Title Text

Subtitle Text, if any

Name1

Affiliation1
Email1

Name2   Name3

Affiliation2/3
Email2/3

Abstract

This is the text of the abstract.

Categories and Subject Descriptors   CR-number [*sub-category*]: third-level

General Terms   term1, term2

Keywords   keyword1, keyword2

1. Introduction

The text of the paper begins here.

2. Related Work

3. Transactifying Apache

Apache[apache] HTTP server is a popular web server application written in C. It supports working on multiprocessor machines with several multi-processing modules (MPMs) each offering a different strategy for handling requests and distributing the work. The most popular threaded MPM is the worker MPM, which works by running multiple worker-threads under several processes, each thread handles a single request at a time. In each such process there are several worker threads, and also a listener thread that fetches incoming requests and dispatches them to the available workers.

There are not many points of interaction between the worker threads themselves, where transactional memory can be used. One such place is Apache's memory cache implemented by the `mod_mem_cache`[apache:mod_mem_cache]

Figure 1. Apache worker MPM architecture

module. This module enables the workers of each process to share a cache of recently served requests. A new request can be served from the memory cache, and save the time required to access the disk and generate the requested page. Since the cache is shared between multiple threads, it is synchronized by a single lock, therefore a good candidate for converting into transactional memory.
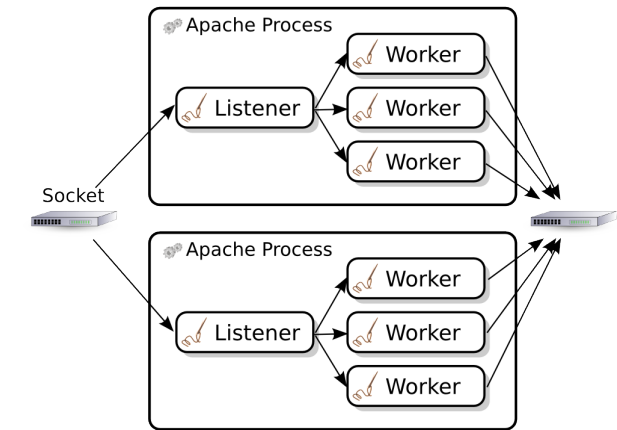
Apache's cache is implemented with a couple of modules. The first, `mod_cache`[apache:mod_cache], implements the logic related to caching. It tests the metadata of each requests to see if it can be supplied from the cache, according to the request's HTTP headers and the system configuration. It uses one of the underlying cache implementation modules, `mod_mem_cache` or `mod_disk_cache`[apache:mod_disk_cache] to do the actual caching.

The `mod_mem_cache` module implements a memory cache using a shared hash table and priority queue. The key to the hash table is the URL of the request,

converted into a canonical form. The cache is limited both by size and by the number of elements, and by memory size, so on insertion, sometimes lower priority entries are removed from the cache. The priority is deterimined by one of two algorithms: LRU, removing the least recently used entries first, and GDSF (Greedy Dual Size Frequency) assigning score to entries based on the cost of a cache miss, and the entry size.

3.1 C STM Systems

C and C++ STM systems divide into two kinds: Library based and compiler based. Library based STMs are built as a C library. Every transaction begins with a call into the library, and commits by another call. All reads and writes to global variables must be done through special library functions when in a transaction. This requires a great amount of work for converting an application to use STM. Not only accesses to global memory in the function that started the transaction must be converted, but also any access from any function being called from this function.

In contrast, compiler-based STM use a specialized compiler, which has extended syntax for transactional memory atomic blocks. The compiler can then automatically convert memory accesses inside transactions into calls to the underlying library, a process sometimes referred as *transactifycation*.

3.1.1 Tanger

The Tanger[felber2007tanger] transctifying compiler is an open-source academic compiler extension for LLVM[LLVM:CGO04], an extensible compiler framework. Tanger aims at creating a transactifying compiler that is independant of the STM system used. It works with the tinySTM[felber2008tinystm] library, but can easily be extended to use other STM libraries by writing a simple plugin.

Tanger created a transactified version of each function in a compilation unit. Every function call inside a transaction was then converted to a call to the new version. This method is a major disadvantage when working on a large application. Many functions do not need a transactified version and this causes uneeded work for the compiler and the linker. Moreover, sometimes the transactifyication might fail because of calls to functions whose source is not available and cannot be transactified. This can cause the entire build process to fail, where in fact the code can be transactified without any error.

Although tanger is accompanied with the Tarifa tool, which transactifies compiled binaries, and might have been a workaround for our problems, using it seemed an unnecessary complication.

This was the main reason why we eventually switched to Intel's STM Compiler. Lately a new release of tanger was announced, one in which the developer can annotate which functions should be transactified, however we didn't get a chance to try it.

3.1.2 Intel STM Compiler

Intel has published[icc] an experimental STM compiler based on their industrial compiler ICC. It solved the above problem by adding some new function attributes to the language that tell the compiler which functions need to be transactified. The attribute `tm_callable` tells the compiler that a transactified version of the function will be needed. This way only functions that are required inside transactions can be marked as `tm_callable` and be transactified.

Although ICC uses a proprietery STM manager, Intel has published their ABI[icc:abi] allowing for other STM managers to replace their own. This feature and its selective transactifycation ability were the main reasons why we preferred ICC.

An extension to the Gnu Compiler Collection (GCC) is being developed[gcctm] to enable transactional memory support for GCC. It is intended to work with tinySTM, but being open source, other STM systems will probably be ported too. The syntax of the C/C++ language extensions is designed to be compatible with ICC. This means that our work will probably be compilable under GCC with this extension, without much modification.

3.2 Transactifycation Process

The conversion process included converting critical sections protected by the cache module's lock into atomic blocks, and decorating required functions as `tm_callable`. The module had used atomic instructions for some memory accesses, and these were converted to full transactions in atomic blocks, so that collisions with these accesses will be detected.

Some transactions, after conversion contained code that belonged with the transactions, but didn't need neccessarily to run atomically with the transaction. An example might be a transaction removing an object from the cache, and freeing its memory. While the removal operation must be protected inside a transaction,

as it is using the shared memory structure of the cache, the memory release can happen any time later, since no other thread can point to the removed object after it had been removed from the cache.

For lock based systems, having the memory release as part of the critical section might cause a thread to hold the critical section a little longer than needed, but doesn't cause any problems other than that. On transactional memory systems, having accesses to other memory structures such as those required by memory management might cause collisions with other threads, thus slowing down the system in a similar way. In addition, the cleanup functions need to be transactified, which requires additional work both from the programmer and the compiler.

In our case, we chose not to transactify such functions, but instead remove them from the atomic section, and execute them after the transaction had committed. Although this requires some changes to the code, the changes are limited to the call-site, and need not modify any of the called libraries.

[TBD: add the section from lyx about commit handlers. add code of an example of this need.] [TBD: discuss other benefits of commit handlers, and mention the fact that Intel added them.]

4. Evaluation

4.1 Methodology

The transactified web server was evaluated using *Siege*[siege] an HTTP load testing tool. The server was loaded with the set of UNIX man-pages - a set of small textual files typical of some web sites. Each page was served using the *man2html*[man2html] program, uncompressed and converted into HTML, to make sure the serving of files requires enough computational resources to make the use of cache worthwhile.

The man2html program is a Common Gateway Interface (CGI) program that serves unix manual (man) pages on internet sites. The pages are usually stores compressed in gzip format, and formatted using the troff format. The program receives a request for a man page from the webserver, uncompresses the required file and converts it to HTML. As every CGI program it outputs the result with relevant HTTP headers.

The default caching policy of apache forbids caching dynamically generated pages such as those of man2html, unless the HTTP headers of the resulting page clearly specify otherwise. To make caching of the man2html

---

Figure 2. Average Response Time, $s = 1$

Figure 3. Average Response Time, $s = 2$

Figure 4. Transaction Rate, $s = 1$

Figure 5. Transaction Rate, $s = 2$

pages possible, we modified man2html to output such headers, specifying the output can be cached for one hour.

The pages were requested randomly according to Zipf distribution, whose parameter s determines how frequently the most popular pages were visited, thus controlling the amount of locality in the requests.

The experiments were done with two computers connected using Gigabit ethernet. The machine running the server was a 4 processors SMP of dual core 2.66GHz Xeons with 8GB of RAM, and the client machine was a 2 processors SMP of quad core 2.33Ghz E5410 Xeons with 8GB of RAM. [TBD: Update to neo and trinity]

4.2 Results

We compared the average latency and request throughput when running on different number of cores, and with different values. For every graph there are three experiments comparing the results of an Apache server running without a cache, a cached version without our transactional modifications, and the transactified version.

5. Conclusions

- Out of 340,000 lines of code in the Apache Web server, the cache module is comprised of only 6651 lines of code, of which only 273 lines were changed by our modifications. This shows the importance of being able to modify only encapsulated sections of the code, and interoperating with legacy code that still uses locks.

- Having commit handlers in the STM system is not only needed for creating efficient open transactions,
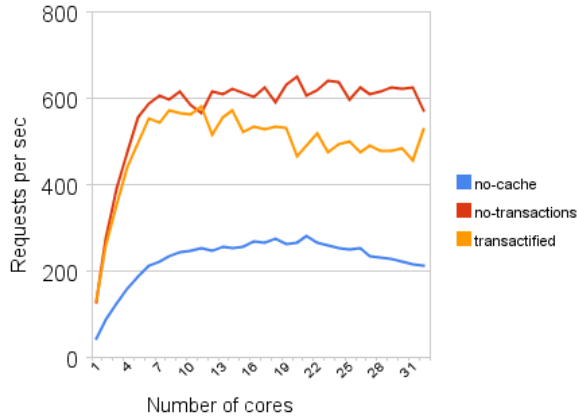
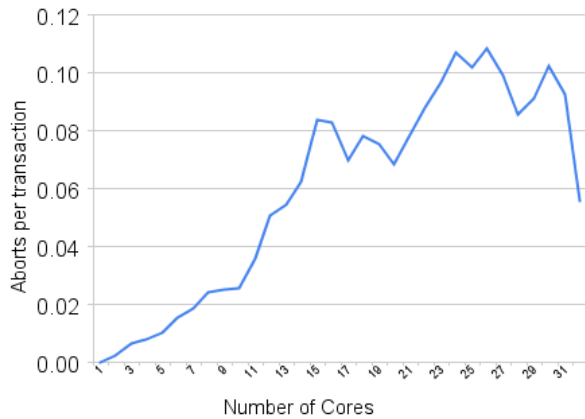**Figure 6.** Transaction Rate, $s = 1$, a single process

Figure 7. Abort Rate, $s = 1$, a single process

but can also aid the process of transactifying legacy code.

- There is great importance in working on real-world applications as they may reveal challenges resulting from engineering problems and not only algorithmic and theoretical problems.

There are many STM systems currently available, and one immediate direction would be to compare them using this benchmark. This would require writing plugins for any such system to match Intel's TM ABI.

In addition, there are other applications that might be interesting as transactional memory applications, following the methods we used.

A. Appendix Title

This is the text of the appendix, if you need one.

References

[1] Smith, P. Q. reference text