Versioned Transactional Shared Memory for the FénixEDU Web Application [*] [†]

Nuno Carvalho
INESC-ID/IST
nonius@gsd.inesc-id.pt

João Cachopo
INESC-ID/IST
joao.cachopo@inesc-id.pt

Luís Rodrigues
INESC-ID/IST
ler@ist.utl.pt

António Rito Silva
INESC-ID/IST
rito.silva@inesc-id.pt

Abstract

The FénixEDU system uses a novel infrastructure for web applications based on the Versioned Software Transactional Memory (VSTM) abstraction. The FénixEDU system has been deployed and is currently in operation in different facilities, including the Instituto Superior Técnico where it serves the entire academic community, processing between 1,000,000 and 4,500,000 transactions per day.

This paper describes the ongoing work on the infrastructure support, in order to increase its scalability and fault-tolerance. For that purpose we are developing a distributed version of the VSTM, such that multiple application servers can concurrently serve different request and still coordinate in an efficient manner to provide strong consistency guarantees to the applications.

1 Introduction

Web applications are a common solution for an increasing wide range of problems. In fact, solutions based on web applications are increasing not only in number, but also in the complexity of their underlying domain logic. Many of these applications manipulate information that is usually stored in relational database management systems (DBMS), which have been for a long time the main tool to store, manipulate, and maintain the integrity of data in information systems.

[*] This paper was partially supported by the GORDA (FP6-IST2-004758) and the Pastramy (PTDC/EIA/72405/2006) projects.

[†] Parts of this work were published in the Proceedings of the Second Workshop on Dependable Distributed Data Management (in conjunction with EuroSys 2008).

Even though web applications may be developed using many different technologies, the de-facto standard used in the industry for developing complex web applications relies on object-oriented platforms, such as the J2EE platform. These technologies store and access DBMSs, which offer a uniform approach to data integrity, durability, and availability.

Despite being widely used, the existing solutions have their drawbacks. On one side, the development of web applications using such complex platforms is time consuming and error prone. Several reasons contribute to the difficulty in the development of web applications when applying an enterprise architecture solution such as J2EE. Some of these reasons are: (*i*) the database cannot be ignored, biasing the development of an object-oriented rich domain model, and (*ii*) concurrent access to objects cannot be ignored, resulting either in error prone code that is difficult to debug, or in reduced performance, or both. On the other side, specialized web based systems are challenging the general purposefulness of a relational DBMS architecture. This is noticed even within the common systems usually based on relational databases, where the need for practical scale-out and near zero downtime translates to an increased need for cheap and efficient consistent replication and for shared-nothing clusters built on commodity hardware and software.

To overcome some of these problems, a new infrastructure for web applications that uses a Versioned Software Transactional Memory (VSTM) has been proposed in [2]. This new infrastructure provides many of the properties that are commonly supported by platforms such as the J2EE. Yet, it resulted in a much simplified programming model, when compared to the J2EE programming model. This new solution is in production since 2005 in the FénixEDU web application [15], which has more than 900,000 lines of code and is executing between 1,000,000 and 4,500,000 transactions per day. Since the shift to this new approach, a significant reduction in the number of bugs and an increase in the development agility was observed.

After some time using this approach in a production environment, new challenges were raised in terms of scalability. To cope with this increased usage, the VSTM solution should scale in terms of processing, memory, and concurrency, allowing the application to execute in several application servers simultaneously. The scalability quality, which is a central quality of J2EE-like architectures, requires that the VSTM approach be enriched with distribution, fault-tolerance, and persistence. This paper introduces a new architecture that should fit the new needs of the FénixEDU system, allowing it to scale in the number of application servers, and improving fault tolerance. At the same time, this architecture should not compromise the simplified programming model that allows programmers to easily develop new features.

This paper is organized as follows. The Section 2 introduces the FénixEDU system and the VSTM infrastructure, and motivates the need for this new architecture. The Section 3 introduces the proposed architecture. Finally, the Section 4 presents related work and the Section 5 concludes the work and points to future directions.

2 The FénixEDU system

The FénixEDU system is a web application that supports a wide range of academic activities in the IST campus (management of web pages for different courses, student enrollment, etc). The FénixEDU system started as a typical web application, with the application logic implemented in Java and its data stored in a relational DBMS. In this first version, denoted FénixEDUv1, it was used an Object/Relational mapping (ORM) tool to (almost) transparently store the objects in the database while maintaining the object oriented programming model to the programmers.

Over time, the system became a very large web application with a rich domain model. Currently, its domain model consists of more than 900 domain classes, each one implementing a distinct entity type of the FénixEDU domain. Unfortunately, the standard practices of persisting application data in a relational DBMS hinders the successful implementation of an object-oriented rich domain model. First, because accessing the data for a complex operation incurs into excessive round-trips to the external DBMS, thereby affecting the system's performance. Second, because rather than being a completely transparent aspect, relational-based persistence affects the programming model.

Both of these problems stem from the need to rely on the external DBMS to ensure the required transactional semantics for an operation. Thus, to solve these problems and benefit from all the expressiveness of an object-oriented rich domain model, the FénixEDU architecture was changed, such that the transactional semantics of an operation is supported by a Software Transactional Memory running at the application server.

2.1 Versioned STM

Mechanisms such as Java's synchronized keyword are useful to develop thread-safe single objects, but are of little help when various objects are involved in more complex operations. Ensuring, with lock-based mechanisms, that all the objects accessed during a complex operation are in a consistent state is difficult and highly error-prone. Much of the recent work on Software Transactional Memory (STM) [14, 8, 6, 7] deals with this problem by introducing into the programming language the notion of atomic actions, or transactions, which allow the atomic execution of a group of operations, thereby ensuring the consistency of the accessed data. The key idea underlying all the work on Software Transactional Memory is that programmers specify which operations should execute atomically, rather than protect accesses to the data with locks. The intended semantics for such actions is that they execute atomically, independently of which data is accessed by the operation.

We developed a new approach to STMs based on boxes that may hold multiple versions of their contents [3]. This Versioned Software Transactional Memory (VSTM) approach has the innovative aspect that it allows the execution of read-only transactions that never conflict with other concurrent transactions. The VSTM was implemented as a Java library – the JVSTM (Java Versioned Soft-

ware Transactional Memory) – and is used in the FénixEDU web application that we call the FénixEDUv2.

Most of the STM systems that have been built execute only in a single machine: i.e., object versions and object state is fully accessible locally by the STM runtime without the need to perform any sort of synchronization with other nodes. Unfortunately, this represents a serious limitation in terms of scalability. To overcome these limitations, the VSTM engine was augmented with support for distribution, by allowing to execute several instances of the application server. The current version that is being used in the production environment, denoted FénixEDUv2, uses a load balancer to distribute the client requests among several application servers. The application servers use a logically centralized DBMS to store the data. The DBMS is also used as a synchronization mechanism to maintain the cache of the servers consistent.

To the best of our knowledge, the FénixEDUv2 is the first distributed STM system being used in production today. Unfortunately, the current solution still relies on the access to a logically centralized database to enforce the global synchronization required to ensure the VSTM correctness. Thus, albeit more scalable than a centralized VSTM, the current solution still has many limitations to scalability.

2.2 The FénixEDU Workload

The major assumption underlying the development of the VSTM for the FénixEDUv2 system was that the number of read-only transactions vastly outnumbered the number of write transactions. Because of that, the main concern during the development of the VSTM was the performance of the read operations. Meanwhile, the FénixEDUv2 team instrumented the system to collect information about the system's workload. The results obtained thus far confirm the initial assumption and give us a greater insight on the nature of each transaction.

The number of read-only and write transactions successfully processed by the system was logged over a period of nine months (from October 2006 to June 2007), giving a total of slightly more than 186 millions of read-only transactions and 1.88 millions of write transactions. Thus, write transactions are approximately only 1% of the total number of transactions in the system. This ratio justifies the use of a versioned STM that requires synchronization only for the write transactions.

Given that a distributed version of the VSTM requires some form of communication between the nodes of the system, it is important to know the sizes of the read-set and of the write-set of each transaction. We collected this information in the FénixEDUv2 system over a period of two weeks, and we show a summary of the results obtained in Table 1. These numbers show that the write-set is much smaller than the read-set: Considering only write transactions, the average read-set is 1,349 times larger than the average write-set. This suggests that the solution for a distributed VSTM should rely on the communication of the write-set only, if possible.

	Read/Write-set size	
	Average	Maximum
Read-set of read-only transactions	5,844	63,746,562
Read-set of write transactions	47,226	2,292,625
Write-set of write transactions	35	32,340

Table 1: Average and maximum read-set and write-set sizes for each kind of transaction in the FénixEDUv2.

3 Towards a Distributed VSTM

The current FénixEDU architecture requires each application server to access the database every time it starts a new transaction to check whether its cache is still up-to-date. Yet, the cache needs to be updated only when another application server in the cluster commits a write transaction. Given the ratio of read/write transactions for the FénixEDU system, this means that only 1 out of 100 transactions would actually need to go to the database. To address this inefficiency, we are currently working on a distributed version of the VSTM that uses group communication to update all the nodes in the cluster when a write transaction commits.

To cope with the new challenges, we propose a new architecture that does not rely on the database to store and to synchronize transactions. Instead, transactions are executed in application servers and committed in a distributed and transactional shared memory system. This system uses group communication and an atomic broadcast primitive to order transactions and maintain a consistent state. The proposed architecture is denoted FénixEDUv3 and is depicted in the Figure 1, including the details of the Application Server.

The system is composed by application servers that access a common (possibly replicated) persistent storage. Each application server runs the following components: a *Request processor* (RP), responsible for receiving the requests and starting the VSTM transactions; a *VSTM* executes the transactions by running a protocol described in the next paragraphs; a *Cache manager* (CM), responsible for maintaining a consistent cache of regularly used objects; a *Persistent store* (PS), responsible for persisting the updates on stable storage for later recovery; a *Certification manager* (Cert), that certifies the transactions ready to commit to see if there are conflicts; a *Group communication service* (GC), responsible for maintaining up-to-date information regarding the membership of the group of application servers (including failure detection) and providing the required communication support for the coordination among the servers. In particular, the service provides both an *Atomic broadcast service* used by the VSTM to disseminate the write-sets of transactions that are ready to commit, and a non-ordered *Reliable broadcast service* used by an application server to notify the other application servers about commit/abort decisions of transactions.

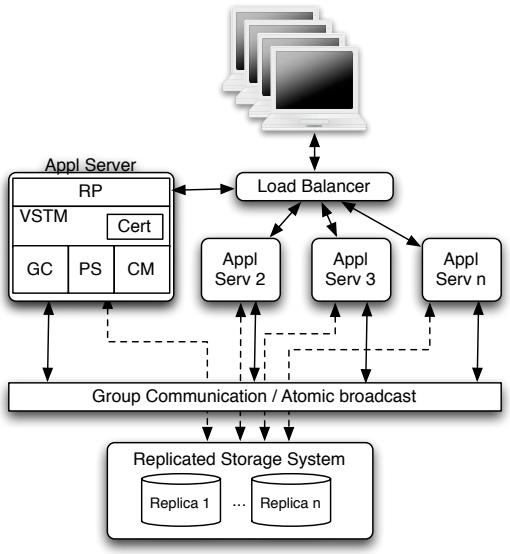

Figure 1: The proposed architecture.

The algorithm works as follows. The application server receives requests from clients (typically, web browsers) via a load balancer and optimistically locally executes a transaction. To execute the transaction, the application server may have to retrieve some objects from the stable storage to fill its cache. Upon commit, the write-set of the transaction (i.e., the set of objects whose value was changed by the transaction) is disseminated to all the other nodes using atomic broadcast. The nodes that receive the write-set should mark their copies of the objects as potentially obsolete. The local node certifies the transaction and decides on its outcome (commit or abort); a write transaction can commit if its read-set has not been updated by other transactions serialized in the past. After the certification procedure, a notification of commit/abort is sent to all nodes using a group communication primitive. If the transaction can commit, all nodes update their version of the objects, otherwise the updates are discarded.

Note that read transactions never abort, since the properties of the VSTM are kept. This algorithm also ensures that all replicas of the application server have up-to-date copies of the objects on their cache. This avoids retrieving an object from the storage system when it is already in the cache, just to check if the cache contains the latest version. If a write transaction aborts, it can be resubmitted later. The consistency of the replicas is ensured by the ordering imposed by the atomic broadcast primitive, which avoids the synchronization of application servers via a central node.

The commit procedure includes the steps required to make data persistent (by accessing the persistent store). Although this step is required in Fénix-

EDUv3, the technologies used to make the state persistent are orthogonal to the mechanisms used to distribute the STM. The reader may note that several existing technologies can be used, including the traditional DBMS, a file system, or Java based object persistence technologies. Naturally, the persistent store will also include fault-tolerant and high-availability mechanisms, such as the ones described in [5, 4]. Due to lack of space, storage systems are not addressed further in this paper.

4 Related work

Several different solutions have been proposed to add support for replication and fault-tolerance in the context of complex web applications. The most common approach relies on using a relational database to implement the application domain model. Replication solutions for relational databases have been extensively studied in the literature, and mature solutions are now available such as primary-backup [11], state-machine [12], or certification based [10] replication. However, for efficiency, our work requires the replication of the volatile state as well as the persistent state, thus solutions for database replication are not enough to reach our goals.

One way to implement a web application domain model is using an object-oriented architecture. In most systems, the current approach to implement it is to use a multi-tier J2EE architecture where the business logic and data are modeled using Enterprise Java Beans, being the data stored in databases by means of an Object/Relational mapping tool. The solution proposed in [13] presents a way to replicate such systems by adding fault tolerance mechanisms on both the application server and the database. The authors rely on a locking based approach and propose a replication protocol based on Snapshot Isolation. Our solution avoids explicit locking by means of a STM and proposes a replication protocol that provides Serializability.

In the scope of object oriented models, most software transactional memory systems have been implemented assuming a single (central) server. Distributed solutions need to address the new challenges that emerge in a environment where processes can fail and recover. A distributed version of an STM should leverage on the expertise accumulated in the design and implementation of Distributed Shared Memory (DSM) systems. For instance, Sinfonia [1] is a recent DSM system that has been shown to scale with the number of servers even if it uses two phase commit to maintain memory consistency. However, Sinfonia is word based and applications must know the memory location (nodes and address) of the data that needs to be accessed. Object-based DSM systems also exist. One such example is Terracotta [9] which replicates objects in a set of servers, offering a transparent location of objects to the applications. On the other hand, our approach is tightly integrated with the VSTM system and the updates are disseminated and validated in a way that preserves the VSTM consistency model.

5 Conclusions and future work

This paper introduces the new architecture for the FénixEDU system that we are currently implementing. The FénixEDU system captures all the requirements of complex web applications that are becoming more and more common today. Its implementation is based on a Versioned Software Transactional Memory system that frees the programmers from coding concurrency control explicitly. The introduction of VSTM in the FénixEDU system has significantly improved the productivity of the development team and the quality of the resulting code. This approach helps to build complex web application systems in a more effective way, but lacks some important properties to cope with scalability and fault tolerance. The architecture described here is aimed at addressing these problems.

The choice of the best technology to make data persistent is still an open issue. Given that a large majority of the transactions in FénixEDU are read-only transactions, we should favor very fast read operations (to update the cache of application servers). This suggests that the persistent copy of data may also be replicated in a way that allows (consistent) concurrent reads from multiple repositories.

References

[1] M. K. Aguilera, A. Merchant, M. Shah, A. Veitch, and C. Karamanolis. Sinfonia: a new paradigm for building scalable distributed systems. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 159–174, New York, NY, USA, 2007. ACM.

[2] J. Cachopo and A. Rito-Silva. Combining software transactional memory with a domain modeling language to simplify web application development. In *Proceedings of the 6th International Conference on Web Engineering*, pages 297–304. ACM Press, July 2006.

[3] J. Cachopo and A. Rito-Silva. Versioned boxes as the basis for memory transactions. *Science of Computer Programming*, 63(2):172–185, Dec. 2006.

[4] Continuent. Sequoia v3.0. http://sequoia.continuent.org, 2006.

[5] A. Correia Jr., J. Pereira, L. Rodrigues, N. Carvalho, R. Vilaca, R. Oliveira, and S. Guedes. Gorda: An open architecture for database replication. In *Proceedings of the 6th IEEE International Symposium on Network Computing and Applications (IEEE NCA07)*, pages 287–290, Cambridge, MA, USA, July 2007.

[6] T. Harris and K. Fraser. Language support for lightweight transactions. In *Proceedings of the 18th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, volume 36 of *SIGPLAN Notices*, pages 388–402. ACM Press, Oct. 2003.

[7] T. Harris, S. Marlowe, S. Peyton-Jones, and M. Herlihy. Composable memory transactions. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM Press, June 2005.

[8] M. Herlihy, V. Luchangco, M. Moir, and W. N. Scherer, III. Software transactional memory for dynamic-sized data structures. In *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing*, pages 92–101. ACM Press, July 2003.

[9] T. Inc. Terracotta. http://www.terracotta.org/.

[10] Y. Lin, B. Kemme, M. Patino-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 419–430, New York, NY, USA, 2005. ACM.

[11] S. Mullender. *Distributed Systems*. ACM Press, 1989.

[12] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. In *Journal of Distributed and Parallel Databases and Technology*, 2003.

[13] F. Perez-Sorrosal, M. Patio-Martnez, R. Jimnez-Peris, and B. Kemme. Consistent and scalable cache replication for multi-tier j2ee applications. In *Middleware*, volume 4834 of *Lecture Notes in Computer Science*, pages 328–347. Springer, 2007.

[14] N. Shavit and D. Touitou. Software transactional memory. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, pages 204–213. ACM Press, Aug. 1995.

[15] I. S. Tecnico. Fenix project. https://fenix-ashes.ist.utl.pt/.