A Formal Model of Crash Recovery in Distributed Software Transactional Memory (Extended Abstract)

Paweł T. Wojciechowski, Jan Kończak
Poznań University of Technology
60-965 Poznań, Poland
{Pawel.T.Wojciechowski,Jan.Konczak}@cs.put.edu.pl

Abstract

In this paper, we describe our ongoing work on efficient recovery of distributed software transactional memory (DSTM) after machine crashes. We focus on a system model in which transaction objects are replicated on servers, and normally stored in a volatile memory only. On every server, an atomic transaction can locally read and write a set of objects atomically. DSTM systems maintain consistency of object replicas by coordinating object accesses, so that one-copy serializability is guaranteed despite failures such as server crashes. After crash, a server can rejoin the system and catch up on the current state. Our work is aimed at DSTM systems in which object replication is based on deferred update relying on atomic broadcast.

We develop an analytical model of such class of DSTM systems with support of crash recovery using various recovery schemes. The model takes into account multicore concurrency, transaction conflicts, catching up on the current state using object snapshots and a log of transaction operations, and optimizations such as multiversioning. The model formalizes the upper and lower bounds on DSTM operations, and on replica recovery after a machine crash. We used the model to propose various strategies for obtaining object snapshots and catching up on the current state, taking into account transaction processing operations, the length of transactions, the size of objects, and the type of workload. In this paper, we discuss our motivations, goals, and summarize ongoing work.

1 Introduction

Replication is an important method to increase service reliability and accessibility. It means deployment of a service on several server machines, each of which may fail independently, and coordination of service replicas so that they maintain a consistent state. Each replica maintains service state in a volatile memory (and optionally in its local nonvolatile store). In this paper, we focus on *distributed software transactional memory (DSTM)*—a novel enabling technology for service replication. The programmers use atomic transactions to access data declared as transactional. These data are replicated on every server. DSTM guarantees that any modifications of data replicas are consistently propagated to every server. Moreover, fault-tolerant DSTM systems can provide continuous service despite of failures. DSTM is a follow-up of the work on *software transactional memory (STM)*—a concurrent programming mechanism intended to replace lock-based synchronization (see [1, 2] among others). Both classes of systems offer programming constructs for transaction creation, abort, and retry, where an *atomic transaction* is any piece of code containing memory reads and writes that must be executed atomically. However, some operations may be forbidden, e.g. I/O operations and irrevocable system calls. To avoid blocking, STM systems use an optimistic concurrency control scheme—a transaction that conflicts with another concurrent transaction is rolled back and retried. In some implementations a mixture of optimistic and pessimistic concurrency control is used. DSTM is essentially like STM but transactional memory is replicated on several network nodes.

In this paper, we describe our ongoing work on efficient recovery of distributed software transactional memory (DSTM) after machine crashes. To experiment with the recovery strategies in DSTM we develop Paxos STM [3, 4]—one of the first fault-tolerant distributed STM systems. The application programming interface has constructs for transaction creation, abort, retry, and for annotating classes as transactional. Concurrent execution of atomic transactions satisfies one-copy serializability [5]. The system is object-based, i.e., the smallest unit shared by transactions is an object. Paxos STM implements the deferred update replication scheme. It is a variant of the primary-copy replication [6] that allows many concurrent master replicas, also called multi-primary passive replication [7] or an eager, update everywhere replication [8]. Paxos STM uses a non-blocking transaction certification (or commitment) protocol which is based on *atomic broadcast (abcast).* Several authors demonstrated the advantages of abcast in the implementation of replicated databases (see e.g., [9, 10]). This approach prevents deadlocks and allows better scalability than using two-phase commitment since transactions are never blocked [11, 8]. The abcast module in Paxos STM is based on an efficient implementation of the Paxos [12] distributed consensus algorithm from JPaxos [13].

JPaxos [13] is an efficient state-machine-based replication tool that we developed in collaboration with EPFL. The goal of the JPaxos project is to investigate engineering issues in the implementation of Paxos, such as various optimization techniques like tuning Paxos for high-throughput with batching and pipelining [14], and support of the crash-recovery model of failure with a minimal use of stable storage [13]. By reusing the implementation of Paxos in Paxos STM, we can experiment with the crash-recovery strategies which are tailored for DSTM. In our system, a server replica can recover after a crash and catch up on the current state automatically (from a local stable storage and other replicas). We have developed several crash-recovery algorithms. Depending on the choice of the algorithm, we can achieve a different trade-off between fault tolerance and performance. For example, by choosing the most efficient algorithms, the system can tolerate failure of up to $\lceil N/2 \rceil - 1$ server crashes at a time, where N is the number of servers. Alternatively, the system can also tolerate catastrophic failures (up to N servers crashed at a time) but this results in a dramatic performance drop. A server can rejoin the system any time after crash.

In Section 2, we define the crash recovery process that we consider. Then, we discuss our motivations, goals, and research problems in Section 3. Next, we describe preliminary results on a formal model of crash recovery in DSTM in Section 4.

2 Crash Recovery

We assume that data stored to a non-volatile memory (or stable storage), such as disks, can survive the crash, while data stored in volatile memory (RAM and processor caches) are lost after a crash. However, operating systems usually buffer data before writing them to stable storage (asynchronous writes)—if a crash occurs some data may be lost. On the other hand, writing to stable storage synchronously is very slow. Thus, our goal is to minimize the amount of synchronous writes in DSTM and/or execute them in parallel with other DSTM operations whenever possible[1].

In the crash-stop model of failure, crashed processes will never be up again. Hence, no recovery phase and stable storage are required. Since processes do not do any synchronous writes to stable storage, DSTM performance can be high. However, there is one significant drawback. Paxos algorithm can decide on a value (and continue working) only if majority of processes are correct. If majority of processes will crash, the algorithm will never decide again and DSTM will stop forever. Because of that using this model is not very practical. Instead, we adopted the crash-recovery model which assumes that machines that crashed will be up again and rejoin the system.

We can consider various recovery schemes that range from a very inefficient algorithm that uses a non-volatile memory to store synchronously a snapshot of transaction objects, up to efficient algorithms that use non-volatile memory only for data which are necessary for DSTM to work correctly. The choice is the trade-off between performance versus tolerance of catastrophic failures (crash of all servers). JPaxos currently implements three example crash recovery algorithms designed for Paxos: crash recovery with stable storage, epoch-based recovery, and view-based recovery. The first algorithm tolerates catastrophic failures but is very inefficient. The other two algorithms are efficient but they

[1]The new generation flash disks are becoming considerably faster than traditional disks but minimizing any synchronous writes remains desirable.

require the majority of servers to remain operational for recovery to succeed. The algorithms have been described in detail in the technical report [13].

3 Motivations and Goals

The key observation is that all three above algorithms depend on a protocol to catch up on the current state of DSTM by obtaining all necessary data from other servers that remain operational. These data include the most up-to-date state (a snapshot of objects) and transactional operations in-progress (a log). JPaxos uses the catch-up algorithm also in normal (no recovery) operation to—whenever possible—avoid leader election which is an expensive operation (the details are in [13]). The catch up algorithm requires object snapshots, which can be either triggered by JPaxos, or by DSTM using some hints from JPaxos, or by DSTM alone. By taking snapshots, the log is not growing forever. More importantly, it would be impractical and too slow to recover from a log only. On the other hand, communicating a snapshot of all objects via a network is an expensive operation. Thus, some strategies are clearly required to optimize the crash recovery process.

Our work aims at developing formally efficient strategies for the catch up protocol and snapshot/log maintenance which take into account the semantics of DSTM. The range of research problems that we plan to investigate include:

- the trade-off between obtaining object snapshots versus missing operations,
- the choice of time when to take snapshots,
- the possible integration of catching up on the current state with transaction certification.

The solutions to these problems can take into account transaction processing operations, the length of transactions, the size of objects, and the type of workload. On the theory side, the above requires to compute the upper and lower time (or other) bounds, and to analyze paths for possible optimizations and efficient strategies.

To our best knowledge there was no prior rigorous work on crash recovery in distributed STM. There may be two reasons for this. Firstly, contrary to Paxos STM the majority of distributed STM systems do not tolerate crashes, or depend on a central coordinator. The closest system to Paxos STM is D2STM [15]—a distributed STM in which all objects are also replicated on each node and atomic broadcast is also used for an optimistic transaction certification. Secondly, these problems may *appear* at first look close to well-researched problems of crash recovery in database systems. However, this work usually assumes crash recovery depending on transaction rollback and the lock-based transaction processing, while we consider object replication based on deferred update relying on atomic broadcast. The latter approach requires the crash recovery protocols designed for group communication (or Paxos, in the context of Paxos STM). Such a recovery protocol is at the lower level of abstraction than the transaction processing protocol. However, to achieve good performance both protocols should cooperate. The challenge is to reconcile the DSTM and Paxos protocols to achieve the best performance for various workloads.

The Paxos distributed consensus protocol and MultiPaxos that extends Paxos to state machine replication were proposed by Lamport in [12]. The work of Oki and Liskov [16] on the Viewstamped Replication protocol for state machine replication is based on the same ideas as MultiPaxos. The crash-recovery model has been the subject of many research papers. In both of the original descriptions of Paxos ([12] and [16]), the authors write that by logging to stable storage one message per consensus instance, their protocols can tolerate catastrophic failures. Unfortunately, this operation introduces a large overhead to the protocol. Therefore, efforts have been made to reduce the usage of stable storage. In [17], Liskov describes a variant of viewstamped replication that reduces the number of stable storage accesses to one per view change (see also [13] for more references). The work on JPaxos described in [13] further improves upon these algorithms, by reducing the amount of stable storage accesses from one per view change to one per recovery.

However, the above work abstracts from the specific features of DSTM, such as:

- multicore concurrency,
- transaction execution vs. abcast execution overhead,
- transaction conflicts and rollback,

- transaction checkpointing,
- and various optimizations, such as multiversioning, i.e., keeping multiple versions of transactional objects to guarantee successful commitment of all read-only transactions.

For efficiency and to avoid duplication of operations, the crash recovery schemes should be carefully redesigned to reflect these characteristics. In order to reason about advantages, limitations, and possible optimizations of crash recovery schemes for DSTM, we need an analytical model that abstracts from any uninteresting details.

4 A Formal Model of Crash Recovery in DSTM

We develop a formal model of crash recovery in distributed software transactional memory systems. Our model is aimed at the distributed STM systems that are based on deferred update replication relying on atomic broadcast. The model considers the aforementioned specific features of DSTM. The main results of this work so far are the following:

- We give the upper and lower bounds on processing concurrent transactions (assuming no delays); the lower bound is given for unoptimized and optimized variants of DSTM, where the latter recognizes read-only requests and treats them differently; the bounds include an (optional) time of synchronous operations to stable storage;
- We estimate the overhead associated with three example recovery schemes (implemented in JPaxos); we do it separately for service replicas during normal DSTM non-faulty operation (i.e., when there are no crashes), and for a replica recovering after a crash. The overhead in the former case is included in the upper and lower bounds analysis above. Our analysis shows precisely the benefits of multi-core CPUs, when some operations can be executed in parallel, and also the impact of DSTM optimizations, such as keeping versions of transaction objects for read-only requests, and detecting conflicts as early as possible.
- We used our model to propose efficient strategies for catching up on the current state by a recovering replica (using the most up-to-date object snapshot and a log obtained from other replicas); the strategies take into account transaction processing operations, the length of transactions, the size of objects, and the type of workload.

Acknowledgments The authors would like to thank Tadeusz Kobus and Maciej Kokociński for their work on the implementation of Paxos STM. This work has been partially supported by the Polish Ministry of Science and Higher Education within the European Regional Development Fund, Grant No. POIG.01.03.01-00-008/08.

References

[1] N. Shavit and D. Touitou, "Software transactional memory," in *Proceedings of PODCS '95: the 14th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Aug. 1995.

[2] T. Harris and K. Fraser, "Language support for lightweight transactions," in *Proceedings of OOPSLA '03: the 18th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Oct. 2003.

[3] T. Kobus, M. Kokociński, and P. T. Wojciechowski, "Practical considerations of distributed STM systems development (abstract)," in *Proceedings of WDTM '12: the 1st Euro-TM Workshop on Distributed Transactional Memory*, Feb. 2012.

[4] *Paxos STM – User Guide, Release 1.0*, http://www.it-soa.pl/paxosstm.

[5] P. A. Bernstein and N. Goodman, "Serializability theory for replicated databases," *Journal of Computer and System Sciences*, vol. 31, no. 3, pp. 355–374, Dec. 1985.

[6] M. Stonebraker, "Concurrency control and consistency of multiple copies of data in distributed Ingres," *IEEE Transactions on Software Engineering (TSE)*, vol. 5, no. 3, pp. 188–194, May 1979.

[7] B. Charron-Bost, F. Pedone, and A. Schiper, Eds., *Replication: Theory and Practice*, ser. LNCS. Springer, 2010, vol. 5959.

[8] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 1996.

[9] D. Agrawal, G. Alonso, A. E. Abbadi, and I. Stanoi, "Exploiting atomic broadcast in replicated databases (extended abstract)," in *Proceedings of EuroPar '97: The Third International Euro-Par Conference on Parallel Processing*, Aug. 1997.

[10] B. Kemme, F. Pedone, G. Alonso, and A. Schiper, "Processing transactions over optimistic atomic broadcast protocols," in *Proceedings of ICDCS '99: the 19th IEEE International Conference on Distributed Computing Systems*, Jun. 1999.

[11] A. Schiper and M. Raynal, "From group communication to transactions in distributed systems," *Communications of the ACM*, vol. 39, no. 4, pp. 84–87, Apr. 1996.

[12] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, May 1998.

[13] J. Kończak, N. Santos, T. Żurkowski, P. T. Wojciechowski, and A. Schiper, "JPaxos: State machine replication based on the Paxos protocol," Faculté Informatique et Communications, EPFL, Tech. Rep. 167765, Jul. 2011.

[14] N. Santos and A. Schiper, "Tuning Paxos for high-throughput with batching and pipelining," in *Proceedings of ICDCN '12: the 13th International Conference on Distributed Computing and Networking*, 2012.

[15] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues, "D2STM: Dependable Distributed Software Transactional Memory," in *Proceedings of PRDC '09: the 15th IEEE Pacific Rim International Symposium on Dependable Computing*, Nov. 2009.

[16] B. M. Oki and B. H. Liskov, "Viewstamped replication: A new primary copy method to support highly-available distributed systems," in *Proceedings of PODC '88: the 7th ACM Symposium on Principles of Distributed Computing*, 1988.

[17] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams, "Replication in the Harp file system," in *Proceedings of SOSP '91: the 13th ACM symposium on Operating Systems Principles*, Oct. 1991.