I/O and Syscalls in Critical Sections and their Implications for Transactional Memory

Lee Baugh and Craig Zilles
University of Illinois at Urbana-Champaign

Side-Effects in Transactions

```
begin_transaction();
myarr[x]= fgetc(myfile);
end_transaction();
```

- *Will* programmers use side-effects?
- *How* will programmers use side-effects?
- *What* implications does this have on proposed mechanisms handling side-effects in transactions?

Analyzing Side-Effects in Transactions

- … is pretty tough because there are no large transactional workloads
- We assume that side-effects in current critical sections are representative of transactions
    - So we looked inside critical sections in two large, multithreaded applications: **Firefox** and **MySQL**

Our Findings

- Critical sections do perform side-effects
  - … and not just for mutual exclusion on I/O resources
- Side-effecting critical sections tend to be long
- Side-effects are distributed through their lives
- Side-effects' outputs tend to be consumed (*deferral* unlikely)
- Serializing side-effecting transactions can be viable
  - If non-conflicting transactions aren't serialized
- *Compensation* can service >90% of side-effecting operations
  - Can be integrated with transactional filesystem and system library
- No proposed transactional I/O technique dominates

Existing TM I/O Proposals

- **Outlaw**: simply forbid any non-*protected* actions inside transaction.
  - \+ clean semantics
  - \- Limits programmability and composition severely
- **Defer**: postpone side-effecting actions until commit
  - \- Prohibits dependences on side-effecting actions
- "**Go Nonspeculative**": serialize side-effecting transactions
  - \+ Very simple and transparent, permits dependences
  - \- Can affect performance; precludes *explicit aborts*
- **Compensate**: protect *unprotected* actions with compensation code
  - \+ Permits explicit aborts, doesn't serialize, permits dependences
  - \- New source of bugs, no implicit isolation or conflict detection

Experimental Method: What's a Side-Effecting Action?

- In TM, side-effects are I/O
- Three ways to perform I/O in x86:
  - `in` and `out` instrs: not seen in critsecs
  - memory-mapped I/O: only performed by the kernel and the single X11 thread
  - syscalls: what we saw plenty of

Experimental Method:
Should all syscalls be considered side-effecting?

- Prior work suggests application transactions ought not to subsume kernel-mode work
  - Performance isolation can be lost in kernel sharing
  - STMs *cannot* subsume kernel-mode work
- So we consider all syscalls to be performed extra-transactionally, and thus potentially side-effecting

Experimental Method

- We use Pin for binary instrumentation
  - Tracked critical sections by counting `pthread_mutex` acquires and releases
  - Only considered top-level critical sections
  - Looked for syscalls in critical sections:
    - when they happened
    - what they were
    - which critical sections they lived in

Results: Syscalls Seen

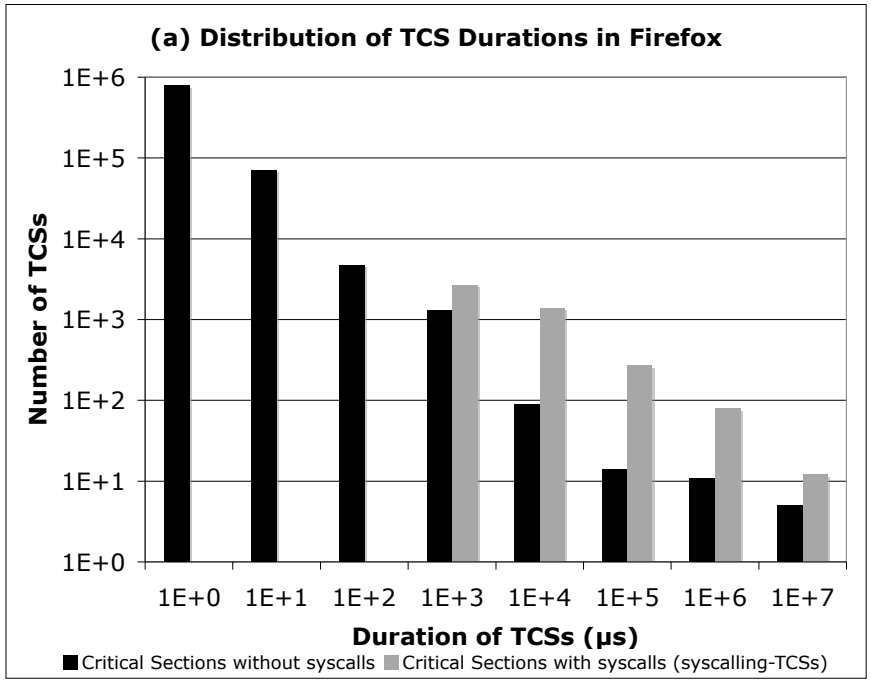
Category of Syscall	Syscalls Seen in Critical Sections	Frequency in Critsecs	
		MySQL	Firefox
Time	`gettimeofday`, `clock_gettime`	3.91%	70.18%
Filesystem	`read*`, `write*`, `open`, `close`, `lseek`, `access`, `dup`, `mkdir`, `ftruncate`, `fsync`, `writev`, `pread*`, `pwrite*`, `stat`, `fstat`, `fcntl`, `getdents`, `getcwd`, `fdatasync`, `mmap*`, `munmap*`, `mprotect*`	53.79%	28.75%
Process Memory	`brk`, `mmap*`, `munmap*`, `mprotect*`	31.03%	0.32%
Process Maintenance	`waitpid`, `clone`, `sched_setscheduler`, `sched_get_priority_max`, `sched_get_priority_min`, `rt_sigaction`, `rt_sigprocmask`, `tgkill`	8.97%	0.32%
Communication	`ioctl`, `socket`, `pipe`, `read*`, `write*`, `pread*`, `pwrite*`	2.07%	0.40%
System Info	`sysinfo`, `uname`	0.23%	0.03%

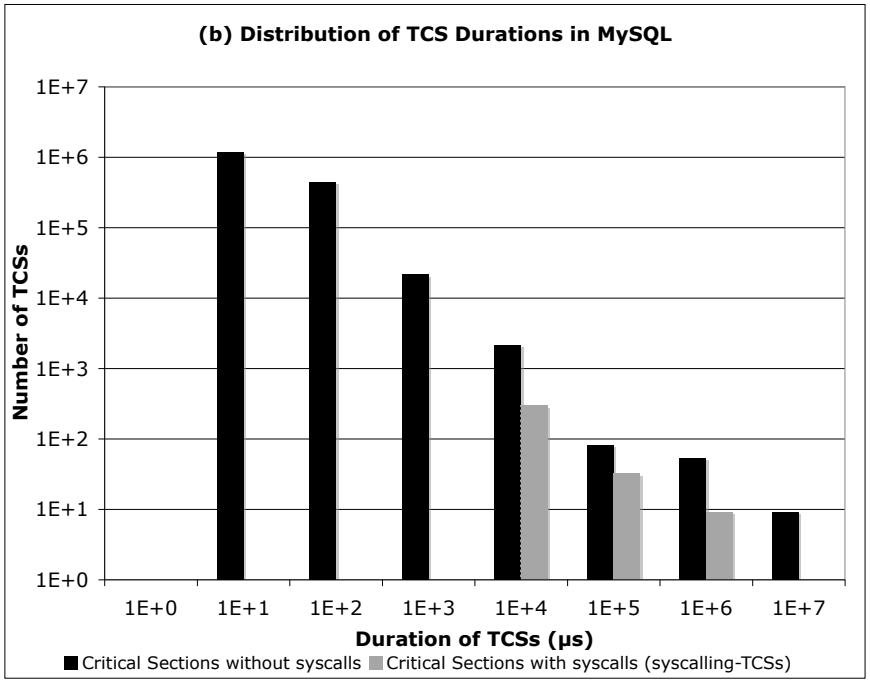
- A *Transactional Filesystem* can protect filesystem syscalls
- Can the rest be compensated for?

Results: the Advantage of Compensation

- Found four "protection classes" among the syscalls we observed, representing what protection they require at the scope of the call:
  - **Null compensation** syscalls require no *protection* -- e.g. '`gettimeofday`'
    - over 70% in Firefox, under 10% in MySQL
  - **Memory-fixup** syscalls only affect kernel state; can easily be *compensated* -- e.g. '`lseek`'
  - **Full compensation** syscalls perform *unprotected* I/O actions, and require '*going nonspeculative*' or *compensation* -- e.g. an '`open`' call creating a file may compensate with '`unlink`'
  - **Real** syscalls cannot be adequately *compensated* for at the scope of the call -- e.g. '`tgkill`', '`socket`'. Programmers may compensate at higher levels
    - 7% in MySQL, <1% in Firefox
- Compensation code within the system library is widely applicable

Results: Critical Section Length

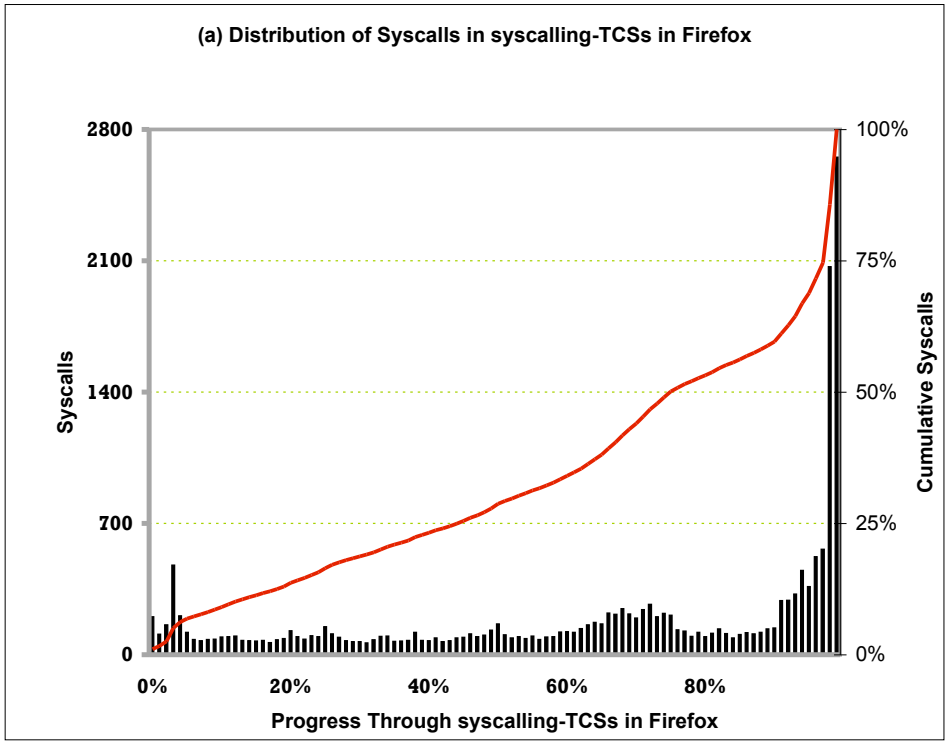


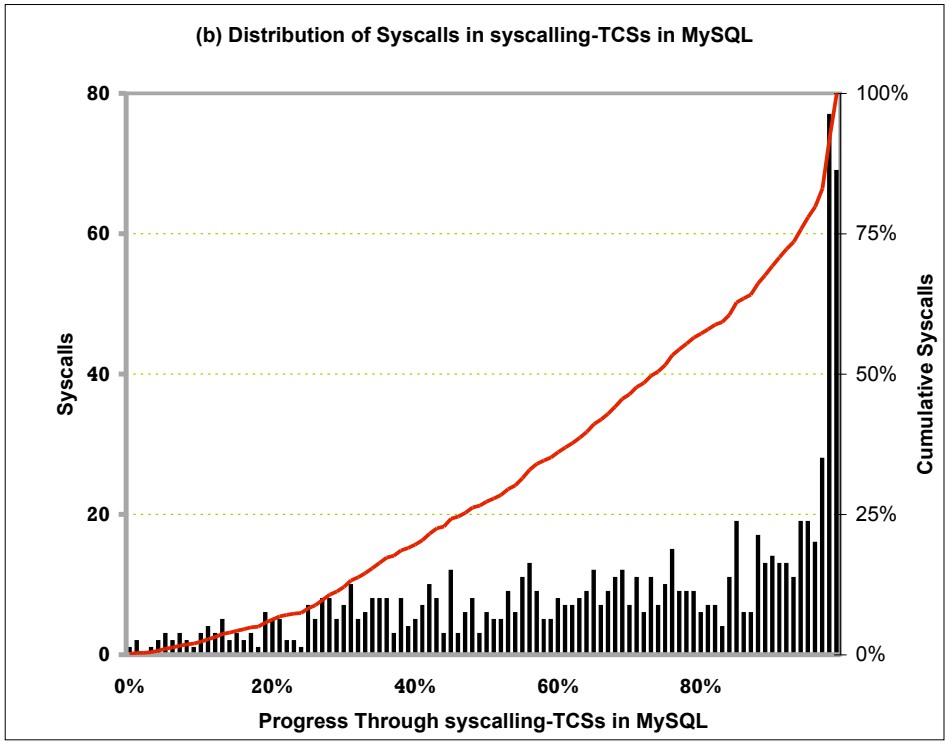


- Syscalling Toplevel Critical Sections (TCSs) are a lot longer than non-syscalling TCSs

➡ Syscalls *deferred* for more time; transactions *going nonspeculative* -- that is, serializing -- for longer

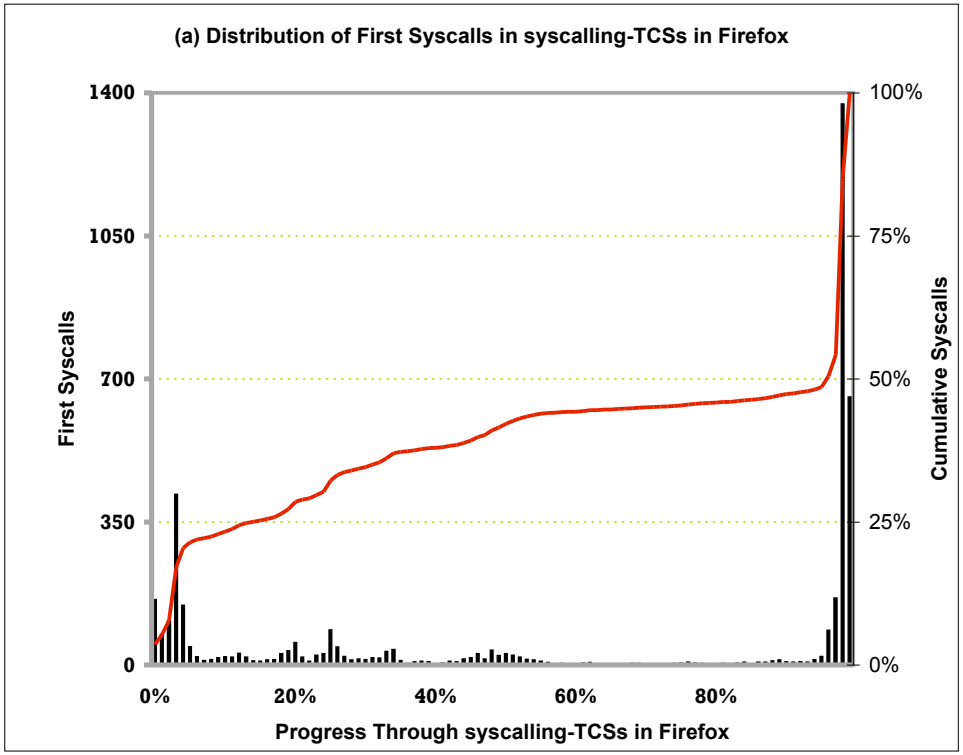
Results: Syscall Distribution

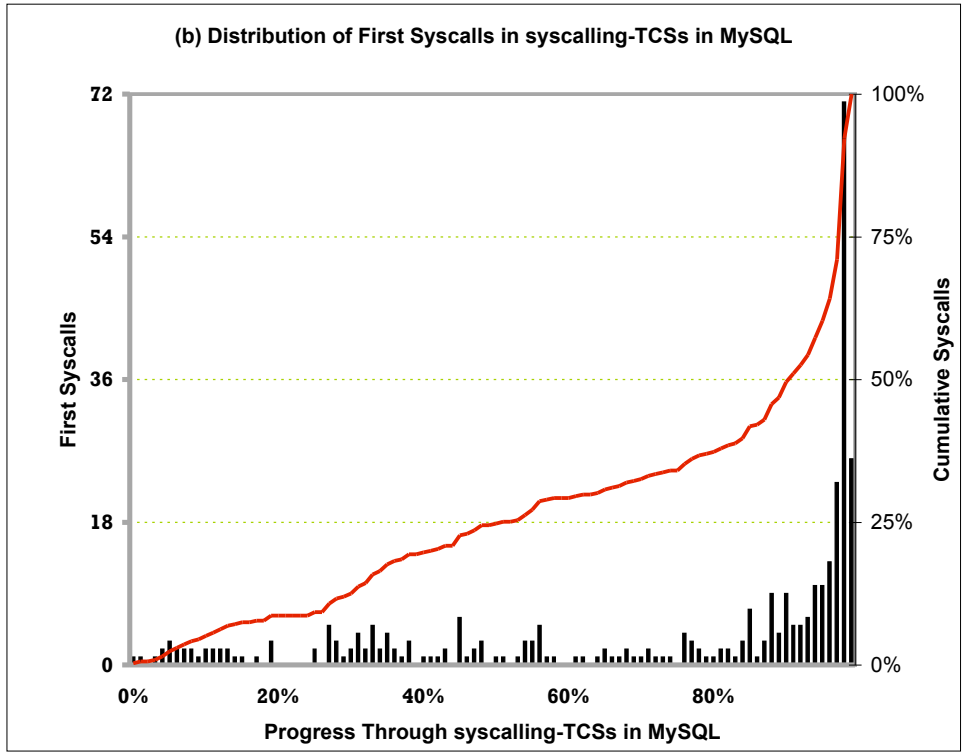




- Syscalls happen throughout their critical sections
  - Increased opportunity for intra-critsec dependence on syscalls

Results: Syscall Distribution





- First syscalls are also fairly distributed
  - If "*going nonspeculative*", serialized regions may be large

Implications for Existing TM I/O Proposals

- **Outlaw**: simply forbid any non-*protected* actions inside transaction.
- **Defer**: postpone side-effecting actions until commit
- "**Go Nonspeculative**": serialize side-effecting transactions
- **Compensate**: protect *unprotected* actions with compensation code

What does our data say about these?

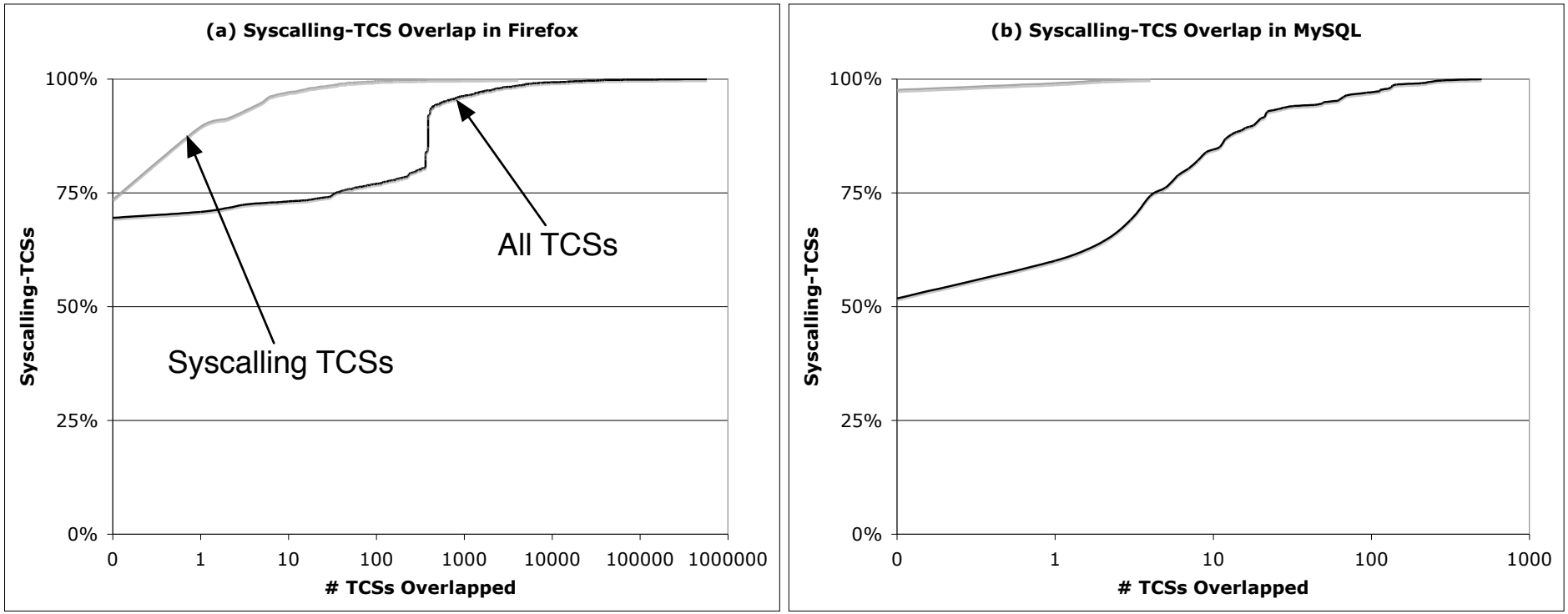
Results: The Applicability of Deferral

- We analyzed syscalling-TCSs responsible for 90% of the dynamic instances in our workloads:
  - Over 96% of those in MySQL, and 100% in Firefox, consumed the result of their first syscall
  - ➡ *Deferral* is not a general solution

Results: The Cost of "Going Nonspeculative"

- Two approaches: "commit lock" and "unkillable"
- We measured the *overlap* of syscalling-TCSs:
  - a syscalling-TCS *x* overlaps with all other TCSs which retire between *x*'s first syscall and its release
- We use this overlap to quantify the cost of "*going nonspeculative*"
  - Overlap represents the number of transactions which would like to retire but cannot

Results: "Going Nonspeculative"



- If "*going nonspeculative*" serializes *all* transactions, much parallelism is lost
- If it serializes only *syscalling* transactions, much less parallelism is lost

Conclusions

- Critical sections *do* have side effects in real code -- *outlawing* won't be trivial
- However, between correct *system-library-level compensation* code and a *transactional filesystem*, nearly all of the observed side effects can be handled speculatively, by protecting them at the library level
- *Deferring* side-effects until commit applies in only a minority of cases
- "*Going nonspeculative*" is not observed to be likely to affect performance, and could be a good choice if explicit aborts are not required
- No solution is a comprehensive answer!

Acknowledgements

- Thanks to Ravi Rajwar, who mentored an internship in which the basis of this research was performed
- This research was supported in part by NSF CAREER award CCR-03047260 and a gift from the Intel Corporation