Causality-Based Versioning

Kiran-Kumar Muniswamy-Reddy and David A. Holland

Harvard School of Engineering and Applied Sciences
Consider this scenario

- I installed a piece of software
  - But.. that broke a few other tools!
- Uninstall not good enough
  - The config files were still corrupt
But which files were modified?

Versioning

Maintains old data to which you can recover
Tracks propagation of data and lets you find which files were modified. Too bad I don’t have those old versions.
Applications of Versioning + Causality

- System Configuration Management
  - Causal data identifies files modified
  - Version data allows you to recover the files modified

- Intrusion Recovery

- IP Compliance

- Reproduce Research Results
Apache split-logfile Vulnerability

- Vulnerability in Apache 1.3
- Vulnerability allows attacker to overwrite any file with a .log extension
Scenario

08AM  open DB.log

09AM  Write tx

10AM

11AM  Write tx

12PM  Detect Corruption
### Open-close

<table>
<thead>
<tr>
<th>Time</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>08AM</td>
<td>Open-close</td>
</tr>
<tr>
<td>09AM</td>
<td>V1;DB.log</td>
</tr>
<tr>
<td>10AM</td>
<td></td>
</tr>
<tr>
<td>12PM</td>
<td>Detect Corruption</td>
</tr>
</tbody>
</table>

- Can only recover to 8 AM
Version-on-every write

08AM

09AM

V1;DB.log
V2;DB.log

Vn+1;DB.log
Vn+2;DB.log

10AM

can recover to 10 AM, but expensive
Goal

Combine versioning and causality, taking advantage of causality information to create versions at just the right time
Contributions

- Two algorithms that create **useful** versions
  - Cycle Avoidance
  - Graph Finesse

- Evaluate efficacy and efficiency of these two algorithms in the context of versioning
Outline

- Introduction
- Background on PASS
- Versioning Algorithms
- Implementation
- Evaluation
- Conclusion
PASS Architecture: P reads A

User process P

Interceptor

Observer

Analyzer

Distributor

Lasagna

Waldo

Kernel

VFS Layer

Syscall Layer

filters events

‘P→A’

version?

cache ‘P→A’

log
PASS Architecture: P writes B

User process P

Interceptor

Observer

Analyzer

Distributor

Lasagna

Waldo

 KERNEL

VFS Layer

Syscall Layer

cache ‘P→A’

Version?

‘B→P’

generates record ‘B→P’

P→A

B→P

log

USER

2/25/2009

Causality-Based
Outline

- Introduction
- Background on PASS
- Versioning Algorithms
- Implementation
- Evaluation
- Conclusion
Intuition for new algorithms

- The creation of a cycle is an indicator that a version created at that instant could be useful later.
- Cycles are violations of causality:
  - Implies that past depends on future!
Open-Close Versioning

1. On the last close of a file, issue a "freeze" operation
   - Freeze declares end of a version

2. The next open and write triggers a new version
## Example scenario

<table>
<thead>
<tr>
<th>Time</th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pread A</td>
<td>Qread A</td>
<td></td>
</tr>
<tr>
<td>Pread B</td>
<td>Qread B</td>
<td></td>
</tr>
<tr>
<td>Pwrite B</td>
<td>Qwrite B</td>
<td></td>
</tr>
<tr>
<td>Pwrite A</td>
<td>Qwrite A</td>
<td></td>
</tr>
<tr>
<td>Pread A</td>
<td>Qread B</td>
<td></td>
</tr>
</tbody>
</table>

Each read/write is enclosed by an open and close.
### Open-Close

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>read A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read B</td>
</tr>
<tr>
<td>write B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write A</td>
</tr>
<tr>
<td>read A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read B</td>
</tr>
</tbody>
</table>
Open-Close

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>read A</td>
<td>read B</td>
</tr>
<tr>
<td>write B</td>
<td>write A</td>
</tr>
<tr>
<td>read A</td>
<td>read B</td>
</tr>
</tbody>
</table>

A1

B1
Open-Close

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>read A</td>
<td>read B</td>
</tr>
<tr>
<td>B2</td>
<td>write B</td>
<td>write A</td>
</tr>
<tr>
<td>B1</td>
<td>read A</td>
<td>read B</td>
</tr>
</tbody>
</table>
Open-Close

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>read A</td>
<td>read B</td>
</tr>
<tr>
<td></td>
<td>write B</td>
<td>write A</td>
</tr>
<tr>
<td></td>
<td>read A</td>
<td>read B</td>
</tr>
</tbody>
</table>
Open-Close

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>read A</td>
<td>read A</td>
</tr>
<tr>
<td></td>
<td>read B</td>
<td>read B</td>
</tr>
<tr>
<td></td>
<td>write B</td>
<td>write A</td>
</tr>
</tbody>
</table>

Diagram:

- **P** connected to **A1** and **B2**
- **Q** connected to **A2** and **B1**
- Arrows indicating read and write operations
Open-Close allows cycles to happen.

Violates Causality
Version-on-every write

- Pros:
  - Preserves causality: there are no cycles
    - Every read creates a new version of the process
    - Every write creates a new version of the file
  - There are no duplicates either

- Disadvantage: most versions are unnecessary
Cycle Avoidance Algorithm

- Preserves Causality by avoiding cycles
- Uses local per-object information to make decisions
- Similar to the timestamp ordering in databases
- Intuition:
  Freeze an object when we add a dependency that does not previously exist, i.e., new causality
Cycle Avoidance Example

- On receiving record A1 → B2
  - If no B in A’s history, then freeze A
  - Else if B in A’s history, then
    - If A’s history has B2, discard record (duplicate)
    - If A’s history has B3 (version > 2), discard record
    - If A’s history has B1 (version < 2), freeze A
Cycle Avoidance

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>read A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>read B</td>
</tr>
<tr>
<td></td>
<td>write B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read B</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

P1 → A1 → P2

A1 → A1
Cycle Avoidance

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>read A</td>
<td>read B</td>
</tr>
<tr>
<td>write B</td>
<td>write A</td>
</tr>
<tr>
<td>read A</td>
<td>read B</td>
</tr>
</tbody>
</table>

Diagram:

- P2 -> A1
- P3 -> B2
- Q2 -> A2
- B1

A1 reads A
B2 reads B
A2 writes A
B1 reads B
Cycle Avoidance

Cycle-Avoidance prevents cycles, but creates more versions.
Graph Finesse Algorithm

- Uses Global knowledge
- Intuition:
  - Check every new record against a global dependency graph.
  - If it forms a cycle, just freeze that one node
- Subsumes open-close algorithm
Graph Finesse Example

- **On receiving record A1 → B2**
  - If B2 is already in A’s history, discard record
  - Else check for a path from B2 → A1
    - If yes, this a cycle, freeze A1 and change the record to A2→B2
    - If no cycle, add A1 → B2 to the graph
Graph Finesse

P1

A1

B2

Q1

A2

B1

P
read A
read B
write B
write A
read A

Q
read B
Cycle Avoidance

Graph Finesse prevents cycles.

But creates fewer versions than Cycle Avoidance
<table>
<thead>
<tr>
<th>Cycle Avoidance</th>
<th>Graph Finesse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses Local state</td>
<td>Uses Global state</td>
</tr>
<tr>
<td>Creates a few unnecessary versions</td>
<td>Creates fewer versions</td>
</tr>
<tr>
<td>Has lower runtime overhead</td>
<td>Can have high runtime overheads</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Background on PASS
- Versioning Algorithms
- Implementation
- Evaluation
- Conclusion
Implementation

- Implemented on Linux 2.6.23.17
- Lasagna is a stackable file system derived from eCryptfs
- Versioning file system
  - Redo log that keeps track of file versioning (deltas)
  - Redo log for directory modifications (deltas)
Outline

- Introduction
- Background on PASS
- Versioning Algorithms
- Implementation
- Evaluation
- Conclusion
Evaluation Goals

- What are the run-time overheads a user might see?
- What are the space overheads?
- How do the algorithms compare during recovery?
Test platform

- Linux 2.6.23.17
- 3Ghz Pentium 4
- 512MB of RAM
- 80GB 7200 RPM IDE Disk
- All results are averages of 5 runs
  - Less than 5% Std. Dev.
Modes

- Without causal data
  - Ext2: Baseline (Lasagna was stacked on Ext2)
  - VER: plain versioning (open-close)

- With causal data
  - OC: open-close
  - CA: Cycle-Avoidance
  - GF: Graph-Finesse
  - ALL: Version-on-every write
Linux Compile: Elapsed Time

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>EXT2</th>
<th>VER</th>
<th>OC</th>
<th>CA</th>
<th>GF</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait</td>
<td></td>
<td>11.9%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Linux Compile: Elapsed Time

<table>
<thead>
<tr>
<th>File System</th>
<th>Wait (%)</th>
<th>User (%)</th>
<th>System (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT2</td>
<td>11.9</td>
<td>17.1</td>
<td>18.3</td>
</tr>
<tr>
<td>VER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>21.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time (s)
Linux Compile: Elapsed Time
Linux Compile: Space Overheads

- Ext2: 2.9%
- Ver: 2.9%
- Oc
- Ca
- Gf
- All

Space (GB)
Linux Compile: Space Overheads

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Space (GB)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT2</td>
<td>2.9%</td>
<td></td>
</tr>
<tr>
<td>VER</td>
<td>15.8%</td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td>17.6%</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>15.8%</td>
<td></td>
</tr>
<tr>
<td>GF</td>
<td>15.8%</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Linux Compile: Space Overheads

<table>
<thead>
<tr>
<th>File System</th>
<th>Space (GB)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT2</td>
<td>1.5</td>
<td>2.9%</td>
</tr>
<tr>
<td>VER</td>
<td>1.7</td>
<td>15.8%</td>
</tr>
<tr>
<td>OC</td>
<td>2.0</td>
<td>17.6%</td>
</tr>
<tr>
<td>CA</td>
<td>1.5</td>
<td>15.8%</td>
</tr>
<tr>
<td>GF</td>
<td>3.0</td>
<td>121.6%</td>
</tr>
<tr>
<td>ALL</td>
<td>3.0</td>
<td></td>
</tr>
</tbody>
</table>
Mercurial Activity: Elapsed Time

- Wait
- User
- System

Time(s):
- EXT2
- VER
- OC
- CA
- GF
- ALL

25.9%
Mercurial Activity: Elapsed Time

25.9%  28.8%  27.9%

Time(s)

Wait  User  System

89.6%
Mercurial Activity: Space Overheads

26.6%
Mercurial Activity: Space Overheads

Space (GB)

EXT2  VER  OC  CA  GF  ALL

26.6%  31.6%  30.2%  31.9%

Causality-Based Versioning - FAST'09
Mercurial Activity: Space Overheads

<table>
<thead>
<tr>
<th></th>
<th>EXT2</th>
<th>VER</th>
<th>OC</th>
<th>CA</th>
<th>GF</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>0.80</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.40</td>
</tr>
<tr>
<td>Percent</td>
<td>26.6%</td>
<td>31.6%</td>
<td>30.2%</td>
<td>31.9%</td>
<td>53.7%</td>
<td></td>
</tr>
</tbody>
</table>
Recovery Benchmarks

- How the algorithms perform in the scenario where open close is not sufficient
- Microbenchmark
  - Models the apache split-log scenario
Recovery MicroBenchmark

P

fork

read

write

pipe

write

Q

read

fork
## Recovery Microbenchmark: Space Util.

<table>
<thead>
<tr>
<th></th>
<th>Causal Data</th>
<th>Version Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC</td>
<td>60KB</td>
<td>12KB</td>
</tr>
<tr>
<td>CA</td>
<td>176KB</td>
<td>470.5MB</td>
</tr>
<tr>
<td>GF</td>
<td>184KB</td>
<td>470.5MB</td>
</tr>
<tr>
<td>ALL</td>
<td>76.9MB</td>
<td>1.97GB</td>
</tr>
</tbody>
</table>
Recovery Times

![Bar chart showing recovery times for Rollback 1, Rollback 5, and Rollback 9 for CA and GF.]
Recovery Times

- **Rollback 1**: 9.3x
- **Rollback 5**: 17.9x
- **RollBack 9**: 25.1x

Legend:
- **CA**
- **GF**
- **ALL**

Recovery Time(s)
Conclusions

- Combining Versioning and Causality enables novel functionality
- New algorithms for Causal Versioning
  - Cycle Avoidance
    - Comparable to open-close
    - May create more versions
  - Graph Finesse
    - Provides greater control on versioning
    - Can be inefficient at times
Questions?

Contact:

pass@eecs.harvard.edu
www.eecs.harvard.edu/syrah/pass