Ad Hoc Synchronization Considered Harmful

Weiwei Xiong, Soyoen Park, Jiaqi Zhang, Yuanyuan Zhou and Zhiqiang Ma

UC San Diego    University of Illinois    Intel
Synchronization is Important

• Concurrent programs are pervasive
• Synchronization in programs
  – Ensure correctness of execution
  – Mutual exclusion
  – Conditional wait
Common Synchronization Primitives

handler handle_slave_sql()
{
    pthread_mutex_lock(&thread_count);
    threads.append(thd);
    pthread_mutex_unlock(&thread_count);
}
/* MySQL */

**pthread lock**

apr_status_t apr_reslist_acquire()
{
    apr_thread_mutex_lock(reslist->lock);
    res = pop_resource(reslist);
    apr_thread_mutex_unlock(reslist->lock);
}
/* Apache */

**customized lock**

where is the sync?
Common Synchronization Primitives

```c
handler handle_slave_sql()
{
    pthread_mutex_lock(&thread_count);
    threads.append(thd);
    pthread_mutex_unlock(&thread_count);
}
```

```c
apr_status_t apr_reslist_acquire()
{
    res = pop_resource(reslist);
    apr_thread_mutex_lock(reslist->lock);
    res = pop_resource(reslist);
    apr_thread_mutex_unlock(reslist->lock);
}
/* Apache */
```

**customized lock**

*MySQL*

*Apache*

where is the sync?
Hard-to-recognize Synchronization

Ad hoc sync

Sync variable

Is it doing sync?

for (deleted=0; ;) {
    ...
    if (dbmfp->ref == 1) {
        if (F_ISSET(dbmfp, OPEN_CALLED))
            TAILQ_REMOVE(&dbmp->dbmfq, ...);
        deleted = 1;
    }
    ...
    if (deleted)
        break;
    __os_sleep(dbenv, 1, 0);
}

/* OpenLDAP */
Hard-to-recognize Synchronization

for (deleted=0; ;) {
    ...
    if (dbmfp->ref == 1) {
        if (F_ISSET(dbmfp, OPEN_CALLED))
            TAILQ_REMOVE(&dbmp->dbmfq, ...);
        deleted = 1;
    }
    ...
    if (deleted)
        break;
    __os_sleep(dbenv, 1, 0);
} /* OpenLDAP */
Hard-to-recognize Synchronization

```c
loop:
    if (shutdown_state > 0)
        goto background_loop;
    ...
    if (shutdown_state == EXIT)
        os_thread_exit(NULL)
        goto loop;
else
    goto background_loop;

background_loop:
/* background operations */
...
Hard-to-recognize Synchronization

```c
loop:
    if(shutdown_state > 0)
        goto background_loop;
    ...
    if(shutdown_state == 0)
        os_thread_exit(NULL);
    goto loop;
```

```
/*
   background operations */
```

```
/*
   MySQL
*/
```

if(shutdown_state > 0)
    goto background_loop;
else
    goto loop;
```
What are the Consequences?

• Introducing bugs or performance issues
  – up to 67% of ad hoc syncs introduced bugs

• Making program analysis more difficult
  – hard-to-detect deadlocks
  – introducing false positives to data race checker
  – confusions to sync performance profiling

• Problematic interactions with compiler and memory consistency model
What are the Consequences?

• Introducing bugs or performance issues
  – up to 67% of ad hoc syncs introduced bugs

• Making program analysis more difficult
  – hard-to-detect deadlocks
  – introducing false positives to data race checker
  – confusions to sync performance profiling

• Problematic interactions with compiler and memory consistency model

More examples later
Our Contribution

• Quantitative evidence to show ad hoc syncs are harmful
• SyncFinder: a tool that automatically identifies and annotates ad hoc syncs
  – helps to detect new deadlocks and bad practices
  – helps to reduce false positive of race detectors
Outline

1. Motivation
2. Ad Hoc Sync Study
3. SyncFinder: Auto-Annotation
4. Evaluation Results
5. Conclusions
Data Set and Methodology

- Different types of concurrent programs
  - servers
  - desktop apps
  - scientific programs
- Manually examine every program
- Two persons each spent 3 months

<table>
<thead>
<tr>
<th>Server</th>
<th>Apps.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Apache</td>
<td>Web server</td>
</tr>
<tr>
<td></td>
<td>MySQL</td>
<td>Database server</td>
</tr>
<tr>
<td></td>
<td>OpenLDAP</td>
<td>LDAP server</td>
</tr>
<tr>
<td></td>
<td>Cherokee</td>
<td>Web server</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desktop</th>
<th>Apps.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mozilla JS</td>
<td>JS engine</td>
</tr>
<tr>
<td></td>
<td>PBZip2</td>
<td>Parallel bzip2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scientific</th>
<th>Apps.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Radiosity</td>
<td>SPLASH-2</td>
</tr>
<tr>
<td></td>
<td>Barnes</td>
<td>SPLASH-2</td>
</tr>
<tr>
<td></td>
<td>Water</td>
<td>SPLASH-2</td>
</tr>
<tr>
<td></td>
<td>OCEan</td>
<td>SPLASH-2</td>
</tr>
<tr>
<td></td>
<td>FFT</td>
<td>SPLASH-2</td>
</tr>
</tbody>
</table>
# Every Studied Program Has Ad Hoc Syncs

<table>
<thead>
<tr>
<th>Apps.</th>
<th>Description</th>
<th>Ad hoc sync loops</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache</td>
<td>Web server</td>
<td>33</td>
</tr>
<tr>
<td>MySQL</td>
<td>Database server</td>
<td>83</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>LDAP server</td>
<td>15</td>
</tr>
<tr>
<td>Cherokee</td>
<td>Web server</td>
<td>6</td>
</tr>
<tr>
<td><strong>Desktop</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla JS</td>
<td>JS engine</td>
<td>17</td>
</tr>
<tr>
<td>PBZip2</td>
<td>Parallel bzip2</td>
<td>7</td>
</tr>
<tr>
<td>Transmission</td>
<td>BitTorrent client</td>
<td>13</td>
</tr>
<tr>
<td><strong>Scientific</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radiosity</td>
<td>SPLASH-2</td>
<td>12</td>
</tr>
<tr>
<td>Barnes</td>
<td>SPLASH-2</td>
<td>7</td>
</tr>
<tr>
<td>Water</td>
<td>SPLASH-2</td>
<td>9</td>
</tr>
<tr>
<td>Ocean</td>
<td>SPLASH-2</td>
<td>20</td>
</tr>
<tr>
<td>FFT</td>
<td>SPLASH-2</td>
<td>7</td>
</tr>
</tbody>
</table>
Ad Hoc Syncs are Error-prone

• Percentage of buggy ad hoc syncs

<table>
<thead>
<tr>
<th>Apps.</th>
<th># ad hoc sync</th>
<th># buggy sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>33</td>
<td>7 (22%)</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>15</td>
<td>10 (67%)</td>
</tr>
<tr>
<td>Cherokee</td>
<td>6</td>
<td>3 (50%)</td>
</tr>
<tr>
<td>Mozilla JS</td>
<td>17</td>
<td>5 (30%)</td>
</tr>
<tr>
<td>Transmission</td>
<td>13</td>
<td>8 (62%)</td>
</tr>
</tbody>
</table>
Hard-to-detect Deadlock

**Thread 1**

S1  JS_ACQUIRE_LOCK(rt->setSlotLock);
    ...
S2  while( rt->gcLevel > 0 ) {...}
S3  JS_RELEASE_LOCK(rt->setSlotLock);

**Thread 2**

S1  rt->gcLevel = 1;
    ...
S2  while(rt->requestCount > 0) {...}
    ...
S3  rt->gcLevel = 0;

**Thread 3**

S1  rt->requestCount++;
    ...
S2  JS_ACQUIRE_LOCK(rt->setSlotLock);
    ...
S3  rt->requestCount--;

Hard-to-detect Deadlock
Hard-to-detect Deadlock

holding: rt->setSlotLock
waiting: rt->gcLevel

Thread 1
S1  JS_ACQUIRE_LOCK(rt->setSlotLock);
    ...
S2  while( rt->gcLevel > 0 ) {...}
S3  JS_RELEASE_LOCK(rt->setSlotLock);

waiting: rt->setSlotLock

Thread 3
S1  rt->requestCount++;
    ...
S2  ACQUIRE_LOCK(rt->setSlotLock);
    ...
S3  rt->requestCount--;

waiting: rt->requestCount

Thread 2
S1  rt->gcLevel = 1;
    ...
S2  while(rt->requestCount > 0) {...}
    ...
S3  rt->gcLevel = 0;

waiting: rt->requestCount
Performance Issues

/* get tuple id of a table */

for(;;) {
    if (m_skip_auto_increment &&
        readAutoIncrementValue(...)
        || getAutoIncrementValue(...)) {
        if (--retries && ...) {
            my_sleep(retry_sleep);
            continue;
        }
    } break;
}

A performance issue from MySQL
Impact to Bug Detection Tools

• Confusing race detectors
  – Benign data race on sync variable

```
Thread 1
#define LAST_PHASE 1
loop:
if(state < LAST_PHASE)
goto Loop;
Thread 2
#define EXIT_THREADS 3
state = EXIT_THREADS;
/* MySQL */
```

– False data race on ordered variable accesses

```
Worker
S1  q_info->pools = new_recycle;
S2  atomic_inc(&q_info->idlers);

Listener
S3  while( q_info->idlers == 0) {...}
S4  first_pool = q_info->pools;
/* Apache */
```
Ad Hoc Syncs are Diverse

**sync variables**

**single cond (sc)**

```c
loop:
  if(state < LAST_PHASE )
  goto loop;
```

```c
while (crc_table_empty);
```

```c
while (QueryStatus(.,&status) {
  if(status == PENDING)
    sleep(10000);
  else break;
}
```

**code style**

**multiple cond (mc)**

```c
for(; i < 1000 && ! finished; i ++) {
  if(global->pbar_count >= 8)
    finished = 1;
}
```

```c
while(1) {
  int oldcount = (global->barrier).count;
  ... 
  if(updatedcount == oldcount) break;
}
```

**#conditions**

**dir**

**func**

**control(cf)**

**data(df)**
## Ad Hoc Syncs are Diverse

<table>
<thead>
<tr>
<th>Apps</th>
<th>Total ad hoc</th>
<th>Single exit cond.</th>
<th>Multiple exit cond.</th>
<th>Total func</th>
<th>async</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>sc-dir</td>
<td>sc-df</td>
<td>sc-cf</td>
<td>sc-func</td>
</tr>
<tr>
<td>Apache</td>
<td>33</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MySQL</td>
<td>83</td>
<td>23</td>
<td>5</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>15</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Cherokee</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Mozilla JS</td>
<td>17</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>PBZip2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Transmission</td>
<td>13</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Radiosity</td>
<td>12</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Barnes</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Water</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OCeans</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FFT</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Ad Hoc Sync Study
3. SyncFinder: Auto-Annotation
4. Evaluation Results
5. Conclusions
Ad Hoc Synchronization

for(i; i < 1000 && ! finished; i++) {
    if(global->pbar_count >= 8)
        finished = 1;
}

global->pbar_count +=

global->pbar_count = 0;

Waiting side
- Sync loop: The loop body
- Exit condition
  {! finished, i < 1000}
- Exit condition variable
  { finished, i}
- Sync variable
  global->pbar_count

Setting side
- Sync write: The write instructions that will release the ad hoc sync loop
  global->pbar_count ++;
Ad Hoc Synchronization

for(i; i < 1000 && ! finished; i ++) {
    if(global->pbar_count >= 8)
        finished = 1;
}

global->pbar_count += ;
global->pbar_count = 0;

Waiting side
• Sync loop: The loop body
• Exit condition
  {! finished, i < 1000}
• Exit condition variable
  { finished, i}
• Sync variable
  global->pbar_count

Setting side
• Sync write: The write instructions that will release the ad hoc sync loop
  global->pbar_count += ;

if(global->pbar_count >= 8) <- sync pair -> global->pbar_count += ;
Flowchart of SyncFinder

Source code

Loop detection

Exit condition extraction (break, ret, exit, etc.)

Exit dependent variable (EDV) detection

Pruning

Reporting and annotation

```c
int finished = 0;
for(i = 0; i < 1000 && !finished; i++) {
    if(global->pbar_count >= 8)
        finished = 1;
}

{ finished, i, 1000}

{ 1, i, 1000, global->pbar_count, 8}

{ global->pbar_count }

sync loop ( taskman.c:1294 )
```
Sync Loop Pruning

• Our observation
  – Sync conditions must depend on remote threads
    • i.e., communicating using shared variables
  – Sync variables should be loop invariants

Normal Computation

for (i = 0; i < nlights; i++) {...}

Ad Hoc Sync Loop

while (global->gsense == lsense);
Sync Pair Identification

Ad hoc sync loops

Sync information collection

False sync pair pruning

```
int finished = 0;
for(i = 0; i < 1000 && !finished; i++) {
  if(global->pbar_count >= 8)
    finished = 1;
}
```

Read
```
global->pbar_count
```

Write
```
global->pbar_count = 0
global->pbar_count ++
```

```
global->pbar_count <-> global->pbar_count = 0
global->pbar_count <-> global->pbar_count ++
```
Sync Pair Identification

Ad hoc sync loops

Sync information collection

False sync pair pruning

int finished = 0;
for(i = 0; i < 1000 && !finished; i ++) {
    if(global->pbar_count >= 8) {
        finished = 1;
    }
}

Read
global->pbar_count

Write
global->pbar_count = 0
global->pbar_count ++

global->pbar_count <-> global->pbar_count = 0
global->pbar_count <-> global->pbar_count ++

R,taskman.c:1294 <-> W,taskman.c:1233
Report and Annotation

• SyncFinder report
  – Line numbers of sync reads and writes
  – Sync loop context: entry/exit points
• Automatic annotations
  – SF_Loop_Begin/End(&loopID)
  – SF.Sync_Read_Begin/End(&loopID, &sync_var)
  – SF.Sync_Write_Begin/End(&loopID, &sync_var)
Outline

1. Motivation
2. Ad Hoc Sync Study
3. SyncFinder: Auto-Annotation
4. Evaluation Results
5. Conclusions
# SyncFinder’s Overall Result

<table>
<thead>
<tr>
<th>Apps.</th>
<th>Total loops</th>
<th>True ad hoc syncs</th>
<th>Missed ad hoc syncs</th>
<th>False positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1462</td>
<td>15</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MySQL</td>
<td>4265</td>
<td>42</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>2044</td>
<td>14</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Cherokee</td>
<td>748</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla JS</td>
<td>848</td>
<td>11</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>PBZip2</td>
<td>45</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Transmission</td>
<td>1114</td>
<td>12</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Radiosity</td>
<td>80</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Barnes</td>
<td>88</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Water</td>
<td>84</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ocean</td>
<td>339</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FFT</td>
<td>57</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## SyncFinder’s Overall Result

<table>
<thead>
<tr>
<th>Apps.</th>
<th>Total loops</th>
<th>True ad hoc syncs</th>
<th>Missed ad hoc syncs</th>
<th>False positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1462</td>
<td>15</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MySQL</td>
<td>4265</td>
<td>42</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>2044</td>
<td>14</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Cherokee</td>
<td>748</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla JS</td>
<td>848</td>
<td>1</td>
<td>1</td>
<td>average 96%</td>
</tr>
<tr>
<td>PBZip2</td>
<td>45</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Transmission</td>
<td>1114</td>
<td></td>
<td>1</td>
<td>average 6%</td>
</tr>
<tr>
<td>Radiosity</td>
<td>80</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Barnes</td>
<td>88</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Water</td>
<td>84</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ocean</td>
<td>339</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FFT</td>
<td>57</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Result on Additional Programs

<table>
<thead>
<tr>
<th>Apps</th>
<th>Total loops</th>
<th>True ad hoc syncs</th>
<th>False positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOLServer</td>
<td>496</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Nginx</td>
<td>705</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>BerkeleyDB</td>
<td>1006</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>BIND9</td>
<td>1372</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>HandBrake</td>
<td>551</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>p7zip</td>
<td>1594</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>wxDFast</td>
<td>154</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Cholesky</td>
<td>362</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>RayTracer</td>
<td>144</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>FMM</td>
<td>108</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Volrend</td>
<td>77</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>LU</td>
<td>38</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Radix</td>
<td>52</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

10/13/10
Use cases: Bug Detection

• A tool to detect bad practices

LOCK
while(...);
UNLOCK

<table>
<thead>
<tr>
<th>Apps.</th>
<th>Deadlock (New)</th>
<th>Bad practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1(0)</td>
<td>1</td>
</tr>
<tr>
<td>MySQL</td>
<td>2(2)</td>
<td>13</td>
</tr>
<tr>
<td>Mozilla</td>
<td>2(0)</td>
<td>2</td>
</tr>
</tbody>
</table>

• Extended race detector in Valgrind

<table>
<thead>
<tr>
<th>Apps.</th>
<th>Original Valgrind</th>
<th>Extended Valgrind</th>
<th>% Pruned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>30</td>
<td>17</td>
<td>43%</td>
</tr>
<tr>
<td>MySQL</td>
<td>25</td>
<td>10</td>
<td>60%</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>7</td>
<td>4</td>
<td>43%</td>
</tr>
<tr>
<td>Water</td>
<td>79</td>
<td>11</td>
<td>86%</td>
</tr>
</tbody>
</table>
Related Work

• Spin and hang detection
  – [LiTPDS2006], [NakkaEDCC2005],

• Concurrency bug detection
  – [EnglerSOSP2003], [JulaOSDI2008],
    [MusuvathiOSDI2008], [BronPPoPP2005],
    [ParkASPLOS2009],

• Software bug characteristics study
  – [ChouSOSP2001], [LuASPLOS2008],
    [SullivanFTCS1992], [OstrandTSE2005]
Conclusions

• A quantitative study of ad hoc syncs
  – 229 ad hoc sync from 12 concurrent programs.
  – 22-67% of ad hoc loops introduced bugs or performance issues.
  – Impact the accuracy and effectiveness of bug detection and performance profiling.

• **SyncFinder**: a tool that automatically and effectively annotates ad hoc syncs
  – helps to detect new deadlocks
  – helps to improve the accuracy of race detector
Limitations

• For characteristic study, we can study more applications
• SyncFinder requires source code
• SyncFinder misses 1-3 ad hoc syncs
• False positives from SyncFinder requires manual validation
Acknowledgement

• Prof. George Candea (shepherd)
• Anonymous reviewers
• Bob Kuhn, Matthew Frank and Paul Petersen
• NSF and Intel research grants
THANK YOU

http://opera.ucsd.edu