Concurrent Programming Made Simple: The (r)evolution of Transactional Memory

Nuno Diegues*[1] and Torvald Riegel†[2]

[1]INESC-ID, Lisbon, Portugal
[2]Red Hat

October 1, 2013

*ndiegues@gsd.inesc-id.pt
†triegel@redhat.com

1 Harnessing Concurrency

Today, it is commonplace for developers to deal with concurrency in their applications. This reality has been driven by two ongoing revolutions in terms of hardware deployments. On one hand, processors have evolved to a multi-core paradigm in which computational power increases by increasing number of cores rather than by enhancing single thread performance. On the other hand, cloud computing has democratized the access to affordable large-scale distributed platforms.

In both cases programmers are faced with a similar problem: if they want to scale out their applications, then they need to tackle the issue of how to synchronize access to data in face of ever growing concurrency levels. For many decades, programmers have been taught to rely on locking mechanisms or centralized components to manage concurrent accesses to data. However, the ongoing architectural trends towards massively parallel/large-scale systems have unveiled the limitations of traditional synchronization schemes — not only can they significantly limit the feasible parallelism, when there may exist tremendous untapped parallel potential; they also force to use intricate programming models that are prone to tricky concurrency bugs, which can be a conundrum to detect and fix. In fact, popular knowledge considers locking approaches simple to understand, but difficult to master.

2 Programming with Transactions

To tackle this fundamental problem in modern software development, during recent years both industry and academia have started to adopt Transactional Memory (TM). With TM support, programmers only have to identify transactions in their code. The synchronization and coordination details are left to the underlying TM mechanisms, much in the same way that classic databases also hide the lock management from the user. By requiring programmers to only identify which code portions need to be executed atomically, and not how atomicity needs to be enforced (as with traditional locking), TM simplifies the development of synchronization in parallel applications.

This programming paradigm has been successfully implemented for both shared memory systems and cloud environments. In this talk we aim to clearly explain to developers why they should be using TM. Many of the attendees of FOSDEM will probably have heard of TM, but are not really sure what all the buzz is about — in particular, since TM has been used more often in languages such as Clojure or Scala. The take-away will be a clear view of how to use TM in different settings (multi-core and distributed cloud environments) as well as to which limit can we go for high performance in those concurrent settings. To do so, in this talk, we propose to address the latest developments of two recent European projects VELOX and CLOUD-TM. These projects brought together prestigious research centres and industry partners with the intent of creating **open-source** full-fledged TM stacks respectively for multi-core processors and cloud computing based on mainstream technologies.

2.1 The VELOX Stack

We have developed a fully functional stack for transactional memory within the EU-funded VELOX project [1]. This stack supports both software and hardware transactional memory. STM permits developers to use transactions on the programming language level before HTM becomes generally available and, later, to ship the same application software for computers that might have various levels of hardware support for transactions. The use of HTM permits to speed up these applications.

The VELOX TM stack consists of several components (see Figure 1): **Language extensions** provide new constructs with well-defined semantics for leveraging TM within program. The VELOX project focuses on two families of languages: C/C++ and Java. **Compiler support** is necessary for implementing an efficient TM stack in a way that is transparent to the programmer. The **TM runtime** is the central component of the TM integrated stack as it implements the synchronization logic of the TM. The VELOX project proposes several such libraries: purely software (C and Java), hardware, and hybrid. **Operating system** (OS) extensions can help improve the performance of TM for some tasks that relate to the system as a whole (e.g., scheduling), in particular because transactional workloads co-exist with traditional ones. Finally, the complete stack builds upon a **hardware platform** that provides the necessary support for designing efficient TM implementations.

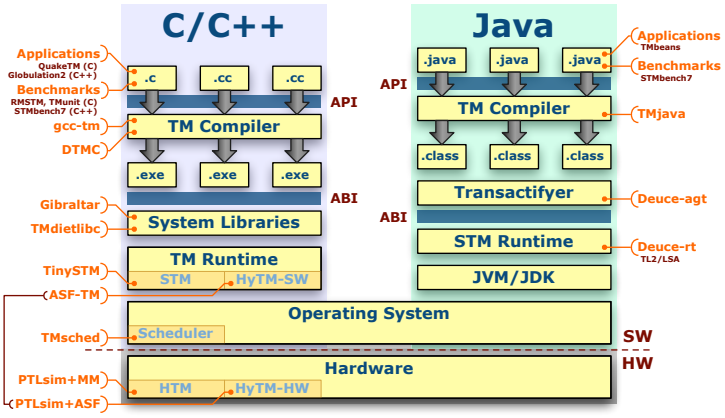

Figure 1: The VELOX stack supports two families of programming languages, C/C++ and Java, and the associated toolchains and libraries.

This most visible aspect of the stack for the end developer is the new language extensions for TM programming. In the context of VELOX, Red Hat and the other partners of the consortium have added TM support into GCC (available since version 4.7).

2.2 The CLOUD-TM Platform

The promise of infinite scalability has catalyzed much interest of late about cloud computing. However, the success of the cloud computing paradigm is menaced by one major pitfall: the lack of programming paradigms and abstractions capable of bringing the power of distributed programming into the hands of ordinary programmers, sheltering from the complexity of developing systems deployed over large scale, elastic cloud platforms.

The CLOUD-TM project [2] answers this urge by exposing a familiar Object-Oriented programming model on top of a scalable, fault-tolerant, and autonomic platform (see Figure 2). The key idea here is to allow developers to still program in a familiar environment, despite targeting the much more challenging distributed environment, which is enabled through one of our open-source projects called Fénix Framework[1]. With CLOUD-TM, programmers specify the structure of the application's domain model, using a domain-specific language (very similar to Java), and then develop the rest of the application in a language targeting the JVM. As a result, the domain of the application is made **transactional**, **distributed**, and **persisted**, in a transparent and automatic fashion. Yet, the programmer still only sees normal Java classes and, at most, has to define where transactions occur. On top of this we obtain the usual plethora of features ranging from indexes to different data-structures, all of that with strong transactional semantics for ease of programming.

To ensure high performance, on top of this simple and familiar development model, the platform incorporates a novel and highly scalable distributed TM integrated in Infinispan, an open-source distributed key-value store[2]. We have also improved this open-source project with runtime auto-configuration features, which is particularly useful since the programmer only interacts with Fénix Framework through the Object-Oriented model, and all the distribution of Infinispan is hidden from him.

[1]More information available at `http://fenix-framework.github.io/`
[2]More information available at `http://www.infinispan.org`

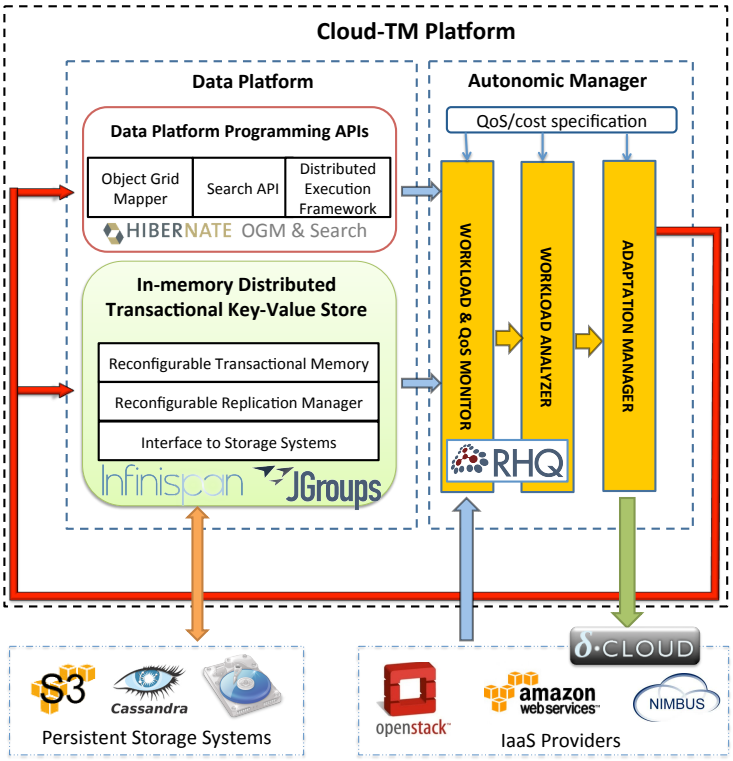

Figure 2: The Cloud-TM stack hides a complex, self-tuning Distributed Transactional Memory beneath a simple and intuitive object-oriented programming model integrated with JAVA and Ruby.

3 Structure of the talk

This talk will be presented by two speakers:

- Nuno Diegues (an academic researcher from INESC-ID, Portugal), who is pursuing his PhD in the area of Distributed TM for Cloud environments, and has been one the main contributors to the development of the Cloud-TM Programming API;
- Torvald Riegel (from Red Hat, Germany), who has focused his PhD on the design of TM for multi-core systems and has been deeply involved in the specification and implementation of C++ extensions for TM.

The talk will be subdivided in three parts:

1. A short introductory part overviewing Transactional Memory, its motivations, aims and key idea. Many programmers are already familiarized with the buzzword of Transactional Memory, but are not really acquainted with its purpose and usage.
2. Next, Torvald Riegel will present the transactional language constructs for C++ that are currently being specified in ISO C++ Study Group 5. For that, he will give an overview of GCC's support for these constructs, of libitm, and GCC's TM runtime library.
3. Finally, Nuno Diegues will present how to use Fénix Framework and Infinispan to develop transactional applications in the cloud. To do so, he will illustrate, with the help of code snippets, how it allows developers to fully exploit the computation capabilities of large-scale distributed systems in a simple and powerful, yet efficient way.

4 Short bios of the speakers

Nuno Diegues. Nuno Diegues has completed my BsC and MsC in Computer Science at Instituto Superior Técnico (IST), in Portugal. Currently he is a PhD student, also in Computer Science, at IST and the research laboratory INESC-ID. In his PhD work. Nuno Diegues is investigating the design of efficient large-scale Transactional Memory systems, where efficiency range from minimizing transaction abort rate (while ensuring strong consistent transactions) to energy savings. In the course of his work, he has published several contributions in the area of Transactional Memory in international Symposia.

Torvald Riegel holds a co-doctorate in Computer Science from Technische Universität Dresden, Germany and Université de Neuchâtel, Switzerland (2013), and a diploma in Computer Science from Technische Universität Dresden (2005). He works for Red Hat, focusing on parallelism and concurrency in Red Hat's toolchain team. He has authored more than a dozen scientific papers in peer-reviewed conferences and journals.

5 Acknowledgements

This talk has been selected and will be supported by the Euro-TM COST Action (http://www.euro-tm.org). Euro-TM is a pan-European research network bringing together top European scientists in the area of Transactional Memory, both from academy and industry. Euro-TM aims at consolidating European research on this important field, by coordinating the European research groups working on the development of complementary, interdisciplinary aspects of Transactional Memories, including theoretical foundations, algorithms, hardware and operating system support, language integration and development tools, and applications.

If the proposal is accepted, the speakers will be able to count on Euro-TM's financial support for what concerns travel and accommodation expenses.

References

[1] Y. Afek, U. Drepper, P. Felber, C. Fetzer, V. Gramoli, M. Hohmuth, E. Rivière, P. Stenström, O. Unsal, W. Maldonado, D. Harmanci, P. Marlier, S. Diestelhorst, M. Pohlack, A. Cristal, I. Hur, A. Dragojevic, R. Guerraoui, M. Kapalka, S. Tomic, G. Korland, N. Shavit, M. Nowack, and T. Riegel, "The Velox Transactional Memory Stack," *IEEE Micro*, vol. 30, no. 5, pp. 76–87, 2010.

[2] Cloud-TM Consortium, "Cloud-TM: A new programming paradigm for the Cloud," `http://www.cloudtm.eu`, 2013.