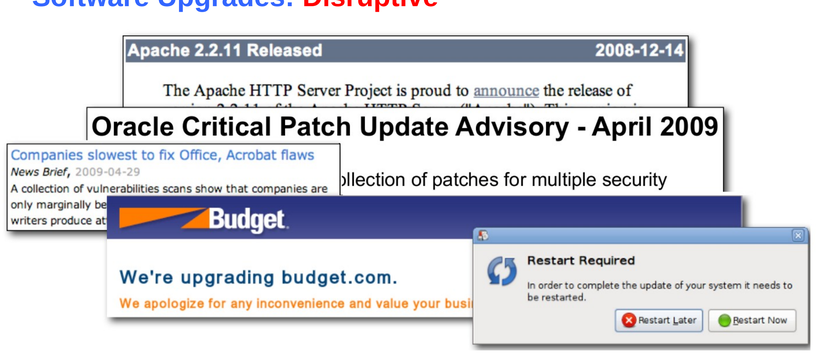
A Study of Dynamic Software Update Quiescence for Multithreaded Programs

Christopher M. Hayden, Karla Saur, Michael Hicks, Jeffrey S. Foster

University of Maryland

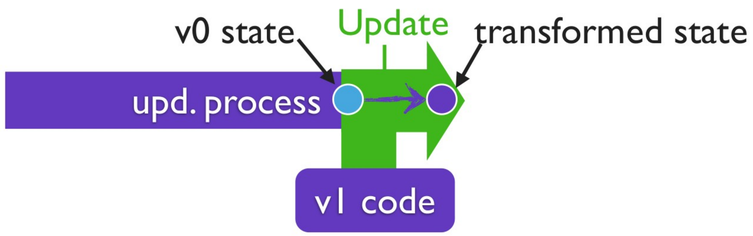
Software Upgrades: Useful

Software Upgrades: Disruptive



Dynamic Software Updating (DSU)

Goal: Update programs while they run



At some point during execution of a program, update to the new version. DSU must preserve and update existing program state such as existing connections, important data on the stack and heap, and the program counter.

Multithreaded Dynamic Software Updating

We support updates at developer-identified ***quiescent points***, which are program locations that are reached between iterations of event-processing loops and at which point there is typically less in-flight state.

```
1 void *thread_entry(void *arg) {
2   /* thread init code */
3   while (1) {
4     qbench_update(); /* update point */
5     /* loop body: typically handles a single program event */
6   }
7 }
```

The state in which *all program threads* have reached an update point is ***full quiescence***, allowing an update to take place.

Goal: Reach full quiescence as quickly as possible.

Challenges:

Achieving full quiescence may be delayed or thwarted by blocking calls (waiting for I/O, waiting for condition variables).

Example: One thread could hold a mutex when it reaches its update point, but then another thread could block on the same mutex prior to reaching its own update point, delaying full quiescence indefinitely.

Solution: QBench Library

QBench provides calls that allow blocking calls to be interrupted.

Blocking I/O Calls: QBench's main signal handler sends a signal to any other thread that has not already reached its update point and is not waiting on a condition variable.

Blocking Condition Variables: QBench replaces the pthread conditional wait call with a call that notes the condition variable argument in the global list of threads so that it can later be signaled by another thread once an update has been requested.

QBench Library Calls to Expediate Quiescence

```
1  void *thread_entry(void *arg) {
2    /* thread init code */
3    while (1) {
4      qbench_update();
5      pthread_mutex_lock(&mutex);
6      while (!input_is_ready() && !qbench_update_requested()) {
7        qbench_pthread_cond_wait(&cond, &mutex);
8      }
9      pthread_mutex_unlock(&mutex);
10     if (qbench_update_requested())
11       continue; /* reaches qbench_update */
12     /* ... handle connection */
13   }
14 }
```

Allows thread to be signaled for update even when waiting on a condition variable

Check for a requested update at top of thread event loop

Results: Few Changes Necessary

We tested programs covering a wide range of domains including media streaming, caching, intrusion detection, and gaming.

The changes required to support full quiescence were small:

Program	LoC Total	# of Threads	Upd Points	Changed LoC (†)	Required Manual Chgs
httpd-2.2.22	232651	$2 + c^*, \quad c = 3$	5	7 (5)	3 (Cond. Var. Loop)
icecast-2.3.2	17038	6	12	3 (3)	1 (Thread Sleeps)
iperf-2.0.5	3996	$3 + n^\circ, \quad n = 1$	5	8 (3)	1 (Cond. Var. Loop)
memcached-1.4.13	9404	$2 + c^*, \quad c = 4$	4	27 (4)	2 (libevent changes)
space-tyrant-0.354	8721	$3 + 2n^\circ, n = 5$	6	8 (6)	1 (Thread Sleeps)
spread-mon-3.17.4	837	4	4	3 (3)	0
suricata-1.2.1	260344	$8 + c^*, \quad c = 3$	7	11 (6)	1 (libpcap break)

*Configurable: c workers °Varies by n connected clients †Calls to QBench excluding update

Results: Time to Quiescence < 1ms on average

In some cases quiescence would not have occurred without changes. The QBench library improved times to reach quiescence in all cases.

Program	w/Load (ms)		w/o Load (ms)	
	All Chgs	Upd only	All Chgs	Upd only
httpd-2.2.22	0.185	0.230	0.123	0.150
icecast-2.3.2	105.152	954.32	107.558	986.265
iperf-2.0.5	0.193	DNQ	0.169	DNQ
memcached-1.4.13	0.166	DNQ	0.155	DNQ
space-tyrant-0.354	0.426	20.583	0.078	20.304
spread-mon-3.17.4	0.200	same	0.196	same
suricata-1.2.1	0.503	68.098	0.378	DNQ

DNQ = Does not quiesce.