</td>
<td>33.18 μs</td>
</tr>
</tbody>
</table>
## Incremental Vacuum

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
<th>Total Latency</th>
<th>Min Total Latency</th>
<th>Avg Total Latency</th>
<th>Std Dev To Total Latency</th>
<th>Max Total Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>* ▼ All *</td>
<td>12</td>
<td>72.00 KiB</td>
<td>882.60 μs</td>
<td>23.22 μs</td>
<td>73.55 μs</td>
<td>88.51 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>▼ Data Write</td>
<td>12</td>
<td>72.00 KiB</td>
<td>882.60 μs</td>
<td>23.22 μs</td>
<td>73.55 μs</td>
<td>88.51 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>▼ ImageViewer (66075)</td>
<td>12</td>
<td>72.00 KiB</td>
<td>882.60 μs</td>
<td>23.22 μs</td>
<td>73.55 μs</td>
<td>88.51 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>/ImageViewer/incremental_vacuum.db-journal</td>
<td>9</td>
<td>48.00 KiB</td>
<td>785.72 μs</td>
<td>23.22 μs</td>
<td>87.30 μs</td>
<td>99.60 μs</td>
<td>301.52 μs</td>
</tr>
<tr>
<td>/ImageViewer/incremental_vacuum.db</td>
<td>3</td>
<td>24.00 KiB</td>
<td>96.88 μs</td>
<td>31.62 μs</td>
<td>32.29 μs</td>
<td>805 ns</td>
<td>33.18 μs</td>
</tr>
</tbody>
</table>
### Vacuuming Comparison

#### Incremental

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>12</td>
<td>72.00 KIB</td>
</tr>
<tr>
<td>Data Write</td>
<td>12</td>
<td>72.00 KIB</td>
</tr>
<tr>
<td>ImageViewer (66075)</td>
<td>12</td>
<td>72.00 KIB</td>
</tr>
<tr>
<td>/ImageViewer/incremental_vacuum.db-journal</td>
<td>9</td>
<td>48.00 KIB</td>
</tr>
<tr>
<td>/ImageViewer/incremental_vacuum.db</td>
<td>3</td>
<td>24.00 KIB</td>
</tr>
</tbody>
</table>

#### Full

<table>
<thead>
<tr>
<th>Operation / Process / Path</th>
<th>Count</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>27</td>
<td>168.00 KIB</td>
</tr>
<tr>
<td>Data Write</td>
<td>27</td>
<td>168.00 KIB</td>
</tr>
<tr>
<td>ImageViewer (665819)</td>
<td>27</td>
<td>168.00 KIB</td>
</tr>
<tr>
<td>/ImageViewer/full_vacuum.db-journal</td>
<td>18</td>
<td>108.00 KIB</td>
</tr>
<tr>
<td>/ImageViewer/full_vacuum.db</td>
<td>9</td>
<td>60.00 KIB</td>
</tr>
</tbody>
</table>
Summary
Apply these lessons
Summary

Apply these lessons

Profile with File Activity Instrument
Summary

Apply these lessons

Profile with File Activity Instrument

Continue optimizing storage!