How to Copy Files

Yang Zhan\textsuperscript{1,2}, Alex Conway\textsuperscript{3,4}, Ian Groombridge\textsuperscript{5}, Yizheng Jiao\textsuperscript{1}, Nirjhar Mukherjee\textsuperscript{1}, Michael A. Bender\textsuperscript{6}, Martín Farach-Colton\textsuperscript{3}, William Jannen\textsuperscript{7}, Rob Johnson\textsuperscript{4}, Donald E. Porter\textsuperscript{1}, Jun Yuan\textsuperscript{5}
Copying is Ubiquitous and Important

- cp -r
- vmrun start
- container instantiation
- backup
Physical Copy
Copying

- Physical Copy
- High Latency
- High Space Use
Existing Logical Copy Implementations

**BTRFS**

Leverages the underlying copy-on-write B-tree to implement `cp --reflink`

**XFS**

Uses an update-in-place B-tree but supports sharing data blocks with copy-on-write via `cp --reflink`

**ZFS**

Implements a limited version of copy-on-write copying via `zfs clone`
Copying

Physical Copy
High Latency
High Space Use

Copy on Write
Copying

Physical Copy

High Latency

High Space Use

Copy on Write

Low Latency

Better Space Use

High Fragmentation
Copying

Physical Copy

High Latency

High Space Use

Copy on Write

Low Latency

Better Space Use

High Fragmentation
Space Amplification and Fragmentation

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk
Space Amplification and Fragmentation

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Space Amplification
additonal file system size / added data (1KiB)

BTRFS | XFS | ZFS

Lower is Better

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk
Space Amplification and Fragmentation

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Space Amplification
additional file system size / added data (1KiB)

Fragmentation
measured by timing a grep over the latest copy

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk
Space Amplification and Fragmentation

Init: Create 64 4MiB files with random data.
Each round: Logically copy all files, then change 16B in each file (1KiB total)

Fragmentation measured by timing a grep over the latest copy
Space Amplification additional file system size / added data (1KiB)

Have large space amplification

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

---

<table>
<thead>
<tr>
<th>Logical copy number</th>
<th>BTRFS</th>
<th>XFS</th>
<th>ZFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>32</td>
<td>34</td>
</tr>
</tbody>
</table>

---

Space Amplification
Lower is Better

BTRFS
XFS
ZFS

---

Have large space amplification
Space Amplification and Fragmentation

Initially, create 4 MiB files with random data. Each round, logically copy all files, then change 16B in each file (1 KiB total).

- **Lower is Better**
  - Space Amplification: additional file system size / added data (1 KiB)
  - Fragmentation: measured by timing a grep over the latest copy

**Have large space amplification**

**Or high fragmentation**

**Dell Optiplex 790**
- 4-core 3.40 GHz Intel Core i7 CPU
- 4 GiB RAM
- 500 GB 7200 RPM SATA disk
Copy Performance Goals
Copy Performance Goals

- Space efficient
- Low latency

Specific to Copying
Copy Performance Goals

- Space efficient
- Low latency

Specific to Copying

- Fast writes
- Fast reads

General file system
Copy Performance Goals

Specific to Copying:
- Space efficient
- Low latency

General file system:
- Fast writes
- Fast reads

Locality
Contributions of this Paper

A high performance logical copy implementation...
Contributions of this Paper

A high performance logical copy implementation...

In BεtrFS, which leverages the properties of Copy-on-Write Bε-trees
Contributions of this Paper

A high performance logical copy implementation...

In BεtrFS, which leverages the properties of Copy-on-Write Bε-trees

- Space efficient
- Low latency
- Copy-specific

- Fast writes
- Fast reads
- General file system
Contributions of this Paper

A high performance logical copy implementation...

In BεtrFS, which leverages the properties of Copy-on-Write Bε-trees

- Space efficient ✔️
- Low latency ✔️
- Fast writes ✔️
- Fast reads ✔️
- Copy-specific
- General file system
What is the Challenge of Logical Copy?
Example: Logical copy in an inode file system
Logical Copy in an Inode File System

Copy /foo to /bar
Copy /foo to /bar
Copy /foo to /bar
Copy /foo to /bar
Logical Copy in an Inode File System

Copy /foo to /bar

- Low latency
- Space efficient
- Fast Reads
- Fast Writes

original

foo

fred

bar

copy
Logical Copy in an Inode File System

Copy /foo to /bar

- Low latency: ✔️
- Space efficient: ??
- Fast Reads: ??
- Fast Writes: ??

Copy /foo to /bar
Copy /foo to /bar

Logical Copy in an Inode File System

- Low latency  ✓
- Space efficient  ❓
- Fast Reads  ❓
- Fast Writes  ❓

change 1 bit here
Logical Copy in an Inode File System

Copy /foo to /bar

- Low latency: ✓
- Space efficient: ?
- Fast Reads: ?
- Fast Writes: ?

Copy /foo to /bar

- Change 1 bit here
- Copy with new bit

Copy with new bit
Copy /foo to /bar

- Low latency: ✔️
- Space efficient: ❓
- Fast Reads: ❓
- Fast Writes: ❓

Added a whole data block to change 1 bit

change 1 bit here

copy with new bit
Copy /foo to /bar

- Low latency ✓
- Space efficient
- Fast Reads
- Fast Writes

Added a whole data block to change 1 bit

This is at least 4KiB and can be more
Copy /foo to /bar

- Low latency: ✔️
- Space efficient: ✗
- Fast Reads: ✔️
- Fast Writes: ✗

Added a whole data block to change 1 bit

This is at least 4KiB and can be more
Logical Copy in an Inode File System

Copy /foo to /bar

- Low latency: ✔️
- Space efficient: ❌
- Fast Reads: ?
- Fast Writes: ?

Copy /foo to /bar
Logical Copy in an Inode File System

Copy /foo to /bar

- Low latency: ✓
- Space efficient: ✗
- Fast Reads: ?
- Fast Writes: ?

no locality guarantees between data blocks
Copy /foo to /bar

- Low latency: ✓
- Space efficient: ✗
- Fast Reads: ❓
- Fast Writes: ❓

no locality guarantees between data blocks

only have locality if the blocks are large
Copy /foo to /bar

- Low latency: ✓
- Space efficient: ✗

Logical Copy in an Inode File System

### Original Folder Structure
- `/`: Root
- `/foo`
- `/bar`

### Copy Folder Structure
- `/fred`
- `/copy`

#### Observations
- **No locality guarantees between data blocks**
- **Low latency**
- **Space efficient**: ✗
- **Fast Reads**: ✓
- **Fast Writes**: ✗

#### Additional Notes
- **Usually 4KiB** ⇒ too small for locality
- **Only have locality if the blocks are large**
Logical Copy in an Inode File System

Copy /foo to /bar

- Low latency: ✓
- Space efficient: ✗
- Fast Reads: ✗
- Fast Writes: ✗

- usually 4KiB ⇒ too small for locality
- only have locality if the blocks are large
- no locality guarantees between data blocks
Space-Locality Tradeoff

- Larger blocks
  - Better locality
  - Worse space efficiency
Space-Locality Tradeoff

Larger blocks:
- Better locality
- Worse space efficiency

Smaller blocks:
- Worse locality
- Better space efficiency
Space-Locality Tradeoff

Larger blocks:
- Better locality
- Worse space efficiency

Smaller blocks:
- Too large for space
- Too small for locality
- Better space efficiency

4KiB blocks:
- Too large for space
- Too small for locality
- 😭
Inode Logical Copy Takeaway

Using a DAG to share data is great for latency
Inode Logical Copy Takeaway

Using a DAG to share data is great for latency

Challenge: Small writes break sharing
Our Solution: $B^\varepsilon$-DAGs

$B^\varepsilon$trFS

$B^\varepsilon$-Trees $\rightarrow$ $B^\varepsilon$-DAGs
Our Solution: $B^\varepsilon$-DAGs

$B^\varepsilon$-trees have good locality and batch together small writes
Our Solution: $B^\varepsilon$-DAGs

$B^\varepsilon$-trees have good locality and batch together small writes

In this paper, we turn $B^\varepsilon$-trees into $B^\varepsilon$-DAGs to share data between files
B\(\varepsilon\)-Trees
A $B^\varepsilon$-tree is a search tree (like a B-tree)
A $\beta$-tree is a search tree (like a B-tree)

- pivots
- the rest buffer

Directory tree:

- B D H
- A B
- R

Diagram:

- [Diagram showing a $\beta$-tree structure]
Bε-Trees

A Bε-tree is a search tree (like a B-tree)

directory tree

/og/n/A

/

pivots

the rest buffer
A $B^\varepsilon$-tree is a search tree (like a B-tree)

**B$^\varepsilon$-Trees**

Inserts get put in the root buffer

New file: /orange/D
A $B^\varepsilon$-tree is a search tree (like a B-tree).

Inserts get put in the root buffer.
A $\textbf{B}^\varepsilon$-tree is a search tree (like a B-tree).

Inserts get put in the root buffer.
A Bε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer
A Bε-tree is a search tree (like a B-tree)

 Inserts get put in the root buffer
A $B^\varepsilon$-tree is a search tree (like a B-tree).

Inserts get put in the root buffer.

(directory tree)

$B^\varepsilon$-Trees
A $B^\varepsilon$-tree is a search tree (like a B-tree). Inserts get put in the root buffer.
Bε-Trees

A Bε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer
**Bε-Trees**

A Bε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer
A $\textbf{B}^\varepsilon$-tree is a search tree (like a B-tree)
**Bε-Trees**

A Bε-tree is a search tree (like a B-tree)

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer

Inserts get put in the root buffer
A $B^\varepsilon$-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer
A $\beta$-tree is a search tree (like a B-tree).

Inserts get put in the root buffer.

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer
A $\mathbb{B}\varepsilon$-tree is a search tree (like a B-tree)

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer

Inserts get put in the root buffer
A $\beta$-tree is a search tree (like a B-tree)

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer

Inserts get put in the root buffer
A $B^\epsilon$-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer
\textbf{Bε-Trees}

A Bε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer
**Bε-Trees**

A $B\varepsilon$-tree is a search tree 
(like a B-tree)

- Inserts get put in the root buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child’s buffer
Bε-Trees

A Bε-tree is a search tree (like a B-tree)

Inserts get put in the root buffer

When a buffer is full:
1. Pick child receiving most messages
2. Move them to the child's buffer

Key Insight: Each flush applies many small changes
Logical Copy with $B^\varepsilon$-DAGs
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

Every /green/* file is in this subtree
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

Every /green/* file is in this subtree
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

Would like to logically copy by adding this edge

Every /green/* file is in this subtree
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

Would like to logically copy by adding this edge

Every /green/* file is in this subtree
Logical Copy with $B^\varepsilon$-DAGs

PROBLEM:
A lookup for /red/M is going to see /green/M in the subtree

Would like to logically copy by adding this edge

Copy /green to /red

Node covering /green/ subtree

Every /green/* file is in this subtree
Logical Copy with $B^\varepsilon$-DAGs

**Solution:**
Pivots can include a prefix translation.

Would like to logically copy by adding this edge.

**PROBLEM:**
A lookup for /red/M is going to see /green/M in the subtree.

Every /green/* file is in this subtree.

Node covering /green/ subtree.
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

PROBLEM: A lookup for /red/M is going to see /green/M in the subtree

Would like to logically copy by adding this edge

Solution: Pivots can include a prefix translation

Every /green/* file is in this subtree
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Read(/red/M)

Node covering /green/ subtree

Copy /green to /red

Read(/red/M)

Node covering /green/ subtree
Logical Copy with $B^\varepsilon$-DAGs

Copy /green/ to /red/

Read(/red/M)

Node covering /green/ subtree

/read/→/green/
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Read(/red/M)

Node covering /green/ subtree
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

Read(/red/M)

/read/ → /green/

Read(/red/M)
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

Read(/red/M)

A R L M

J H

B S Z

A A B

/read/→/green/
Logical Copy with $\mathbf{B^\varepsilon}$-DAGs

Copy /green to /red

Node covering /green/ subtree

Read(/red/M)

Read(/red/M)

Read(/red/M)

Copy /green to /red

Node covering /green/ subtree

Read(/green/M)
Logical Copy with $B^\varepsilon$-DAGs

Copy /green to /red

Node covering /green/ subtree

Read(/red/M)

Read(/red/M)

Read(/red/→/green/)

Read(/green/M)
$B^\epsilon$-DAGs and Small Writes
Bε-DAGs and Small Writes

Make some small writes to /red/
Bε-DAGs and Small Writes

Make some small writes to /red/
Bε-DAGs and Small Writes

Make some small writes to /red/

/red/ → /green/
Bε-DAGs and Small Writes

Make some small writes to /red/
Bε-DAGs and Small Writes

Make some small writes to /red/
Bε-DAGs and Small Writes

Make some small writes to \(\text{/red/}\)
Bε-DAGs and Small Writes

Make some small writes to /red/
**Bε-DAGs and Small Writes**

Make some small writes to /red/

Now flush with copy-on-write

/red/ → /green/

/red/ → /green/
Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

1. copy
B\(\varepsilon\)-DAGs and Small Writes

1. Make some small writes to /red/

2. Now flush with copy-on-write

1. Copy

\[/\text{red}/ \rightarrow /\text{green}/\]

2. Translate

\[/\text{green}/ \rightarrow /\text{red}/\]
Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

1. copy

2. Translate

/green/ → /red/
B^\varepsilon\text{-DAGs and Small Writes}

Make some small writes to /red/.

Now flush with copy-on-write.

1. Copy /red/ \rightarrow /green/.

2. Translate /green/ \rightarrow /red/.

3. Delete unreachable data.
Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

1. copy

2. Translate /green/ → /red/

3. Delete unreachable data
Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

1. copy /green/ → /red/
2. Translate /green/ → /red/
3. Delete unreachable data
4. Move pivot translation
Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

1. copy

2. Translate /green/ → /red/

3. Delete unreachable data

4. Move pivot translation
Bε-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

1. Copy

2. Translate /green/ → /red/

3. Delete unreachable data

4. Move pivot translation

5. Flush
**Bε-DAGs and Small Writes**

Make some small writes to /red/

Now flush with copy-on-write

1. Copy

2. Translate /green/ → /red/

3. Delete unreachable data

4. Move pivot translation

5. Flush
B-$\varepsilon$-DAGs and Small Writes

Make some small writes to /red/

Now flush with copy-on-write

1. Copy /green/ → /red/
2. Translate /green/ → /red/
3. Delete unreachable data
4. Move pivot translation
5. Flush
Bε-DAGs and Small Writes

Make some small writes to /red/

Broke sharing of one node

Now flush with copy-on-write

Applied multiple small changes

2. Translate /green/ → /red/

3. Delete unreachable data

4. Move pivot translation

1. copy
Bε-DAGs and Small Writes

1. Copy

Broke sharing of one node

2. Translate

3. Delete unreachable data

4. Move pivot translation

5. Flush

Now flush with copy-on-write

Applied multiple small changes

Make some small writes to /red/

Still sharing rest of subtree
Copy-on-Abundant-Write

Broke sharing of one node
Still sharing rest of subtree

Bε-DAGs and Small Writes

Applied multiple small changes

1. Copy

2. Translate /green/ → /red/

3. Delete unreachable data

4. Move pivot translation

5. Flush

Now flush with copy-on-write
Logical Copy with $B^\epsilon$-DAGs

Performance Goals

- Space efficient
- Low latency
- Fast writes
- Fast reads
- Copy-specific
- General file system
Logical Copy with $B^ε$-DAGs

Performance Goals

- Space efficient: ?
- Low latency: ?
- Fast writes: ✔️
- Fast reads: ✔️

Copy-specific

General file system

*Bile Systems Fated for Senescence? Nonsense, Says Science!, Conway et al, FAST 2017*
Logical Copy with $B^\varepsilon$-DAGs

**Performance Goals**

- **Copy-on-Abundant-Write**
  - Space efficient: ✓
  - Low latency: ?
  - Copy-specific

- **$B^\varepsilon$-trees have large nodes**
  - Fast writes: ✓
  - Fast reads: ✓
  - General file system

Reducing Copy Latency in $B^\varepsilon$-DAGs
Reducing Copy Latency in B<sup>ε</sup>-DAGs

Copy /green to /violet
1. Flush messages to node covering /green subtree
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree

Copy /green to /violet
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree

Copy /green to /violet
Reducing Copy Latency in B^ε-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
Reducing Copy Latency in B$^\varepsilon$-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message
Reducing Copy Latency in Bε-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

Copy /green to /violet
Reducing Copy Latency in $B^\epsilon$-DAGs

A GOTO message changes the structure of the tree. It functions as a pivot including prefix translation.

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

Copy /green to /violet
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

Copy /green to /violet

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When a lookup finds a GOTO message, it skips to the target
Reducing Copy Latency in B^ε-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When a lookup finds a GOTO message, it skips to the target

Copy /green to /violet

Read(/violet/H)
Reducing Copy Latency in Bε-DAGs

When a lookup finds a GOTO message, it skips to the target.

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree.

It functions as a pivot including prefix translation.

A GOTO message changes the structure of the tree.
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering `/green` subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When a lookup finds a GOTO message, it skips to the target

Copy `/green` to `/violet`
Reducing Copy Latency in Bε-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When a lookup finds a GOTO message, it skips to the target

Copy /green to /violet

GOTO

Read(/violet/H)
Reducing Copy Latency in Bε-DAGs

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

Copy /green to /violet

When a lookup finds a GOTO message, it skips to the target

The GOTO translates into

Read(/violet/H)

Read(/green/H)
Reducing Copy Latency in Bε-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When a lookup finds a GOTO message, it skips to the target

The GOTO translates into

Read(/violet/H)

Read(/green/H)
Reducing Copy Latency in Bε-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When a lookup finds a GOTO message, it skips to the target

The GOTO translates into

Read(/green/H)

Read(/violet/H)
Reducing Copy Latency in $B^\varepsilon$-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Copy /green to /violet
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Copy /green to /violet
Reducing Copy Latency in $B^\epsilon$-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation
Reducing Copy Latency in $B^\varepsilon$-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation
Reducing Copy Latency in $B_{\varepsilon}$-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation
Reducing Copy Latency in $B^\varepsilon$-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

Copy /green to /violet
Reducing Copy Latency in $\mathcal{B}^\varepsilon$-DAGs

A GOTO message changes the structure of the tree.

1. Flush messages to node covering /green subtree.
2. Insert a GOTO message.

It functions as a pivot including prefix translation.

Copy /green to /violet.
Reducing Copy Latency in $B^\varepsilon$-DAGs

A GOTO message changes the structure of the tree.

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

It functions as a pivot including prefix translation.

Copy /green to /violet

GOTO
Reducing Copy Latency in Bε-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation
Reducing Copy Latency in $B^\varepsilon$-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When the GOTO reaches the height of its target + 1, it becomes a real pivot
Reducing Copy Latency in Bε-DAGs

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree
It functions as a pivot including prefix translation

When the GOTO reaches the height of its target + 1, it becomes a real pivot
Reducing Copy Latency in B^ε-DAGs

Copy /green to /violet

1. Flush messages to node covering /green subtree
2. Insert a GOTO message

A GOTO message changes the structure of the tree

It functions as a pivot including prefix translation

When the GOTO reaches the height of its target + 1, it becomes a real pivot
Logical Copy with $B^\varepsilon$-DAGs

Performance Goals

- Space efficient: ✓
- Low latency: ?
- Fast writes: ✓
- Fast reads: ✓
- $B^\varepsilon$-trees have large nodes: ✓
- Copy-on-Abundant-Write
- Copy-specific
- General file system

Logical Copy with $\mathcal{B}^\varepsilon$-DAGs

Performance Goals

- Space efficient
  ✓

- Low latency
  ✓

Copy-on-Abundant-Write

GOTO messages

Copy-specific

$\mathcal{B}^\varepsilon$-trees have large nodes

Fast writes
  ✓

⇒ Locality*

Fast reads
  ✓

General file system

Evaluation
Space Amplification and Fragmentation

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Space Amplification
additional file system size / added data (1KiB)

Fragmentation
measured by timing a grep over the file system

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk
Space Amplification and Fragmentation

Init: 64 4MiB files with random data. Each round: logically copy all files, then change 16B in each file (1KiB total).

Low space amplification and low fragmentation measured by timing a grep over the file system.

Space Amplification: additional file system size / added data (1KiB)

Fragmentation: measured by timing a grep over the file system

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk
Copy Latency

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk
Copy Latency

Init: 64 4MiB files with random data.
Each round: logically copy all files, then change 16B in each file (1KiB total)

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk

Copy Latency

Lower is Better

Dell Optiplex 790
4-core 3.40 GHz Intel Core i7 CPU
4GiB RAM
500GB 7200 RPM SATA disk
General File System Microbenchmarks

Throughput (MiB/sec)

Sequential Read
- ext4
- BTRFS
- XFS
- ZFS
- BεtrFS 0.4
- BεtrFS 0.5

Sequential Write

Random Read

Random Write

Lower is Better

Higher is Better

BεtrFS 0.5
5.5 seconds
Application Benchmarks

- **Time (seconds)**
  - git clone
  - git diff
  - tar
  - untar

- **Bandwidth (MiB/sec)**
  - rsync -in-place
  - rsync rename

- **Throughput (ops/sec)**
  - IMAP

- Filesystems compared:
  - ext4
  - BTRFS
  - XFS
  - ZFS
  - Btrfs 0.4
  - Btrfs 0.5
  - Btrfs 0.5 (copy)
**Technical Conclusions**

**B$^\varepsilon$-DAGs transform copy-on-write into copy-on-abundant-write**

- Gives strong bounds on space amplification
- Preserves locality, even with small writes
- Exploits B$^\varepsilon$-tree’s batching and flushing

**B$^\varepsilon$-DAGs preserve the fast reads and writes of B$^\varepsilon$-trees**

- Preserve logarithmic tree height and query cost
- Preserve asymptotic costs of inserts and updates

**Copies are fast and cheap**

- GOTO messages enable low-latency DAG mutations
- Total work of copies is $O$(tree height)
Evaluation Conclusions

BetrFS with $B^\varepsilon$-DAGs has strong copy performance in practice

- Low space amplification
- Low latency copying
- Good locality

$B^\varepsilon$-DAGs preserve BetrFS’s performance gains on other operations

- Fast random writes
- Good sequential I/O throughput
- No aging
- Strong across-the-board application benchmark performance
Thank you!

BétrFS Episode V: Attack of the Clones

O’Really? betrfs.org