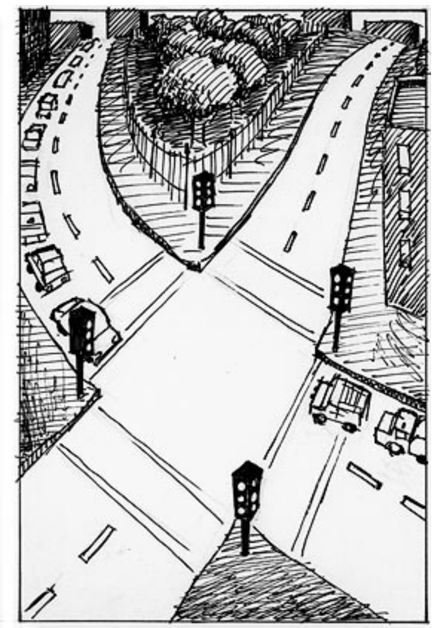
Ad Hoc Synchronization Considered Harmful

Weiwei Xiong, Soyoen Park, Jiaqi Zhang,
Yuanyuan Zhou and Zhiqiang Ma
UC San Diego University of Illinois Intel

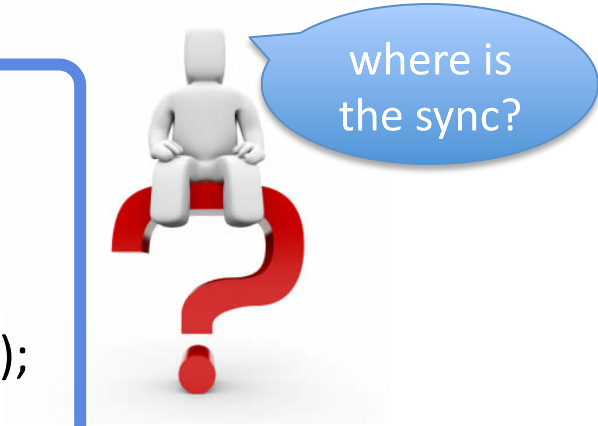
Synchronization is Important

- 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 */
```

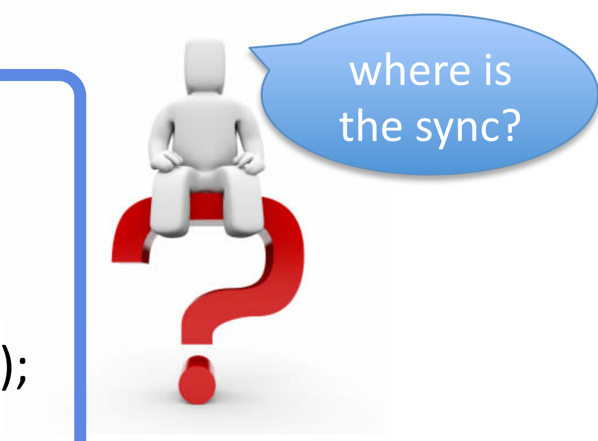


pthread lock

```
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

Common Synchronization Primitives




```
handler handle_slave_sql()
{
   pthread_mutex_lock(&thread_count);
   threads.append(thd);
   pthread_mutex_unlock(&th      count);
}
                                     ySQL */
```

EASY TO IDENTIFY

```
apr_s                        quire()
{
   apr_th        utex_lock(reslist->lock);
   res = po   _resource(reslist);
   apr_thread_mutex_unlock(reslist->lock);
}
                                  /* Apache */
```

customized lock

Hard-to-recognize Synchronization

Ad hoc sync

Sync variable

```
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 */
```

Is it doing 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_C              D))
            TAILQ_REM                  q, ...);
        deleted
    }
    ...
    if (de
        brea
    __os_sleep(dbenv, 1, 0);
}                                   /* OpenLDAP */
```

HARD TO IDENTIFY

Hard-to-recognize Synchronization


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

```
    ...
background_loop:
    /* background operations */
    ...
    if(new_activity_counter > 0)
            goto loop;
    else
            goto background_loop;
```

/* MySQL */

Hard-to-recognize Synchronization


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

      ...
  ba                op:
                    operations */

              _activity_counter > 0)
        goto loop;
    else
        goto background_loop;
                                  /* MySQL */
```

HARD TO IDENTIFY

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?

More examples later

- Introducing bugs or performance issu
  - 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

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

	Apps.	Description
Server	Apache	Web server
	MySQL	Database server
	OpenLDAP	LDAP server
	Cherokee	Web server
Desktop	Mozilla JS	JS engine
	PBZip2	Parallel bzip2
	Transmission	BitTorrent client
Scientific	Radiosity	SPLASH-2
	Barnes	SPLASH-2
	Water	SPLASH-2
	OCean	SPLASH-2
	FFT	SPLASH-2

Every Studied Program Has Ad Hoc Syncs

	Apps.	Description	Ad hoc sync loops
Server	Apache	Web server	33
	MySQL	Database server	83
	OpenLDAP	LDAP server	15
	Cherokee	Web server	6
Desktop	Mozilla JS	JS engine	17
	PBZip2	Parallel bzip2	7
	Transmission	BitTorrent client	13
Scientific	Radiosity	SPLASH-2	12
	Barnes	SPLASH-2	7
	Water	SPLASH-2	9
	Ocean	SPLASH-2	20
	FFT	SPLASH-2	7

Ad Hoc Syncs are Error-prone

- Percentage of buggy ad hoc syncs

Apps.	# ad hoc sync	# buggy sync
Apache	33	7 (22%)
OpenLDAP	15	10 (67%)
Cherokee	6	3 (50%)
Mozilla JS	17	5 (30%)
Transmission	13	8 (62%)

Hard-to-detect Deadlock

Thread 1

```
S1   JS_ACQUIRE_LOCK(rt->setSlotLock);
     ...

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

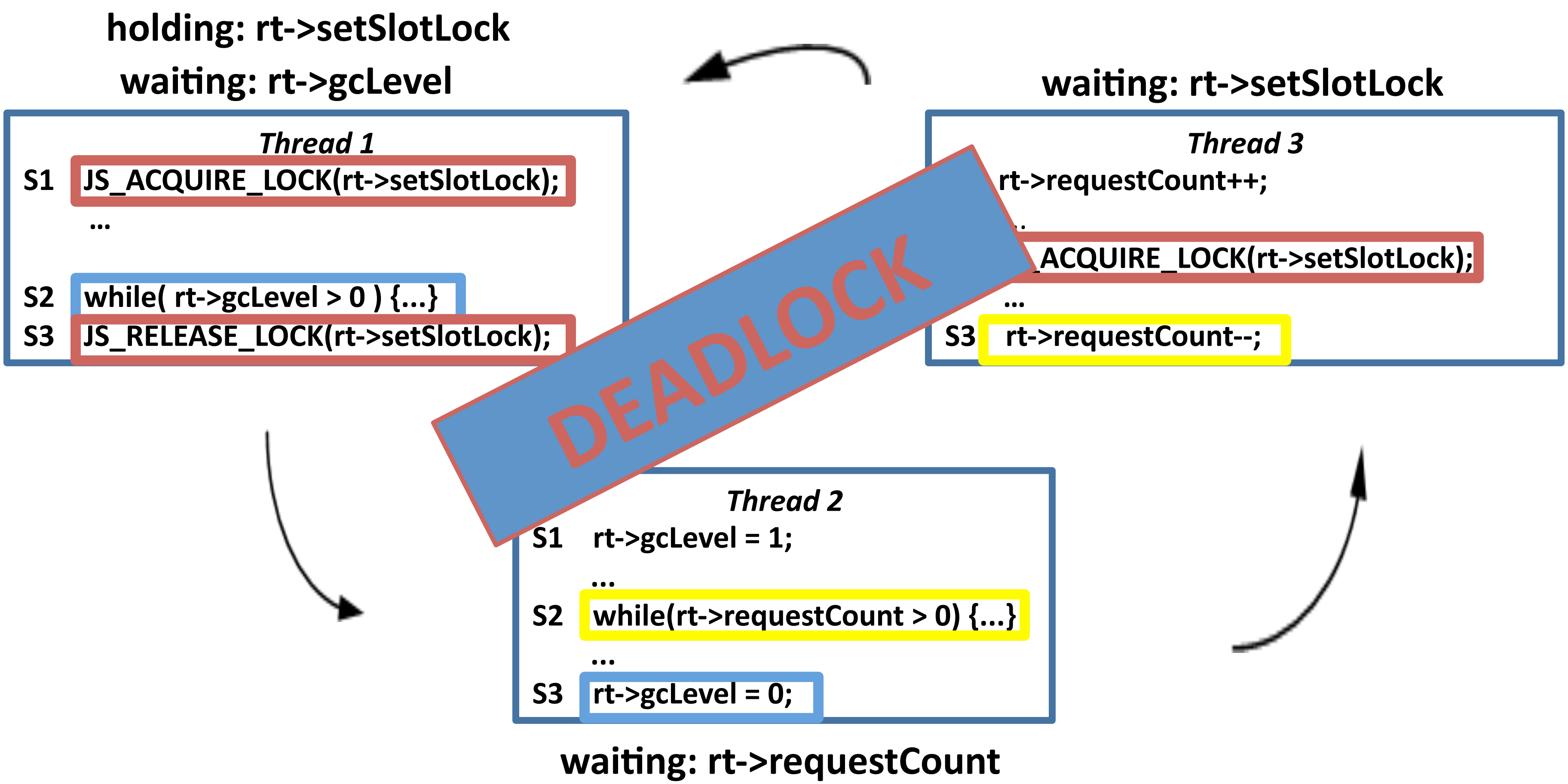
Thread 3

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

Thread 2

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

Hard-to-detect Deadlock



Performance Issues

```
/* get tuple id of a table */
do {
  ret = m_skip_auto_increment ?
          readAutoIncrementValue(…):
          getAutoIncrementValue(…);
 } while (ret == -1 && --retries && …)
```

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

  - False data race on ordered variable accesses

```
            Worker                                   Listener
S1   q_info->pools = new_recycle;       S3   while( q_info->idlers == 0) {…}
     …
S2   atomic_inc( &(q_info->idlers) );   S4   first_pool = q_info->pools;
                                                          /* Apache */
```

Ad Hoc Syncs are Diverse

single cond (sc)

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

dir

```
while (crc_table_empty);
```

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

func

code style

multiple cond (mc)

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

control(cf)

#conditions

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

data(df)

sync variables

Ad Hoc Syncs are Diverse

Apps.	Total ad hoc		Single exit cond.				Multiple exit cond.			Total func	async
		sc-dir	sc-df	sc-cf	sc-func	total	mc-all	mc-Nall	total		
Apache	33	4	0	1	3	8	22	3	25	16	25
MySQL	83	23	5	4	11	43	13	27	40	32	64
OpenLDAP	15	2	0	0	2	4	4	7	11	9	15
Cherokee	6	0	2	0	1	3	0	3	3	1	5
Mozilla JS	17	2	4	1	4	10	4	1	5	5	15
PBZip2	7	0	0	0	1	1	0	6	6	7	7
Transmission	13	6	0	0	1	7	0	6	6	3	2
Radiosity	12	5	5	1	0	11	1	0	1	0	1
Barnes	7	6	1	0	0	1	0	0	0	0	0
Water	9	9	0	0	0	9	0	0	0	0	0
OCean	20	20	0	0	0	20	0	0	0	0	0
FFT	7	7	0	0	0	7	0	0	0	0	0

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

```
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	**Write**
global->pbar_count	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	**Write**
global->pbar_count	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

Apps.	Total loops	True ad hoc syncs	Missed ad hoc syncs	False positives
Apache	1462	15	1	2
MySQL	4265	42	3	6
OpenLDAP	2044	14	1	4
Cherokee	748	6	0	0
Mozilla JS	848	11	1	5
PBZip2	45	7	0	0
Transmission	1114	12	1	2
Radiosity	80	12	0	0
Barnes	88	7	0	0
Water	84	9	0	0
Ocean	339	20	0	0
FFT	57	7	0	0

SyncFinder's Overall Result

Apps.	Total loops	True ad hoc syncs	Missed ad hoc syncs	False positives
Apache	1462	15	1	2
MySQL	4265	42	3	6
OpenLDAP	2044	14	1	4
Cherokee	748	6	0	0
Mozilla JS	848		1	
PBZip2	45		0	
Transmission	1114		1	
Radiosity	80	12	0	0
Barnes	88	7	0	0
Water	84	9	0	0
Ocean	339	20	0	0
FFT	57	7	0	0

average 96%

average 6%

Result on Additional Programs

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

Use cases: Bug Detection

- A tool to detect bad practices

```
LOCK

while(…);

UNLOCK
```

Apps.	Deadlock (New)	Bad practice
Apache	1(0)	1
MySQL	2**(2)**	13
Mozilla	2(0)	2

- Extended race detector in Valgrind

Apps.	Original Valgrind	Extended Valgrind	% Pruned
Apache	30	17	43%
MySQL	25	10	60%
OpenLDAP	7	4	43%
Water	79	11	86%

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

UCSD