Bridling Concurrency to Boost Performance in Distributed STMs

Nuno Carvalho, Paolo Romano, Luís Rodrigues
INESC-ID/IST
Lisbon, Portugal
Email:{nonius, romanop}@gsd.inesc-id.pt, ler@ist.utl.pt

I. Introduction

Over the last years, a wide body of literature has been developed in the area of Software Transactional Memory (STM) systems and, recently, the first STM-based applications have started to be deployed in production systems [1]. One of the key lessons learnt from the development and deployment of these applications is that existing STM platforms suffer from a significant limitation: the lack of efficient replication schemes capable of fulfilling the scalability and reliability levels of real-world applications [1].

Replication of STM systems represents a very recent research field, and, at current date, only a few solutions have been proposed and evaluated [2]. On the other hand, since STMs and databases share the common notion of atomic transaction, the extensive research developed in the area of replicated databases represents a natural source of inspiration for the design of replication schemes for STMs. Among the plethora of database replication schemes, recent approaches are based on Atomic Broadcast (AB) [3] and a distributed certification procedure [4]. These schemes ensure the consistency of replicas only at commit-time via a distributed certification phase that uses a single AB to enforce agreement on a common transaction serialization order. This avoids distributed deadlocks, and provides non-blocking guarantees in the presence of (a minority of) replica failures. Unfortunately, the overhead of previously published AB based certification schemes can be particularly detrimental in STM environments [5].

Further, distributed certification schemes are based on an inherently optimistic approach: transactions are only validated at commit time and no bound is provided on the number of times that a transaction will have to be re-executed due to the occurrence of a remote conflict. This can lead to undesirably high abort rates in high conflict scenarios. Also, when considering heterogeneous workloads containing mixes of short and long-running transactions (as it is actually the case for several well-known TM benchmarks [6]), the latter ones may be constantly aborted due to the occurrence of (remote) conflicts with a stream of short-lived transactions.

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and through project "Pastramy" (PTDC/EIA/72405/2006).

In this paper we briefly announce the novel Asynchronous Lease Certification (ALC) scheme to tackle the above issues. In the core of the ALC scheme is the notion of *asynchronous lease*. Analogously to classic lease schemes [7], asynchronous leases are used by a replica to establish temporary privileges in the management of a subset of the replicated dataset. Unlike classic lease based approaches, where the lease duration is defined at the time of the lease establishment, in ALC leases are said to be asynchronous since that the concept of lease is detached from the notion of time. Conversely, once a replica acquires a lease on a set of data items, it holds the lease as long as there is no explicit lease request from another replica.

II. Asynchronous Lease Certification

In the ALC replication scheme, the replica starts by executing locally the transaction. When the commit process is started, and before it can be safely certified, the replica must hold the leases for the data set of the committing transaction. If the replica does not have all the leases, it must request them through an AB, which ensures a global serialization order for the lease ownership. By disseminating lease requests via AB and atomically enqueueing them at each node in the AB-delivery order, distributed deadlocks are avoided. Fairness is ensured by establishing leases in FIFO order and leases are transferred to a requesting replica as soon as the transactions (in execution at the lease-owner replica) to which those leases had been granted have committed. Finally, we rely on Virtual Synchrony [3] with non partitionable views to free the leases owned by failed or unreachable nodes.

The ownership of an asynchronous lease on a set of data items provides a replica with several key benefits. First of all, it reduces the commit phase latency of the transactions that access a given set of data items. The ownership of the lease, in fact, ensures that no other replica will be allowed to validate any conflicting transaction, making it unnecessary to enforce distributed agreement on the global transactional serialization order. ALC takes advantage of this by limiting the use of AB exclusively for establishing a global order for the lease ownership. Subsequently, as long as the lease is owned by the replica, transactions can be locally validated and their updates can be disseminated using a Uniform Reliable Broadcast (URB) [3] primitive, which

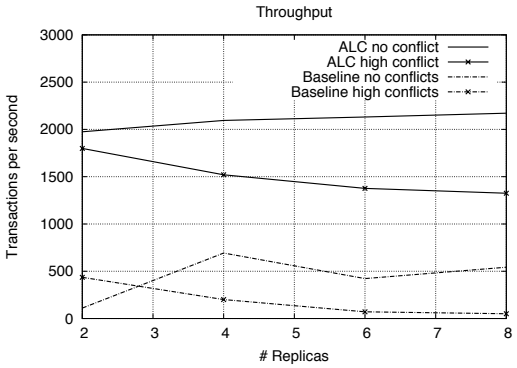

Figure 1. Throughput of the ALC scheme.

can be implemented in a much more efficient manner than AB. Secondly, the ALC shelters transactions from repeated abortions due to remote conflicts. With ALC, analogously to classic certification schemes, transactions run based on local data, avoiding any inter-replica synchronization until they enter the commit phase. At this stage, however, ALC ensures to have established a lease for the accessed data items, prior to proceed with the transactions' validation. In case a transaction T is found to have accessed stale data, T is re-executed without releasing the lease. This ensures that, during T's re-execution, no other replica can update any of the data items accessed during the first execution of T. This guarantees the absence of remote conflicts on the subsequent re-execution of T provided that it deterministically accesses the same set of data items accessed during its first execution, as it is typically the case with realistic applications. This bounds the maximum abort rate to 50%.

III. Preliminary Experimental Results

We now report preliminary results of a experimental study aimed at evaluating the performance gains achievable by the ALC certification scheme in a real distributed STM system, namely when using our prototype, in face of a STM benchmark. The target platform for these experiments is a cluster of 8 nodes, each one equipped with an Intel Quad-Core Q6600 at 2.40GHz and 8 GB of RAM running Linux 2.6.27.7 and interconnected via a private Gigabit Ethernet. We used a synthetic workload (obtained by adapting the Bank Benchmark originally used in [8]) which serves for the sole purpose of validating the ALC scheme in two extreme scenarios for what concerns conflicts. In detail, we initialize the STM at each replica with a vector of $numMachines \cdot 2$ items. In the first scenario, each machine reads and updates a distinct fragment of the array, thus never conflicting. In the second scenario, each machine reads and updates the same data items, thus always conflicting. We compare the ALC with a baseline certification scheme [4].

The Figure 1 depicts the throughput (committed transactions per second) of the replicated system. In the scenario with no conflicts, the ALC scheme is able to maintain a stable throughput as the number of replicas increase and is able to achieve a throughput 3 times higher than the baseline certification scheme, thanks to the ALC lease establishment: once the lease owners are established, the several replicas can concurrently send the updates to the other replicas using URB. On the other hand, the baseline must send an AB for each committing transaction. In the high conflict scenario, the throughput of the baseline scheme degrades as the number of replicas increases. This is because with more replicas, the conflicts cause a very high abort rate (up to 90% for 8 replicas). ALC holds the lease for a transaction that aborts and accesses the same data items when restarted, ensuring that each transaction never aborts more than once.

Even though these experimental results have been obtained based on very simple synthetic workloads, they do highlight the potentialities of ALC in significantly boosting performance of a classical certification scheme.

References

[1] N. Carvalho, J. Cachopo, L. Rodrigues, and A. Rito Silva, "Versioned transactional shared memory for the FenixEDU web application," in *Proc. of the Workshop on Dependable Distributed Data Management (WDDDM)*. ACM, 2008.

[2] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues, "D2stm: Dependable distributed software transactional memory," in *Proceedings of the 15th Pacific Rim International Symposium on Dependable Computing (PRDC 09)*, Shanghai, China, Nov. 2009.

[3] R. Guerraoui and L. Rodrigues, *Introduction to Reliable Distributed Programming*. Springer, 2006.

[4] F. Pedone, R. Guerraoui, and A. Schiper, "The database state machine approach," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 71–98, 2003.

[5] P. Romano, N. Carvalho, and L. Rodrigues, "Towards distributed software transactional memory systems," in *Proc. of the Workshop on Large-Scale Distributed Systems and Middleware (LADIS)*, 2008.

[6] M. Ansari, C. Kotselidis, I. Watson, C. Kirkham, M. Luján, and K. Jarvis, "Lee-tm: A non-trivial benchmark suite for transactional memory," in *ICA3PP '08: Proceedings of the 8th international conference on Algorithms and Architectures for Parallel Processing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 196–207.

[7] C. Gray and D. Cheriton, "Leases: an efficient fault-tolerant mechanism for distributed file cache consistency," in *SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 1989, pp. 202–210.

[8] M. Herlihy, V. Luchangco, and M. Moir, "A flexible framework for implementing software transactional memory," *SIGPLAN Not.*, vol. 41, no. 10, pp. 253–262, 2006.