CCIT REPORT #718 February 2009

# Transactifying Apache

Haggai Eran Ohad Lutzky Zvika Guz Idit Keidar

Department of Electrical Engineering, Technion - Israel Institute of Technology

haggaie@tx.technion.ac.il lutzky@gmail.com zguz@tx.technion.ac.il idish@ee.technion.ac.il

## Abstract

Apache is a large-scale industrial multi-process and multi-threaded application, which uses lock-based synchronization. We report on our experience in modifying Apache to employ transactional memory instead of locks, a process we refer to as *transactification*; we are not aware of any previous efforts to transactify legacy software of such a large scale. Along the way, we learned some valuable lessons about which tools one should use, which parts of the code one should transactify and which are better left untouched, as well as on the intricacy of commit handlers. We also stumbled across weaknesses of existing software transactional memory (STM) toolkits, leading us to identify desirable features they are currently lacking. Finally, we present performance results from running Apache on a 32-core machine, showing that, there are scenarios where the performance of the STM-based version is close to that of the lock-based version. These results suggest that there are applications for which the overhead of using a software-only implementation of transactional memory is insignificant.

***Categories and Subject Descriptors*** D.1.3 [*Programming Techniques*]: Concurrent Programming

***General Terms*** Measurement, Performance, Experimentation

***Keywords*** Software Transactional Memory

## 1. Introduction

The vast shift to multi-core machines in recent years creates a major challenge for software developers, who must learn how to exploit the parallelism that such architectures can offer. In this context, *Transactional Memory (TM)* is one of the leading paradigms targeted at allowing programmers to easily harness the parallelism of future multi-core machines and to extract the performance promise these systems can offer. Since hardware transactional memory implementations are not yet in the market, *Software Transactional Memory (STM)* tools offer a viable alternative in the interim.

One of the principal challenges that TM systems confront, (besides delivering performance), is the ability to handle large-scale commercial applications. Developers that wish to employ TM, face the challenge of applying it to large legacy code. As TM systems are maturing, these aspects of convertibility and completeness are becoming critical in order to allow TM to shift from a promising concept to a full-fledged commercial tool. In this work, we try, via a design example, to answer how far we are from achieving these goals.

To date, TM was mostly employed within the niche of complex concurrent data structures, such as red-black trees and skip lists [9, 7], and isolated scientific algorithms (such as STAMP [3]); it was additionally used for benchmarks such as STMBench7 [8], which measures operations on a complex yet still artificial object graph. Moreover, Transactional Memory was mostly used thus far in benchmarks that were implemented explicitly with TM from the outset.

In this paper, we use Transactional Memory for the first time in the context of large-scale industrial software, which, moreover, does not pertain to any of the typical niches of transactional memory benchmarks. Furthermore, we convert legacy code, which used lock-based synchronization, to work with transactional memory, rather than write the benchmark from scratch. We refer to this conversion process as *transactification*. Specifically, we transactify the Apache web server. Since Apache is written in C, we needed to employ C-based STM toolkits. We next recognized that it would not be feasible to use *library-based* STM tools, as this would entail changing all reads and writes to global variables in the code. Instead, we opted for *compiler-based* STMs. We experimented with two such