Sisteme de operare avansate

MUVI: Automatically Inferring Multi-Variable Access Correlations and Detecting Related Semantic and Concurrency Bugs

Shan Lu[†], Soyeon Park[†], Chongfeng Hu[†], Xiao Ma[†], Weihang Jiang[†]
Zhenmin Li[†‡], Raluca A. Popa[§], Yuanyuan Zhou[†‡]
[†]University of Illinois, [‡] CleanMake Inc., [§] MIT
{shanlu,soyeon,chu7,xiaoma2,wjiang3,zli4,yyzhou}@uiuc.edu, [§]ralucap@mit.edu

Introducere

- Paper-ul adresează două din cele mai dificile clase de bug-uri:
  - bug-uri semantice
  - bug-uri de concurenţă
- Principiul folosit
  - variabilele logic corelate trebuie accesate împreună

Tipuri de bug-uri cauzate de lipsa de corelare

- Actualizare inconsistentă
  - atât pentru programe concurente cât şi pentru programe non-curente
- Buguri de concurenţa

Actualizări inconsistente

```
Class THD
{
   char *db;
 /* currently selected database name*/
   ...
   uint      db_length;
/* length of the database name */
   ...
} /* client connection descriptor*/

        MySQL-5.20 sql_class.h
```

(a) Definition

```
1655   int Event_job_data::execute( ··· )
1656   {
          ......
1674      thd→db = my_strdup(dbname.str);
1675      thd→db_length = dbname.length;
          ......
1701   } /* Execute a connection event*/

   MySQL-5.2 event_data_objects.cc
```

(b) Variable access correlation

Actualizări inconsistente [2]

```
820   void Query_cache::store_query ( ··· )
821   {
          ......
902     if (thd->db_length)
903       memcpy(thd->query,
              thd->db, thd->db_length);
          ......
991   } /* Store a query to cache */

      MySQL-5.2 sql_cache.cc
```

(c) Variable access correlation

```
1721   int Event_job_data::compile( THD* thd)
1722   {
          ......
1820      thd→db= old_db;
          ......
1833   } /* Compile an event*/

      MySQL-5.2 event_data_objects.cc
```

Forgets to write thd→db_length ! Will lead to misbehavior or crash!

(d) **Bug** (violating the access correlation)

Actualizări inconsistente [3]

```
class String
{
  ...
  uint32 str_length;
          /*occupied string length*/
  uint32 Alloced_length;
          /*allocated string length*/
  ...
}
```

MySQL-5.2.0 sql_string.h

(e) Definition

```
184   int String:: free ( ··· )
185   {
          ......
189      Alloced_length = 0;
192      str_length = 0;
          ......
194   } /* free a String */
```

MySQL-5.2 sql_string.h

(f) Variable access correlation

Actualizări inconsistente [4]

```
408  bool String::append ( ··· )
409  {
412      if (Alloced_length < newlen + 1 ){
           ··· /* realloc and copy string */
442       Alloced_length = newlen + 1;
443      }
444      str_length = newlen;
         ...
446  } /* String appending */
     MySQL-5.2 sql_string.cc
```

(g) Variable access correlation

```
663  void String::qs_append ( ... )
664  {
665     memcpy ( Ptr + str_length, str, len+1);
666     str_length += len;
667  }
   MySQL-5.2 sql_string.cc
```

Increasing string-length without even a check on Alloced_length is wrong!

(h) **Bug** (violating the access correlation)

Condiţii de cursă

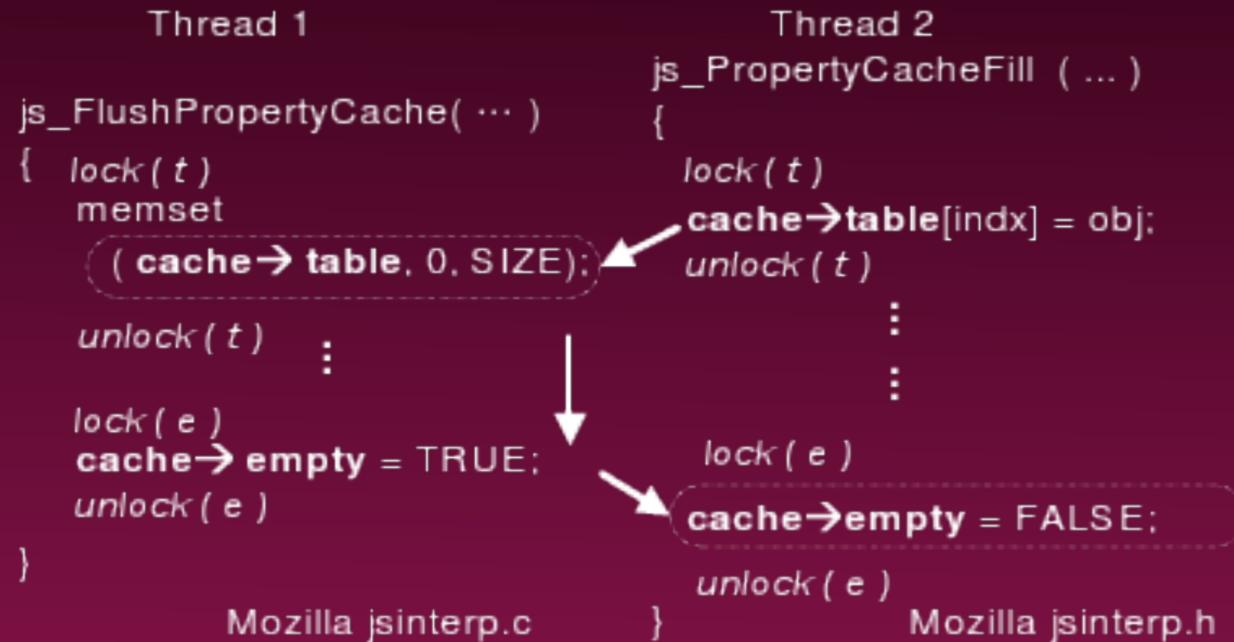
Access interleaving order

Example 1

```
struct JSPropertyCache {
  ...
  JSPropertyCacheEntry
     table [SIZE];


  JSBool empty;
     /* whether the
        table is empty*/
   ...
}
```

Mozilla jsinterp.h



(a) Variables with access correlation

(b) Bug (violating the access correlation due to conflict accesses from another thread, even though no data race on any single variable)

Condiţii de cursă [2]

Inconsistent variables that will lead to crash or wrong results

Example 2

```
struct JSRuntime {
    ...
    uint32 totalStrings;
  /* # of allocated strings*/

    double lengthSum;
     /* Total length of
      allocated strings */
    ...
}
```

Mozilla jscntxt.h

(c) Variables with access correlation

Thread 1	Thread 2
js_NewString(···)	
{	
// allocate a new string	printJSStringStats (...)
JS_ATOMIC_INCREMENT	{
(&(**rt->totalStrings**));	count = **rt →totalStrings;**
	mean = **rt →lengthSum** / count;
	printf ("%lu strings,
PR_Lock(rtLock);	mean length %g \n" ,
rt->lengthSum += length;	count, mean);
PR_Unlock(rtLock);	}
}	
Mozilla jsstr.h	Mozilla jsstr.c

(d) **Bug** (using different locks to protect correlated variable accesses leads to multi-variable concurrency bugs)

Contribuţiile paper-ului

- Detectarea automată a variabilelor ce trebuiesc accesate corelat
  - funcţionează pe programe de dimensiuni mari (Linux, Mozilla, MySql, PostgreSql)
  - tehnici de analiză statică şi data mining
  - gradează corelaţiile
  - reduce false positive-urile
  - timp de rulare: 18-175 minute (0.8 – 3.6 milioane linii de cod); au fost identificate ~6500 de accese corelate cu un grade de acurateţe de 83%

Contribuţiile paper-ului [2]

- Detectarea automată a variabilelor ce trebuiesc accesate corelat [2]
  - odată identificate, corelaţii pot fi
    - documentate într-o specificaţie
    - folosite pentru a anota codul sursă pentru ca alte utilitare să poată folosi aceste informaţii semantice (AutoLocker, Colorama, AtomicSet)

Contribuţiile paper-ului [3]

- Detectarea actualizărilor inconsistente
  - primul utilitar de acest gen
  - 39 de buguri detectate în Linux, Mozilla, MySQL, PostgreSql (22, 7, 9, 1)
    - 17 confirmate
  - 20 modalităţi de folosire riscante
  - 41% false pozitives

Contribuţiile paper-ului [3]

- Adresează limitările fundamentale ale metodelor de detecţie a bugurilor de concurenţa anterior folosite
  - extinde două metode clasice: lock-set, happens-before pentru a lua în considerare cazul multi-variabilelor
  - discută modalităţi de extindere pentru alte detectoare de bug-uri cum ar fi AVIO, RaceTrack, RacerX
  - evaluaează aceste extensii prin detectarea a 5 bug-uri a căror cauză un putea fi detectată anterior; 4 alte noi buguri sunt descoperite

Corelarea variabilelor

- Există multe cazuri pentru care variabilele pot fi (semantic) corelate:
  - specificarea unei constrângeri: o variabilă exprimă o constrângere, o stare sau o proprietate pentru o altă variabilă (e.g. thd->db_length, thd->db)
  - reprezentări diferite:informaţia este reprezentată în moduri diferite (e.g. rx_bytes, rx_packets)
  - aspecte diferite: variabile individuale sunt folosite pentru a reprezenta diverse aspecte (e.g. tm_min, tm_sec)
  - cerinţe de implementare: mai multe variabile cooperează pentru a implementa o anumită funcţionalitate (e.g. .next şi .prev dintr-o listă)

Exemple

Variables **With** access correlation

ID	source	Variable definitions	# of functions they are together (not)

a	Linux net-device.h	struct net_device_stats {
		u64 rx_bytes; /* #of received bytes */
		u64 rx_packets; /* #of received packets*/
		}
			49 (1)

b	PgSQL time.h	struct tm { int tm_sec; /* second */ int tm_min; /* minute */ } /* time */	25 (0)
c	Linux fb.h	struct fb_var_screeninfo { u32 red_msb; /* red */ u32 blue_msb; /* blue */ u32 green_msb; /*green*/ u32 transp_msb; /*transparency*/ } /* for color display */	11 (1)

d	Linux libiscsi.h	struct iscsi_session {
		spinlock_t lock; /* lock */
		int state; /* critical data */
		}
			20 (0)

e	Linux list.h	struct hlist_node { struct hlist_node *next; /* next */ struct hlist_node **pprev; /* pevious */ } /* linked list */	32 (0)
f	MySQL mysql-test.c	struct st_test_file* cur_file; struct st_test_file* file_stack; /* cur_file points to the top of stack */	69 (0)

Exemple [2]

Variables **WithOUT** access correlation				

	g	Linux net-device.h	struct net_device_stats {
			u64 rx_bytes; /* #of received bytes */
			u64 tx_aborted_erros; /* #of transfer aborts*/
			}
				4 (68)

	h	MySQL sql_class.h	Class THD {
			NET net; /* client connection descriptor */
			uint db_length; /*length of database name*/
			}
				3 (87)

Constrângeri de access în corelări

- read(x) => read(y): citirea lui x necesită citirea y
  - Ex: cache->table trebuie precedat de verificarer valorii lui cache->empty
- write(x) => write(y): scrierea în x necesită scrierea în y
  - Ex: modificarea thd->db necesită modificarea thd->db_length

Constrângeri de access în corelări [2]

- write(x) => AnyAcc(y): scrierea în x necesită scrierea sau citirea din y
    - Ex: scrierea în state trebuie precedată de verificarea sau modificarea (luarea) lock-ului
- AnyAcc(x) => AnyAcc(y): accesare împreună
    - Ex: accesarea (read sau write) a oricărui câmp red, green, blue, transp necesită accesarea tuturor câmpurilor

Analiza variabilelor corelate

- Inferare automată
- Se presupune că programul este suficient de matur
- Se examinează care din variabile sunt citite/scrise împreună

Accesarea împreună

- Posibilităţi:
  - distanţa în cod binar dintre accesări
  - distanţa în codul sursă dintre accesări
  - gruparea acceselor în cadrul aceluiaşi basic block, funcţii sau fişier
- Metrica folosită:
  - accesările din cadrul aceleaşi funcţii, cu distanţa în codul sursă sub *MaxDistances*

Accese corelate

- x este corelat cu y, i.e. A1(x) ⇒A2(y), dacă A1(x) şi A2(y) apar împrerună de cel puţin de MinSupport ori şi de câte ori apare A1(x), A2(y) apare împreună cu probabilitatea de cel puţin MinConfidence
- MinSupport, MinConfidence parametri setaţi
- A1, A2 pot fi: read, write sau AnyAcc

Paşii folosiţi pentru identificarea variabilelor corelate

- Colectarea infomaţiilor de accesare:
  - se colectează informaţii despre fiecare variabilă din fiecare funcţie (modul de access şi locaţia) în Acc_Set
- Analiza patternului de accesare:
  - “frequent pattern mining technique”
  - se generează un set de canditaţi
- Generarea corelărilor
  - gradarea si alegerea candidaţilor

Colectarea informaţiilor de accesare

- Tipuri de variabile analizate:
  - variabile globale
  - câmpuri din structuri/clase (pot fi local, global sau dinamic alocate)
- Informaţii necesare despre fiecare access:
  - tipul: read, write
  - locaţia în sursă (numele funcţiei şi linia)
- Modul de referire a variabilei din funcţie
  - direct (access direct din funcţie)
  - indirect (access din alte funcţii chemate din funcţia curentă); se ia în considerare doar primul nivel, locaţia în sursă se setează la locaţia apelului de funcţie

Exemplu

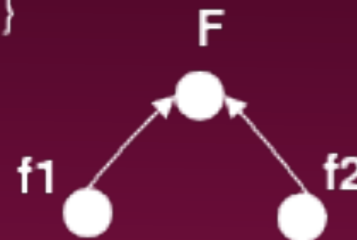
```
F ( ) {
  f1 ( );
  read x ;
  f2 ( );
}
```

```
f1 ( ) {
a:  read y ;
}
```

```
f2 ( ) {
  read z ;
}
```

(a) Exemplary code

Acc_Set (F)
= { x } U Acc_Set (f1) U Acc_Set (f2)
= { x, y, z }



Acc_Set (f1) = { y } Acc_Set (f2) = { z }

(b) Acc_Sets are generated based on call-graph

Limitări

- Algoritmul este independent de context
    - ar trebui ţinut un Acc_Set pentru fiecare call de funcţie
- Biblioteci fară cod sursă
    - Acc_Set construit manual pentru funcţiile importante (memcpy, strlen, etc.)
    - În viitor: analiză cod binar
- Pointer aliasing:
    - nu e o problemă pentru structuri/clase pentru că analiza este object agnostic
    - poat fi o problemă pentru variabilele globale, dar rezultatele obţinute indică contrariul

Analiza patternului de accesare

- Access pattern = un set de variabilelor care sunt accesate în cadrul aceleaşi funcţii de un număr de ori ce depăşeşte o anumită limită
- Access pattern != corelări
- Access pattern poate infera un set de candidaţi pentru corelare
- Tehnica folosită: “frequent itemset mining” [13]
  - algoritmul folosit: FPclose [13]

Frequent itemset mining

- Avem o bază de date ce conţine un set de itemset-uri (itemset = set de obiecte)
- Tehnica determină în mod eficient ce subitemset-uri sunt frecvente, i.e., conţinute în mai mult de MinSupport itemset-uri
- Exemplu:
  - D = {{w, y, z}, {v, w, y, z}, {w, x, y}}
    - MinSupport=3 -> {w}, {y}, {w, y}
    - MinSupport=2 -> {w}, {y}, {z}, {w, y}, {w, z}, {y, z}, {w, y, z}

Generarera corelărilor

- Generează corelările, elimină false positive-urile, şi gradează corelările folosind metrici
- Metrici
  - Support
    - Pentru o corelare C: A1(x) => A2(y), support(C)=numărul de funcţii în care A(x) şi A2(y) sunt (acceste) împreună
    - MinSupport
  - Confidence
    - Pentru o corelare C: A1(x) => A2(y), confidence(C)=support(C)/ support(A1(x)); support(A1(x))=numărul de funcţii care efectuează A1 (x)
    - MinConfidence

Genererarera corelărilor [2]

- MinDirectSupport
    - trebuie să existe un minim de accesări directe
- Se ignoră variabile care apar în foarte multe locuri (e.g. stderr, stdin)
- Gradarea
    - daca support e suficient de mare, se gradează după confidence
    - altfel se gradează după support

Alegerea parametrilor

- Compromis între false positives şi false negatives
- Determinaţi experimentali
- Poti fi modificaţi
- Setările default ar trebui să fie ok pentru majoritatea aplicaţiilor

Bug-uri cauzate de actualizări inconsistente

- În contextul curent, un astfel de bug poate fi exprimate prin faptul ca nu se respectă corelarea write=>AnyAcc
- Detecţia:
    - pentru fiecare corelare write(x)=>AnyAcc(y) se caută operaţii de scriere asupra x care nu sunt grupate cu accesarea y
- Exemplu confirmat:
    - In Linux: velocity_receive_frame violates write (net_device_stats::rx_packets) ⇒write (net_device_stats ::rx_bytes)

Gradarea şi eliminarea false pozitive-urilor

- Dacă avem un bug candidate pentru funcţia F în care y un se cheamă, dar se cheamă într-o funcţie chemată din F sau o funcţie ce chemă F, se ignoră
    - se verifică doar două nivele
- Gradarea
    - violările corelărilor write -> write primesc grade mari
    - dacă există multe violări gradul scade
    - cu cât corelaţia are un scor mai bun, cu atât scorul pentru bug creşte

Bug-uri de concurenţă

- Condiţii de cursă
  - detectoarele de condiţii de cursă existente se focalizează pe o singură variabilă, nu iau în considerare cazul variabilelor corelate
- Algoritmi ce detectează condiţiile de cursă
  - Lock-set
  - Happens-before

Lock-set

- Un set de lock-uri luate de fiecare thread (Lock Set)
- Un set de lock-uri folosite deja pentru a proteja o variabilă (Candidate Set)
- La fiecare accesare, CS = CS intersectat cu LS
- Initial CS = LS
- Dacă CS devine mulţimea vidă avem o posibilă condiţie de cursă

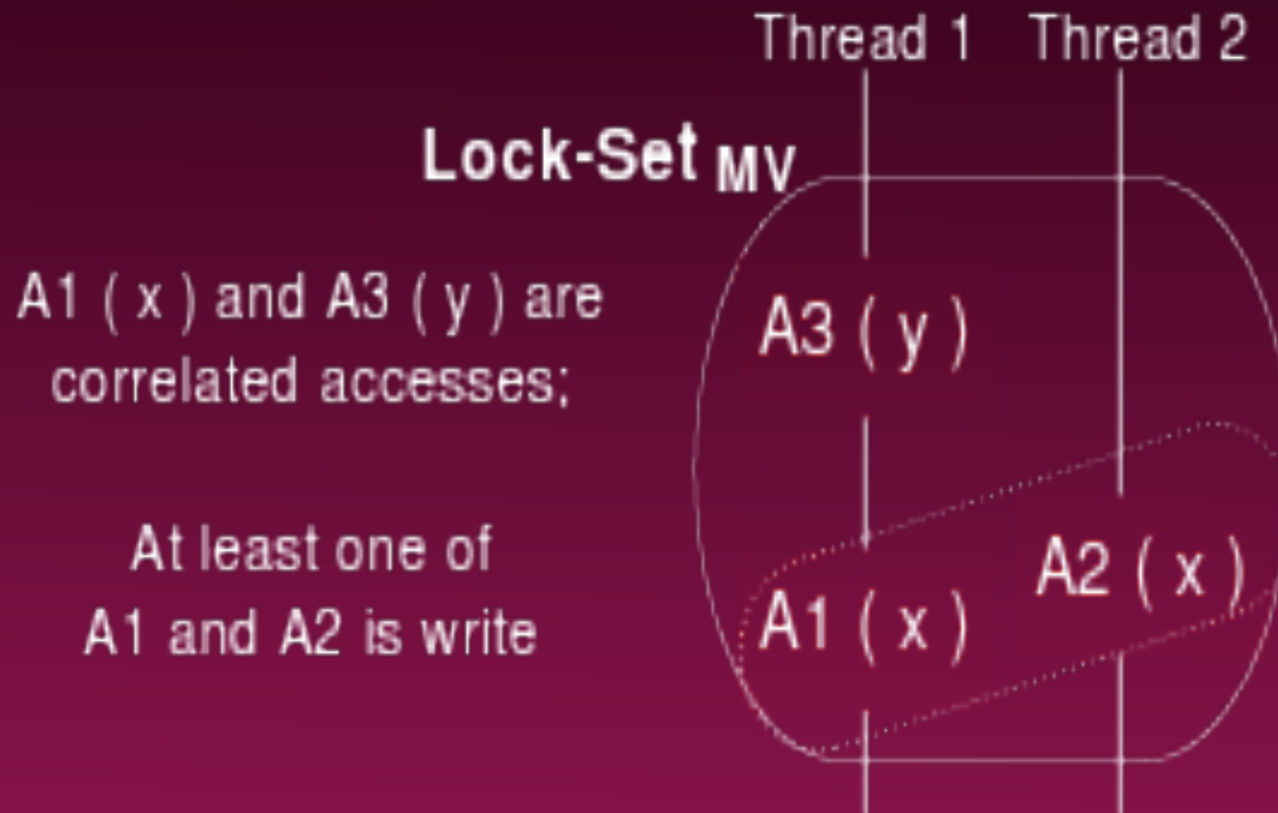
Lock-set [2]

Program	*locks_held*	*C(v)*
	{}	{mu1,mu2}
lock(mu1);		
	{mu1}	
v := v+1;		
		{mu1}
unlock(mu1);		
	{}	
lock(mu2);		
	{mu2}	
v := v+1;		
		{}
unlock(mu2);		
	{}	

Extinderea Lock-Set pentru mai multe variabile

- A1(x) – threadul 1, A2(x) – threadul 2
- Se verifică accesul la fel ca şi la Lock-Set
- Dacă există un A3(y) corelat cu A1(x) sau A2(x), atunci şi A3(y) trebuie să fie protejat de acelaşi lock

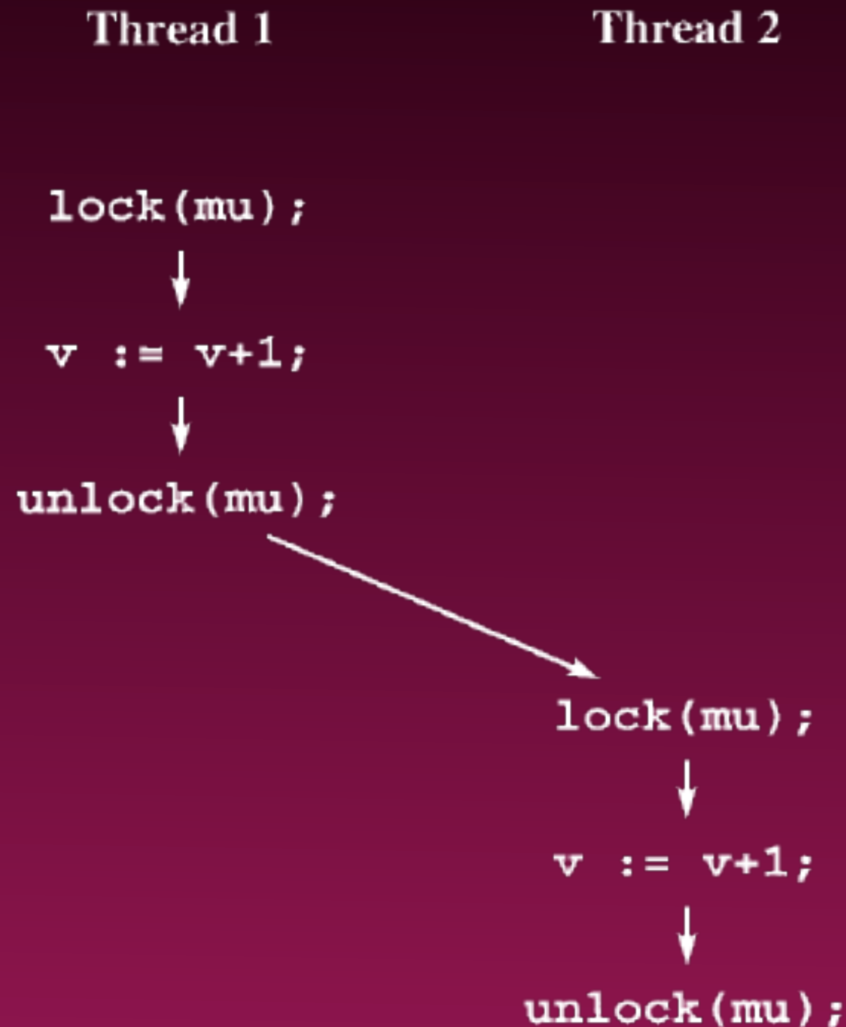
Extinderea Lock-Set pentru mai multe variabile [2]



Happens-before

- Algoritmul stabileşte relaţii de ordine temporală între operaţii
- A se execută înainte de B (A -> B)
  - dacă se execută din acelaşi thread şi A se execută înainte de B
  - A execută unlock(L) şi B execută lock(L)
  - Dacă există un C astfel încât A->C şi C->B

Happens-before [2]



Happens before [2]

- Două evenimente A şi B sunt concurente dacă
  - nici A->B şi
  - nici B->A
- Există o condiţie de cursă dacă:
  - există două evenimente concurente A şi B
  - A şi B accesează aceaşi locaţie de memorie
  - unul din accese este de tip scriere

Extinderea Happens-Before pentru mai multe variabile

- Se va face ordonarea temporală şi pentru variabilele corelate
- Dacă se detectează că nu există o ordine temporală între oricare din variabile (corelate) înseamnă că am identificat un race

Rezultate experimentale

Application	Version	LOC	Description
Linux (drivers)	2.6.20	3.6M	Operating System
Mozilla-Firefox	2.0.0.1	3.4M	Web browser
MySQL	5.2.0	1.9M	Database Server
PostgreSQL	8.2.3	832K	Database Server

Rezultate experimentale [2]

BugId	App.	Description
Moz-js1	Mozilla-suite v0.9	Wrong-ordered concurrent updates make empty table's *empty* flag false; leads to system crash (Figure 1)
Moz-js2	Mozilla-suite v0.8	Wrong-ordered read/write to `gcPoke` flag leads to reading wrong `liveAtoms`; makes garbage collection failure
Moz-imap	Mozilla-Thunder--bird v1.7	Wrong-ordered concurrent updates make URL-in-progress flag true, but URL string NULL; system crash
MySQL-log	MySQL-v4.0.16	Un-atomic read to log-file's name and log-file are interleaved by remote thread switching file; log-file failure
MySQL-blog	MySQL-v3.23.56	Un-atomic table deletion and logging are interleaved by remote thread's table insertion and logging; security problem (Figure 8)

Rezultate experimentale [3]

App.	#Access-Correlations	#Involved Variables	#Involved Structures	%False Positive	Analysis Time
Linux	3353	3038	587	19%	175m2s
Mozilla	1431	1380	394	16%	157m40s
MySQL	726	703	209	13%	19m25s
PostgreSQL	939	833	277	15%	98m23s
Total	**6449**	**5954**	**1467**	**17%***	**450m30s**

Rezultate experimentale [4]

App.	#MUVI Bug Report	#New Bugs Found	#New Bugs Confirmed	#Bad program-ming	#False Positives	False pos. sources		
						S1	S2	S3
Linux	40	22	12	5	13	6	3	4
Mozilla	30	7	0	8	15	8	7	0
MySQL	20	9	5	3	8	5	2	1
PgSQL	10	1	0	4	5	5	0	0
Total	100	**39**	**17**	20	41	24	12	5

Rezultate experimentale [5]

Bug	Lock-set$_{MV}$			Happens-before$_{MV}$		
	Detect Bug?	False Pos.	Over-head*	Detect Bug?	False Pos.	Over-head*
Moz-js1	Y	1	39.9%	Y	1	21.2%
Moz-js2	Y	2	39.8%	Y	5	1.0%
Moz-imap	Y	0	13.2%	Y	0	1.0%
MySQL-log	Y	3	6.5%	Y	6	5.0%
MySQL-blog	N	0	5.9%	N	1	3.2%

Note: In addition to the above existing concurrency bugs, we detected four *new* multi-variable concurrency bugs that have never been reported before.

Rezultate experimentale [6]

```
struct JSDHashTable {
    ...
    uint32 entryCount;
     /* number of entries
                  in table */
    uint32  removedCount;
     /* number of entries
     removed from table */
    ...
}
```

Mozilla jsdhash.h

(a) Variables with access correlation

Thread 1

```
JS_DHashTableRawRemove( ··· )
{
  ...
  /* remove an entry from table */
  table→ removedCount++;

  table→ entryCount--;
}
```

Mozilla jsdhash.c

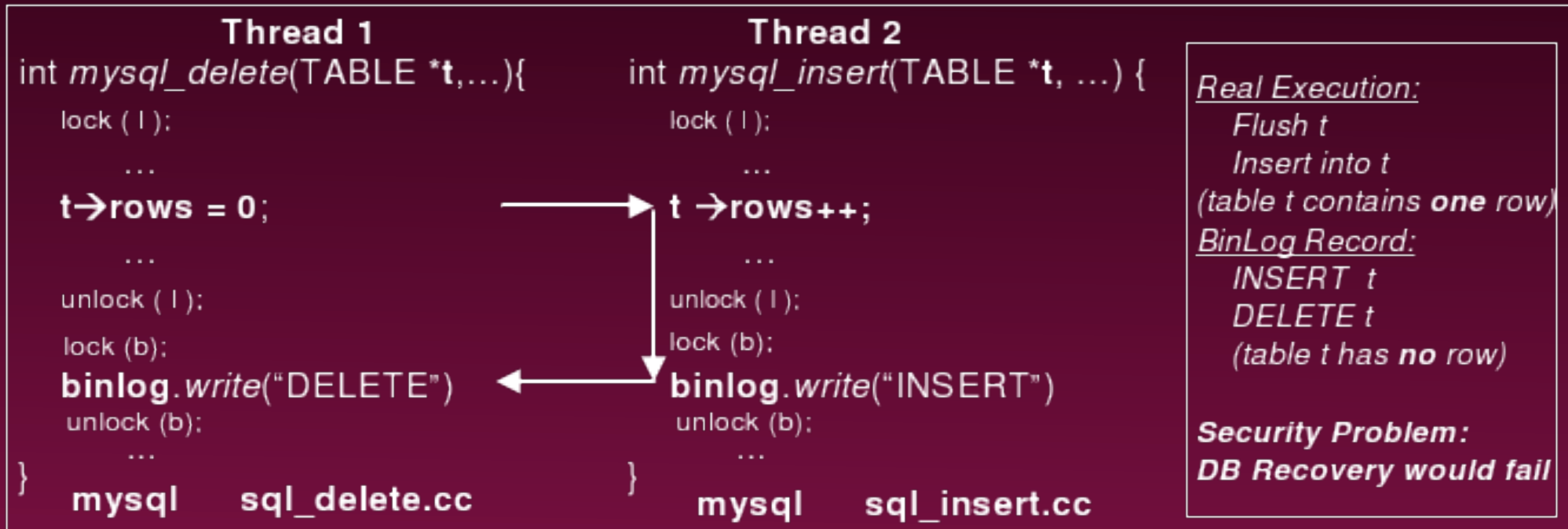
Thread 2

```
JS_DHashTableEnumerate ( ... )
{/* shrink table if many entries removed*/
  if ( table→ removedCount
                  >= threshold ) {
    entries = table→ entryCount;
   /*checking & calculating new capacity
    based on entries*/
  }
  ...
 /* shrink table based on above capacity*/
}
```

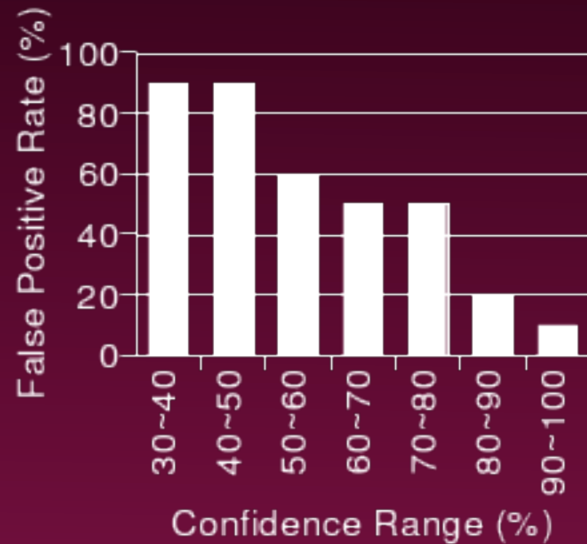
Mozilla jsdhash.c

(b) Bug

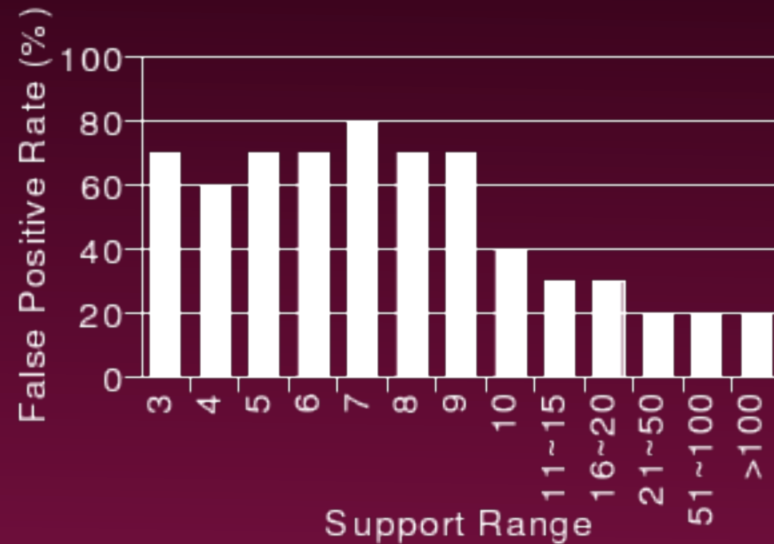
Rezultate experimentale [7]



Rezultate experimentale [8]



(a) Confidence



(b) Support

Rezultate experimentale [9]