Improving Server Applications with System Transactions

Sangman Kim, Michael Z. Lee, Alan M. Dunn, Owen S. Hofmann, Xuan Wang, Emmett Witchel, Donald E. Porter
Poor OS API Support for Concurrency

Parallelism

Fine-grained locking
- Bug-prone, hard to maintain
- OS provides poor support

Maintainability

Coarse-grained locking
- Reduced resource utilization
System Transaction Improves OS API Concurrency

- Server Applications working with OS API

Parallelism vs. Maintainability

System Transaction

Server Applications working with OS API
Improving System Transactions

- TxOS provides operating system transaction
  [Porter et al., SOSP 2009]
  - Transaction for OS objects (e.g., files, pipes)
  Middleware state sharing with multithreading

- TxOS system calls

```plaintext
Application

Middleware state sharing

JVM

TxOS
```
Improving System Transactions

- TxOS provides operating system transaction
  [Porter et al., SOSP 2009]
  - Transaction for OS objects (e.g., files, pipes)

Synchronization in legacy code

- TxOS system calls

Application

Middleware state sharing
Synchronization primitives
Improving System Transactions

- **TxOS** provides operating system transaction
  
  [Porter et al., SOSP 2009]

  **Up to 88% throughput improvement**

  **At most 40 application line changes**

- **TxOS system calls**

  - **Application**
  - **JVM**
  - **TxOS+**

  Middleware state sharing
  Synchronization primitives

  **TxOS+: pause/resume, commit ordering, and more**
Outline

- Background: system transaction
- System transactions in action
- Challenges for rewriting applications
- Implementation and evaluation
Background: System Transaction

- Transaction Interface and semantics
  - System calls: xbegin(), xend(), xabort()

- ACID semantics
  - Atomic – all or nothing
  - Consistent – one consistent state to another
  - Isolated – updates as if only one concurrent transaction
  - Durable – committed transactions on disk

- Optimistic concurrency control

- Fix synchronization issues with OS APIs
Lazy versioning: speculative copy for data

TxOS requires no special hardware
Outline

- Background: system transaction
- System transactions in action
- Challenges for rewriting applications
- Implementation and evaluation
Applications Parallelized with OS Transactions

- Parallelizing applications that synchronize on OS state
  - Example 1: State-machine replication
    - Constraint: Deterministic state update
  - Example 2: IMAP Email Server
    - Constraint: Consistent file system operations
Example 1: Parallelizing State-machine Replication

- Core component of fault tolerant services
  - e.g., Chubby, Zookeeper, Autopilot

- Replicas execute the same sequence of operations
  - Often single-threaded to avoid non-determinism

- Ordered transaction
  - Makes parallel OS state updates deterministic
  - Applications determine commit order of transactions
Example 2: Parallelizing IMAP Email Servers

- Everyone has concurrent email clients
  - Desktop, laptop, tablets, phones, ....
  - Need concurrent access to stored emails

- Brief history of email storage formats
  - mbox: single file, file locking
  - Lockless Maildir
  - Dovecot Maildir: return of file locking
mbox: Database Without Parallelism

- **mbox**
  - **Single file** mailbox of email messages

```
~/.mbox
```

- **Synchronization with file-locking**
  - One of `fcntl()`, `flock()`, lock file (.mbox.lock)
  - Very coarse-grained locking

From MAILER-DAEMON Wed Apr 11 09:32:28 2012
From: Sangman Kim <sangmank@cs.utexas.edu>
To: EuroSys 2012 audience
Subject: mbox needs file lock. Maildir hides message.

.....

From MAILER-DAEMON Wed Apr 11 09:34:51 2012
From: Sangman Kim <sangmank@cs.utexas.edu>
To: EuroSys 2012 audience
Subject: System transactions good, file locks bad!

.....
**Maildir: Parallelism Through Lockless Design**

- **Maildir: Lockless alternative to mbox**
  - Directories of message files
  - Each file contains a message
  - Directory access with no synchronization (originally)

- **Message filenames contain flags**

```
Maildir/cur
```

```
00000000.00201.host:2,T  Trashed
00001000.00305.host:2,R  Replied
00002000.02619.host:2,T  Trashed
00010000.08919.host:2,S  Seen
00015000.10019.host:2,S  Seen
```
Messages Hidden with Lockless Maildir

**PROCESS 1 (LISTING)**

```python
while (f = readdir("Maildir/cur")):
    print f.name
```

**PROCESS 2 (MARKING)**

```python
if (access("043:2,S")):
    rename("043:2,S", "043:2,R")
```

"Maildir/cur" directory

```
018:2,S  021:2,S  043:2,S  052:2,S  061:2,S
  Seen   Seen   Seen   Seen   Seen
```
Messages Hidden with Lockless Maildir

**PROCESS 1 (LISTING)**

```python
while (f = readdir("Maildir/cur")):
    print f.name
```

**PROCESS 2 (MARKING)**

```python
if (access("043:2,S")):
    rename("043:2,S", "043:2,R")
```

```
<table>
<thead>
<tr>
<th>043:2,R</th>
<th>018:2,S</th>
<th>021:2,S</th>
<th>043:2,S</th>
<th>052:2,S</th>
<th>061:2,S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replied</td>
<td>Seen</td>
<td>Seen</td>
<td>Seen</td>
<td>Seen</td>
<td>Seen</td>
</tr>
</tbody>
</table>

```

Process 1 Result
- 018:2,S
- 021:2,S
- 052:2,S
- 061:2,S

**Message missing!**
Maildir synchronization

- Lockless

  "certain anomalous situations may result"

  – Courier IMAP manpage

- File locks
  - Per-directory coarse-grained locking
  - Complexity of Maildir, performance of mbox

- System transactions
**Maildir Parallelized with System Transaction**

**PROCESS 1 (MARKING)**

```c
xbegin()
if (access("XXX:2,S"):)
    rename("XXX:2,S",
          "XXX:2,R")
xbend()
```

**PROCESS 2 (MESSAGE LISTING)**

```c
xbegin()
while (f = readdir("Maildir/cur"):)
    print f.name
xbend()
```

**Consistent directory accesses with better parallelism**
Outline

- Background: system transaction
- System transactions in action
- Challenges for rewriting applications
- Implementation and evaluation
Challenges of Rewriting Applications

1. Middleware state sharing
2. Deterministic parallel update for system state
3. Composing with other synchronization primitives
Problem with memory management
  - Multiple threads share the same heap

```
In Transaction
Thread 1
  xbegin();
  p1 = malloc();
  xabort();
```

```
Thread 2
  p2 = malloc();
  *p2 = 1;
```
Problem with memory management
- Multiple threads share the same heap

```
In Transaction
Thread 1
xbegin();
p1 = malloc();
-> xabort();

Thread 2
-> p2 = malloc();
-> *p2 = 1;
FAULT!
```

Certain middleware actions should not roll back
Two Types of Actions on Middleware State

**USER-INITIATED ACTION**
- User changes system state
  - Most file accesses
  - Most synchronization

**MIDDLEWARE-INITIATED**
- System state changed as side effect of user action
  - malloc() memory mapping
  - Java garbage collection
  - Dynamic linking
- Middleware state shared among user threads
  - Can’t just roll back!
Handling Middleware-Initiated Actions

- Transaction pause/resume
  - Expose state changes by **middleware-initiated actions** to other threads

- Additional system calls
  - `xpause()`, `xresume()`

- Limited complexity increase
  - We used pause/resume 8 times in glibc, 4 times in JVM
  - Only used in application for debugging
SysTransaction.begin();

files = dir.list();

SysTransaction.end();

Java code

JVM Execution

xbegin();

files = dir.list();

xpause()

VM operations
(garbage collection)

xresume()

xend();
Other Challenges for Maturing TxOS

- 17,000 lines of kernel changes
  - Transactionalizing file descriptor table
  - Handling page lock for disk I/O
  - Memory protection
  - Optimization with directory caching
  - Reorganizing data structure
  - and more

- Details in the paper
Background: system transaction

System transactions in action

Challenges for rewriting applications

Implementation and evaluation
Application 1: Parallelized BFT Application

- Implemented in UpRight BFT library

- Fault tolerant routing backend
  - Graph stored in a file
  - Compute shortest path
  - Edge add/remove

- Ordered transactions for deterministic update
## Minimal Application Code Change

<table>
<thead>
<tr>
<th>Component</th>
<th>Total LOC</th>
<th>Changed LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing application</td>
<td>1,006</td>
<td>18 (1.8%)</td>
</tr>
<tr>
<td>Upright Library</td>
<td>22,767</td>
<td>174 (0.7%)</td>
</tr>
<tr>
<td>JVM</td>
<td>496,305</td>
<td>384 (0.0008%)</td>
</tr>
<tr>
<td>glibc</td>
<td>1,027,399</td>
<td>826 (0.0008%)</td>
</tr>
</tbody>
</table>
Deterministic State Update with Better Throughput

Dense graph: 88% tput ↑
Sparse graph: 11% tput ↑

Work to add/delete edges small compared to scheduling overhead

Throughput (req/s)

BFT graph server

Write ratio (%)
Application 2: Dovecot Maildir access

- Dovecot mail server
  - Uses directory lock files for maildir accesses

- Locking is replaced with system transactions
  - Changed LoC: 40 out of 138,723

- Benchmark: Parallel IMAP clients
  - Each client executes operations on a random message
    - Read: message read
    - Write: message creation/deletion
    - 1500 messages total
Mailbox Consistency with Better Throughput

- Dovecot benchmark with 4 clients

Better block scheduling enhances write performance
Conclusion:
OS Transactions Improve Server Performance

- System transactions parallelize tricky server applications
  - Parallel Dovecot maildir operations
  - Parallel BFT state update

- System transaction improves throughput with few application changes
  - Up to 88% throughput improvement
  - At most 40 changed lines of application code