Transactifying Apache's Cache Module

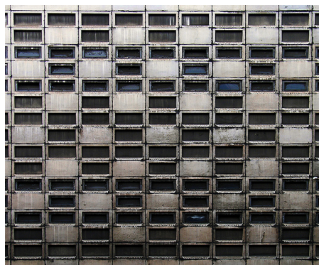
H. Eran O. Lutzky Z. Guz I. Keidar

Department of Electrical Engineering
Technion – Israel Institute of Technology

SYSTOR 2009 – The Israeli Experimental Systems Conference

Outline

Transactifying Apache's Cache Module

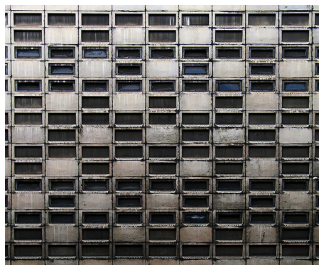


The shift to multicore machines challenges software developers to exploit parallelism. Transactional Memory is one approach to make this easier.

Our Goals:

- Transactifying a large-scale legacy application.
- Creating a performance evaluation method for STM systems.

Transactifying Apache's Cache Module



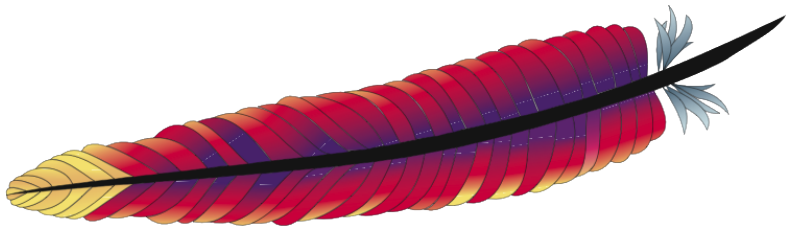
The shift to multicore machines challenges software developers to exploit parallelism. Transactional Memory is one approach to make this easier.

Our Goals:

- Transactifying a large-scale legacy application.
- Creating a performance evaluation method for STM systems.

Why Apache?

- Large-scale
- Popular
- Already parallel

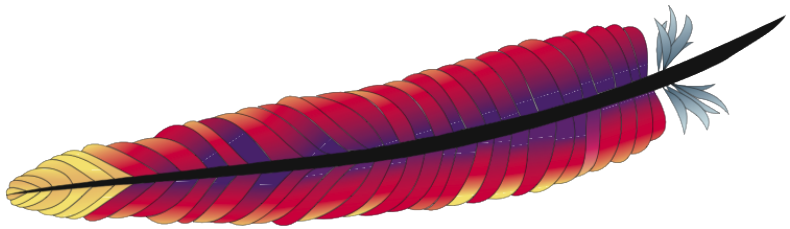


And why Apache's Cache module?

- One of the points of interaction between Apache's worker threads.
- Well encapsulated.
- Currently implemented using one big lock.

Why Apache?

- Large-scale
- Popular
- Already parallel



And why Apache's Cache module?

- One of the points of interaction between Apache's worker threads.
- Well encapsulated.
- Currently implemented using one big lock.

Previous work

- Concurrent data structures
  (e.g. red-black trees and skip lists.)
- STMBench7 – Measures operations on a more complex yet still artificial object graph.
- STAMP – Standford Transactional Applications for Multi-Processing:
  A collection of transactified scientific algorithms.

Which STM to use?

- Library-based or compiler-based.
- Tanger
  - Open source
  - LLVM compiler extension
  - Supports tinySTM and other STM systems.
- Intel STM Compiler
  - Experimental version of Intel's ICC
  - Proprietary STM system.
  - Has published ABI for other STM systems.

Which STM to use?

- Library-based or compiler-based.
- Tanger
  - Open source
  - LLVM compiler extension
  - Supports tinySTM and other STM systems.
- Intel STM Compiler
  - Experimental version of Intel's ICC
  - Proprietary STM system.
  - Has published ABI for other STM systems.

Which STM to use?

- Library-based or compiler-based.
- TANGER
  - Open source
  - LLVM compiler extension
  - Supports tinySTM and other STM systems.
- Intel STM Compiler
  - Experimental version of Intel's ICC
  - Proprietary STM system.
  - Has published ABI for other STM systems.

Which STM to use?

- Library-based or compiler-based.
- TANGER
  - Open source
  - LLVM compiler extension
  - Supports tinySTM and other STM systems.
- Intel STM Compiler
  - Experimental version of Intel's ICC
  - Proprietary STM system.
  - Has published ABI for other STM systems.

What to transactify

1. Convert mutex critical sections into transactions.
2. Wrapping atomic instructions inside transactions.
3. Decorate functions with Intel's `tm_callable` attribute.

Defining Atomic Blocks

Sometimes just converting a critical section in not optimal:

Get an object from the cache

1. Mutex lock
2. `obj` ← find key in cache
3. if `obj` found
   1. increment reference count on `obj`
   2. register `obj` for reference count decrementation when done.
4. Mutex unlock

Defining Atomic Blocks

Sometimes just converting a critical section in not optimal:

Get an object from the cache

1. Begin transaction
2. `obj` ← find key in cache
3. if `obj` found
   1. increment reference count on `obj`
   2. register `obj` for reference count decrementation when done.
4. End transaction

Defining Atomic Blocks

Sometimes just converting a critical section in not optimal:

Get an object from the cache

1. Begin transaction
2. `obj` ← find key in cache
3. if `obj` found
   1. increment reference count on `obj`
   2. register `obj` for reference count decrementation when done.
4. End transaction

Defining Atomic Blocks

Suppose we know cleanup won't happen too soon.

Get an object from the cache

1. Begin transaction
2. `obj` ← find key in cache
3. if `obj` found
   1. increment reference count on `obj`
4. End transaction
5. if `obj` found
   1. register `obj` for reference count decrementation when done.

Commit Handlers

- Pieces of code to be run on commit.
- Together with abort handlers allow for more efficient transactions.
- Can also be used in the shown scenario to clean up the code.

Handler Closurs

- Intel's commit handler syntax is:

```
_ITM_addUserCommitAction(transaction,
commitFunction, resumingTransactionId,
userArgument)
```

- Probably like any such construct in C.
- In languages that support closures, the use of commit handlers for our purpose would be much cleaner.

Evaluation

Client The *Siege* HTTP load testing tool.

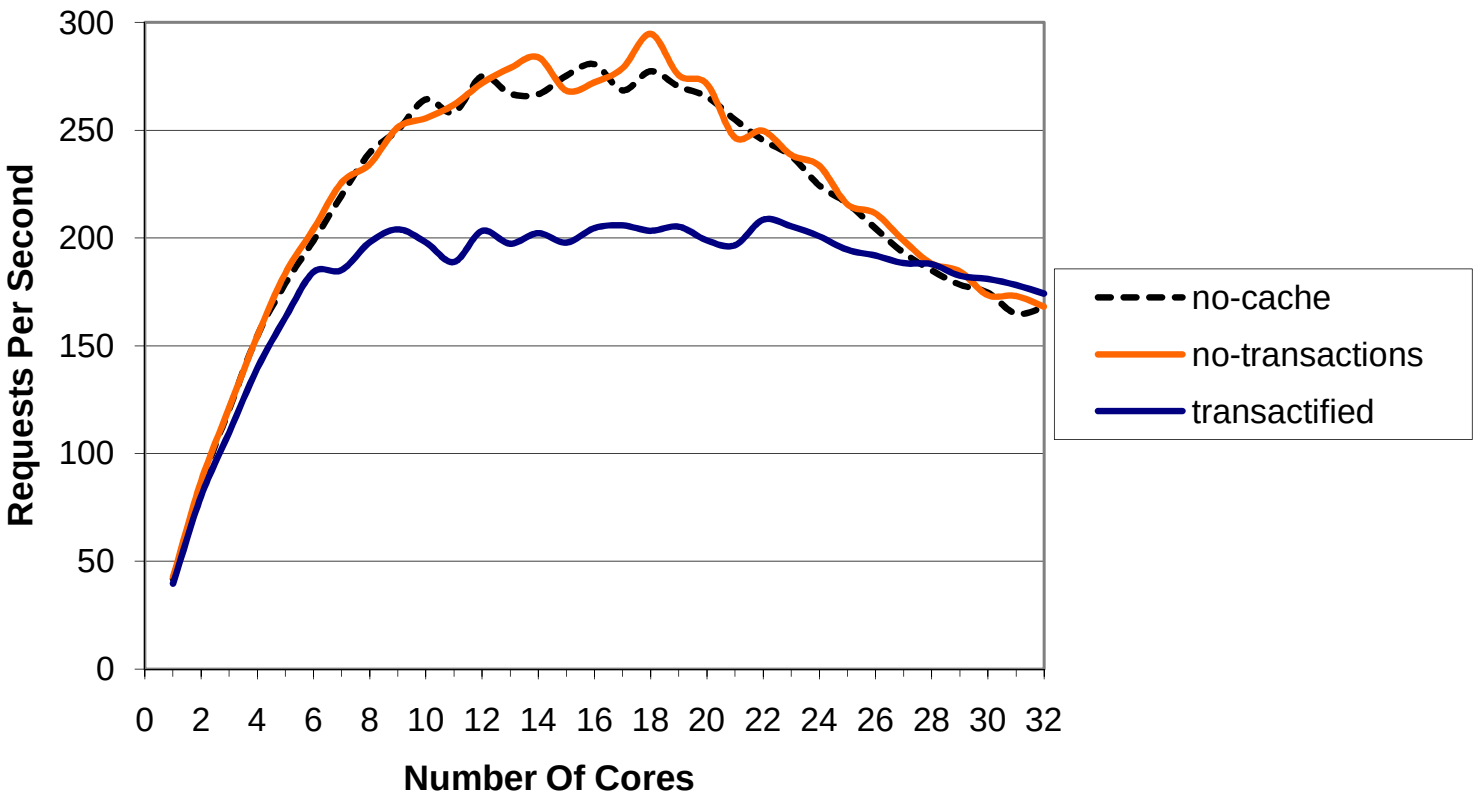
Workload The set of unix man-pages, served using the *man2html* CGI program. The program uncompressed the man-pages and rendered them to HTML.

Distribution Request files by Zipf distribution, whose s parameter determines the level of localilty in the requests.

Setup Two machines connected by Gigabit ethernet, each an 8-processors SMP with quard core 2.3GHz AMD Opteron processors and 126GB of RAM.

Results – Requests per Second

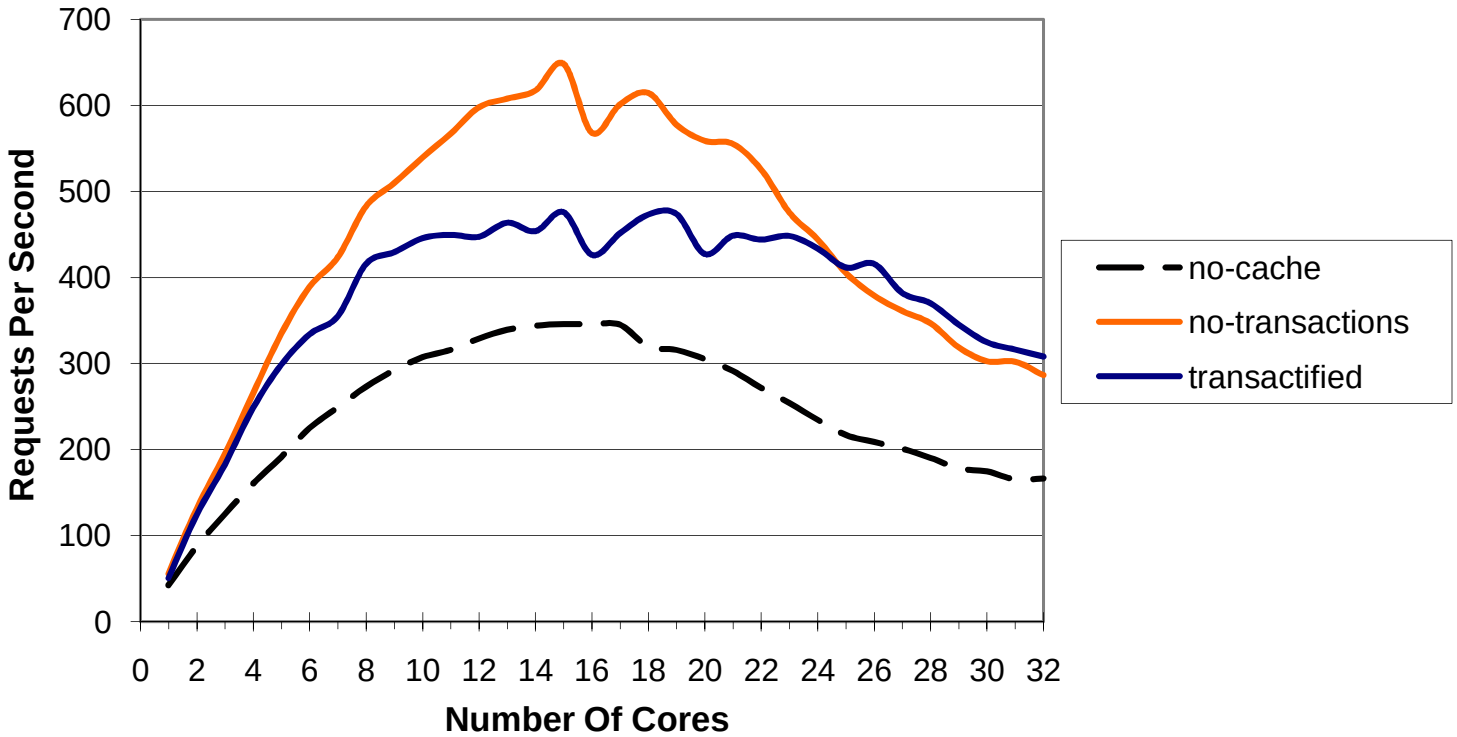
$s = 0.1$



Very low locality. Cache not effective. STM penalty high.

Results – Requests per Second

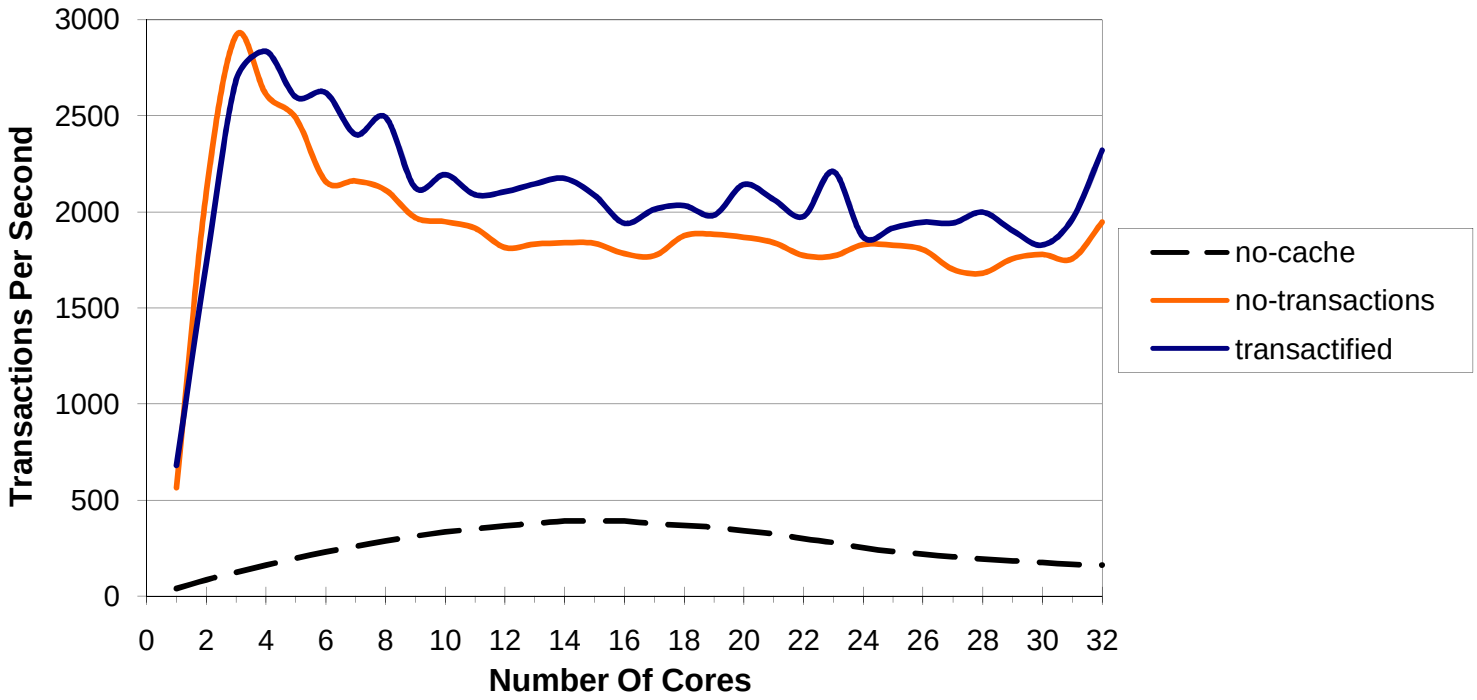
$s = 1$



Medium locality. Cache is an improvement. STM incurs penalty.

Results – Requests per Second

$s = 2$



High locality. Cache is vital. STM version works best.

Conclusion

- Encapsulation is important.
- Commit handlers might be useful not only for open transactions.
- Real-world applications are challenging and important to work on.

Questions?