Supporting Partial Data Replication in Distributed Transactional Memory

[Extended Abstract]

João A. Silva, Tiago M. Vale, Ricardo J. Dias, Hervé Paulino, and João M. Lourenço
CITI — Departamento de Informática
Universidade Nova de Lisboa, Portugal
{jaa.silva,t.vale,ricardo.dias}@campus.fct.unl.pt {herve.paulino,joao.lourenco}@fct.unl.pt

1. INTRODUCTION

Transactional memory (TM) [8] is consistently making its way into mainstream programming, being already deployed by some of the major CPU manufacturers [11] and in several reference compilers [5]. To cope with requirements such as scalability and dependability, recent proposals explore the combination of TM with data replication, bringing TM to distributed environments — conceiving distributed transactional memory (DTM).

However, current DTM frameworks support only full data replication [2, 10]. They provide the best possible level of tolerance to data loss, but limit the system's total storage capacity to the capacity of the node with fewer resources, and require coordination among all the system's nodes, an approach bound to hamper scalability in large scale systems.

In this context, a partial data replication [1] strategy can help to lessen these shortcomings. Each node replicates only a subset of the system's dataset, an approach that aims at combining the best of data distribution and full replication, while trying to attenuate their disadvantages. The key idea is to allow the dataset to be distributed among the participating nodes and to decrease the number of nodes that have to participate in a transaction's confirmation, as any given transaction only has to be confirmed by the nodes that replicate the data items in its read and write sets. By distributing the data and reducing the coordination cost among nodes, partial data replication leverages the system's scalability.

Although this strategy has already been explored by the distributed databases research field [6], it is yet to be addressed in the context of (D)TM. More specifically, partial data replication has been broadly applied in key-value stores [7], and even though these work on in-memory data and support transactions, they present significant differences when compared with DTM systems for general purpose programming languages.

To this extent, we propose PARDSTM, to the best of our knowledge, the first DTM framework to include support for partial data replication. As such, the contributions of this work are: a reasoning on how partial data replication shall be supported in general purpose programming languages (Java, in particular), and a modular software framework that embeds such principles to provide a highly expressive and non-intrusive programming API. Initial experimental results give evidence that our approach may enhance scalability in large scale systems, when compared to full data replication. An ongoing comprehensive study will allow us to assess in which contexts of use (workloads, number of nodes, etc.) partial data replication may be an effective alternative.

2. SUPPORTING PARTIAL DATA REPLICATION IN A GENERAL PURPOSE PROGRAMMING LANGUAGE

Supporting partial data replication in general purpose programming languages may build upon the cumulative knowledge of applying such techniques to (in-memory) databases. However, the expressiveness of programming languages is much higher than that of database query languages, specially compared to the put/get interface of key-value stores. As such, addressing partial data replication in this new context raises an extra set of challenges, such as (1) what data should be partially replicated; (2) how to express that replication "level" in a general purpose programming language; and (3) how to partially replicate graphs of objects. Orthogonal to these are the technical challenges raised by the architectural and functional requirements for a runtime system to support partial data replication.

In partially replicated databases, the tables are present in every node, i.e., fully replicated, while the data they contain is partitioned among the nodes. The same approach is also applied to key-value stores, where data organizing structures exist in every node and just the hard data is partially replicated. In sum, partial replication is, to some extent, always combined with full replication. In our opinion, the same reasoning should be applied to DTM in order to mitigate the overhead of remote read operations. A pragmatic example is the list. If we imagine that all the nodes are partially replicated, the simple task of traversing the list would entail a possibly remote read operation for each iterated node. On the other hand, if we just partially replicate the hard data stored in each node, the number of remote read operations resultant from the list's traversal would be limited by the hard data we really want to inspect. Thus, our answer to challenge (1) is to fully replicate the (data) structures (small or frequently accessed data) and to partially replicate the hard data (big or infrequently accessed data).

We want to keep the public API as simple as possible, while allowing for a high degree of expressiveness. Accordingly, our answer to challenge (2) is to grant the programmer with the power to express what data should be partially replicated. By default, everything is assumed to be fully replicated. Further, we added a Java annotation — `@Partial` — to be applied in class' fields, expressing that everything downstream of this field, in the heap objects' graph, should be replicated in a single group, i.e., partially replicated.

Regarding challenge (3), we distribute and replicate the graphs of objects associating distribution metadata with each object. These metadata enable the system to know in which nodes the objects are replicated and to manage the references indirections. To ensure the semantics of `@Partial`, cycles in the graph may not include both full and partially replicated objects.

Additionally, the runtime system needs to be aware of which nodes belong to which groups and which data items are associated to which groups. These two problems are taken care of by two modules: the *group* and *data* partitioners[1], respectively. Hence, replicated data items are identifiable in the system by an unique identifier (to distinguish between different objects) and the identifier of the group where they are replicated (to be able to request a copy, if needed).

The developed framework is highly modular, allows for multiple implementations of the *group* and *data* partitioners, and of the protocol for transactions' confirmation. Concerning the latter, the proof-of-concept implementation resorts to an implementation of SCORe [4].

3. EXPERIMENTAL RESULTS

Our initial experimental results give evidence of some nice properties, namely[2]: the system uses less memory per node, since nodes do not replicate the entire system's dataset; the system has a fair workload distribution, since each node contributes roughly with the same amount of work for the system's overall throughput; and the system scales in the presence of workloads with large amounts of transactions that modify partially replicated data, this relates to the protocol that we use for transactions' confirmation, since it was specially tailored for pure partially replicated environments (and not for environments using a combination of both full and partial data replication).

We also verified that the system presents a considerable overhead in read operations, both for local and remote reads. For local reads, the problem is related with the protocol we use, since it requires additional checks (and in some cases busy waiting). For remote reads, the problem is the inevitable cost of distribution — the network (this is not the case in full replication, where all read operations are local).

To evaluate our system, we used some well known benchmarks such as Vacation from the STAMP suite [3], TPC-W [9] and a Red-Black Tree microbenchmark. In Figure 1, we show the result of running an adapted version of the Red-Black Tree microbenchmark. This version of the microbenchmark was adapted so that write transactions only modify partially replicated data (in this case, it only modifies the data stored in the tree's nodes).

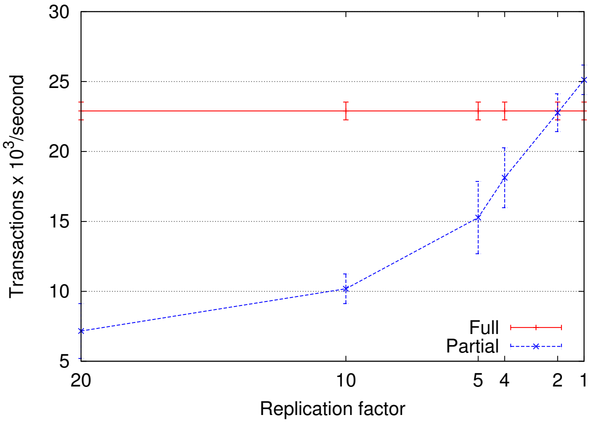

Figure 1: System's throughput on the Adapted Red-Black Tree microbenchmark (using JGroups, 20 nodes, 1 thread per node and 10% writes).

4. CONCLUDING REMARKS

To conclude, we have a working prototype of a DTM framework — PARDSTM — supporting both full and partial data replication. Its modularity and non-intrusive API allow the easy implementation of several of it components.

Ongoing work encompasses framework optimizations, caching of remote objects and an extensive comprehensive study that will allow us to accurately assess in which contexts of use (workloads, number of machines, etc.) our approach may be an effective alternative.

5. REFERENCES

[1] G. Alonso. Partial database replication and group communication primitives (extended abstract). In *ERSADS*, 1997.

[2] N. Carvalho, P. Romano, and L. Rodrigues. A generic framework for replicated software transactional memories. In *NCA*, 2011.

[3] C. C. Minh, J. Chung, C. Kozyrakis, et al. Stamp: Stanford transactional applications for multiprocessing. In *IISWC*, 2008.

[4] S. Peluso, P. Romano, and F. Quaglia. Score: A scalable one-copy serializable partial replication protocol. In *Middleware*. 2012.

[5] M. Schindewolf, A. Cohen, W. Karl, et al. Towards transactional memory support for gcc. In *GCC Research Opportunities Workshop*, 2009.

[6] N. Schiper, R. Schmidt, and F. Pedone. Optimistic algorithms for partial database replication. In *Principles of Distributed Systems*, 2006.

[7] N. Schiper, P. Sutra, et al. P-store: Genuine partial replication in wide area networks. In *SRDS*, 2010.

[8] N. Shavit and D. Touitou. Software transactional memory. In *PODC*, 1995.

[9] Transaction Processing Performance Counsil. TPC Benchmark W. `http://www.tpc.org/tpcw`, May 2013.

[10] T. M. Vale, R. J. Dias, and J. M. Lourenço. Uma infraestrutura para suporte de memória transacional distribuída. In *INForum Simpósio de Informática*, 2012.

[11] A. Wang, M. Gaudet, P. Wu, et al. Evaluation of blue gene/q hardware support for transactional memories. In *PACT*, 2012.

[1] We implemented some basic strategies for both components.

[2] During all the evaluation, we compared partial replication against full replication using our framework, in both cases.