Brief Announcement: On Speculative Replication of Transactional Systems*

Paolo Romano
INESC-ID
Lisbon, Portugal
romano@gsd.inesc-id.pt

Roberto Palmieri,
Francesco Quaglia
Sapienza Rome University
Rome Italy
palmieri@dis.uniroma1.it,
quaglia@dis.uniroma1.it

Nuno Carvalho,
Luís Rodrigues
INESC-ID/IST
Lisbon, Portugal
nonius@gsd.inesc-id.pt,
ler@ist.utl.pt

ABSTRACT

We define the problem of speculative processing in a replicated transactional system layered on top of an optimistic atomic broadcast service. A realistic model is considered in which transactions' read and write sets are not a priori known and transactions' data access patterns may vary depending on the observed snapshot. We formalize a set of correctness and optimality properties ensuring the minimality and completeness of the set of explored serialization orders within the replicated transactional system.

Categories and Subject Descriptors

D4.7 [**Organization and Design**]: Distributed Systems; D4.5 [**Fault Tolerance**]: Operating Systems

General Terms

Algorithms,Performance,Reliability

Keywords

Replication, Serialization Theory, Atomic Broadcast.

1. INTRODUCTION

Active Replication (AR) is a fundamental approach for achieving fault-tolerance and high availability. When applied to transactional systems, classic AR schemes, e.g. [5], require that, prior to start executing transactions, replicas agree on a common total order for transactions' serialization. This is typically achieved by executing some form of non-blocking distributed consensus protocol, such as Atomic Broadcast [3] (AB).

Since the latency of AB can significantly degrade the performance of a replicated system, recent approaches, e.g. [6], have pursued the idea of overlapping transaction processing and replica coordination by relying on a, so called, Optimistic Atomic Broadcast (OAB) service. OAB provides an "early" (though potentially erroneous) guessing of the final outcome of the coordination phase [11]. Exploiting optimistic message delivery, each site may immediately start the (optimistic) processing of the transactional request without waiting for the completion of the coordination phase. Clearly, this strategy pays off only if the final total order does not contradict the initial guess; otherwise optimistically activated transactions may access inconsistent snapshots and be forced to rollback.

Unfortunately, existing OAB-based replication solutions suffer from several limitations. First, they only permit the parallel activation of optimistically delivered transactions that are known not to conflict with each other [6]. Such a choice simplifies the management of local processing activities, sparing from the risks of propagating the results generated by optimistically delivered transactions. On the other hand, it requires a-priori knowledge of both read and write sets associated with incoming transactions in order to label transactions in distinct conflict classes. This requirement raises the non-trivial problem of systematically predicting transaction data access patterns and may, in practice, lead to significant over-estimation of the likelihood of transaction conflicts, especially in scenarios entailing forms of non-determinism in the data access pattern. As discussed in [8], this can yield to a strong reduction of the achievable parallelism, thus representing a major performance impairment for modern multi-core systems. Further, the effectiveness of existing OAB-based solutions is severely challenged in geographical scale replication, where the chances of accurately guessing the final order are often very small [7]. Finally, if the ratio between the coordination delay and the computation granularity is very large, as in the emerging scenario of Distributed Transactional Memories (DTMs) [2, 12], the actual performance gains achievable by existing OAB-based approaches can be extremely limited. As shown in [12], in fact, the local transaction processing time in a DTM environment is typically one or two orders of magnitude smaller than the replicas' coordination latency. In such contexts, the overlap between processing and communication achievable by existing OAB-based approaches provides negligible performance benefits [8], even in favorable scenarios where the optimistic guessing of the final delivery order happens to be correct with an extremely high probability.

In this paper, we address these limitations by investigating, from a theoretical perspective, the issues related to the adoption of a speculative approach to replication of transactional systems, which we call Speculative Transactional Replication (STR). The idea underlying STR is rather simple and consists of two main aspects: i) exploring multiple serialization orders for each optimistically delivered transac-

*This work was partially supported by the ARISTOS (PTDC/EIA-EIA/102496/2008) project.

SPAA'10, June 13–15, 2010, Thira, Santorini, Greece.
ACM 978-1-4503-0079-7/10/06.

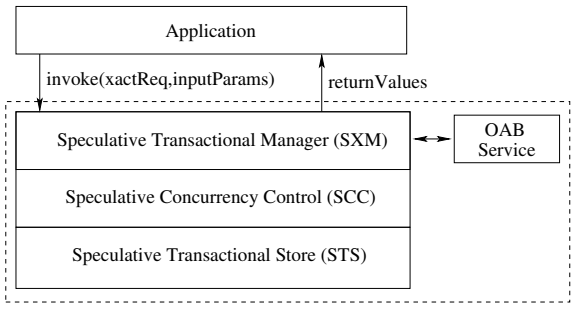

Figure 1: Software Architecture of Each Process.

tion, so to increase the probability of guessing a serialization order equivalent to the one finally determined by the OAB service; ii) allowing to observe the snapshots generated by optimistically executed transactions, rather than pessimistically blocking waiting for the outcome of the coordination phase, so to maximize the performance gains achievable by overlapping communication with local processing.

We frame the problem in a realistic model which does not assume the availability of any a-priori information on the set of data items to be accessed by the transactions (in either read or write mode), and in which transaction data access patterns can be influenced by the state observed during execution. Next, we formalize a set of correctness and optimality criteria for the speculative exploration of the permutations of the optimistically delivered transactions, demanding the *on-line* identification of all and only the transaction serialization orders that would cause the optimistically executed transactions to exhibit distinct outcomes.

2. SYSTEM MODEL

We consider a classical asynchronous distributed system model consisting of a set of processes $\Pi = \{p_1, \ldots, p_n\}$ that communicate via message passing and can fail according to the fail-stop (crash) model. We assume that the number of correct processes (i.e. the processes that do not fail) and the system's synchrony level are sufficient to permit implementing an OAB service.

OAB provides the following interface: *TO-broadcast*(m), which allows broadcasting messages to all the processes in Π; *Opt-deliver*(m), which delivers message m to a process in Π in a tentative, also called optimistic, order; *TO-deliver*(m), which delivers a message m to a process in Π in a so called *final order* which is the same for all the processes in Π.

The diagram in Figure 1 shows the architecture of each process $p_i \in \Pi$. Applications generate transactions by calling the `invoke` method of the local Speculative Transaction Manager (SXM), specifying the business logic to be executed (e.g. a DBMS stored procedure, or a method in a transactional memory) and any associated input parameters. The SXM is responsible of (i) propagating (through the OAB service) the transactional request across the set of replicated processes, (ii) executing the transactional logic on the underlying Speculative Transactional Store (STS), and (iii) returning the corresponding result to the application.

The STS layer abstracts low level storage mechanisms, which may encompass RAM-only memory accesses (as in transactional memories) and logging on persistent storage to ensure durability (as in conventional DBMSs). The STS maintains the state of a replica, modelled as a set of (multi-versioned) data items, and provides classical facilities for making visible the new versions of the set of data items updated by transactions. We assume that each data item X maintained by STS is associated with a set of versions, of which, at any time, a single committed version exists. Uncommitted versions residing in STS are, on the other hand, reflections of speculative computations, used to propagate updates along chains of speculatively executed transactions.

The interactions between the SXM and the STS are mediated by the Speculative Concurrency Control (SCC) layer, which externalizes a classical interface to trigger read/write operations on the data items maintained by the STS, as well as to commit and abort transactions. SCC can additionally trigger the re-spawn (i.e. the restart) and the fork of a (not yet committed) transaction, so to speculatively explore different serialization orders for that transaction. In order to univocally identify multiple, speculatively executed, instances of a same transaction T_i, we use the notation T_i^j to refer to the j-th instance of transaction T_i. We say that two speculative instances of the same transaction, say T_i^j and T_i^k, are two sibling transactions.

Each transaction T_i^j is associated with a sequence of operations $\mathcal{O}(T_i^j) = \{o_1^{T_i^j}, \ldots, o_n^{T_i^j}\}$ where an operation $o_l \in \mathcal{O}(T_i^j)$ is either a read or a write on a data item X, denoted, resp., as $r_l(X)$ and $w_l(X)$. We say that a transaction T_i^j is *completed* when it has fully executed its sequence of operation $\mathcal{O}(T_i^j)$. We assume that neither the sequence of operations to be executed within a transaction, nor the data items to be accessed by each operation are a-priori known. Conversely, we assume that the transaction data access pattern can vary depending on the current state of the underlying transactional store. More precisely, we assume that the transactional business logic is *snapshot deterministic* in the sense that if the same transaction is activated multiple times, the sequence of read/write operations it executes does not change unless the return value of any of its reads changes. In other words, if whichever instance of a transaction T always sees a snapshot S, defined as the set of values returned by all its read operations, then it behaves deterministically by always executing the same set of read/write operations. On the other hand, instances of a transaction T activated on different snapshots may generate different sequences of operations. More formally, consider two sibling transactions T_i^a, T_i^b, producing, respectively, the two sequences of operations $\mathcal{O}(T_i^a) = \{o_1^{T_i^a}, \ldots, o_n^{T_i^a}\}$ and $\mathcal{O}(T_i^b) = \{o_1^{T_i^b}, \ldots, o_k^{T_i^b}\}$. Let us assume, with no loss of generality, that $k \leq n$ and that $j \in [1 \ldots k]$ is the index of the first operation in $\mathcal{O}(T_i^a), \mathcal{O}(T_i^b)$ for which $o_j^{T_i^a} \neq o_j^{T_i^b}$. This implies that before executing $o_j^{T_i^a}$, resp. $o_j^{T_i^b}$, a read on a data item X was previously executed by T_i^a, resp. T_i^b, and that the two reads returned two different values.

3. THE STR PROBLEM

Our target correctness criteria is 1-copy serializability [1], which ensures that a transaction execution history $\mathcal{H}$ across the whole set of replicated processes Π is equivalent to a serial transaction execution history in a non-replicated system. More specifically, we are interested in *view serializability* [1, 10] defined as a property of $\mathcal{H}$ such that, for any prefix $\mathcal{H}'$ of

$\mathcal{H}$, its committed projection $C(\mathcal{H}')$ (obtained by deleting all operations not belonging to transactions committed in $\mathcal{H}'$) is *view equivalent* to some serial history.

We now introduce the notion of optimality for an STR algorithm. This is done by formalizing a set of properties ensuring both the consistency (view serializability) of the snapshot observed by any speculative transaction, and that *all and only* the speculative serialization orders in which the transactions observe *distinct* snapshots are explored.

Let $\Sigma = \{T_1, \ldots, T_n\}$ be the set of Opt-delivered, but not yet TO-delivered, transactions, and denote with $\Sigma^* = \{T_1^1, \ldots, T_1^k, \ldots, T_n^1, \ldots, T_n^m\}$ the set of the corresponding speculative transactions that have run to completion. We say that an STR algorithm is optimal if it guarantees the following properties:

• **Consistency:** *the history of execution of every speculative transaction in Σ^* is view serializable.*
• **Non-redundancy:** *no two sibling transactions in Σ^* observe the same snapshot, i.e.:*
$\forall T_i^a, T_i^b \in \Sigma^* \Rightarrow$
$\exists r_l^{T_i^a}(X) \in \mathcal{O}(T_i^a), \exists r_l^{T_i^b}(X) \in \mathcal{O}(T_i^b)$ s.t. $r_l^{T_i^a}(X) \neq r_l^{T_i^b}(X)$
• **Completeness:** *if the system is quiescent, namely the OAB service stops Opt-delivering and TO-delivering transactions, then for every permutation of Σ, say $\pi(\Sigma)$, there eventually exists a speculative transaction $T_i^j \in \Sigma^*$ that observes the snapshot produced by sequentially executing all the transactions preceding T_i in $\pi(\Sigma)$.*

The optimality property of an STR algorithm filters out trivial solutions based on the exhaustive enumeration of every possible permutation of the Opt-delivered transactions for the construction of plausible serialization orders. In fact, while such an approach would certainly enumerate the permutation that will be eventually generated by the final TO-deliver (thus providing completeness), it would require the processing of $\sum_{i=1\ldots n} \frac{n!}{(n-i)!} = \Theta(n!)$ speculative transactions (i.e. the number of nodes of a permutation tree [4] for a set of cardinality n), being n the number of Opt-delivered (but not yet TO-delivered) messages. More importantly, such a number of speculative transactions would be spawned independently of the actually developed conflict relations. This would likely cause the useless exploration of a (possibly very large) number of redundant serialization orders in which transactions execute along identical trajectories, thus producing the same snapshots and externalizing the same results to the application.

It is therefore desirable to design conflict-aware mechanisms able to identify all and only the serialization orders that would cause the Opt-delivered transactions to exhibit distinct execution trajectories and, ultimately, externalize different results to the application. The problem's complexity appears clearly manifest if one considers that the transactions' data access patterns, and consequently their mutual conflict relations, can be in practice significantly influenced by the observed state values. This raises the requirement for *on-line* solutions able to correctly deal with scenarios in which the activation of a same transaction according to different serialization orders causes the generation of different sequences of operations (as captured by the *snapshot deterministic* transaction execution model formalized in Section 2).

4. AN OPTIMAL STR ALGORITHM

An optimal solution to the STR problem can be found in our technical report in [9]. In order to determine the set of speculative serialization orders in which transactions need to be executed, this optimal STR algorithm relies on a novel graph-based construct, called Speculative Polygraph (SP). SPs are inspired by Papadimitirou's polygraphs, originally introduced in [10] to test the view-serializability of a non-speculative history. Conventional polygraphs are in fact unfeasible to reason on view serializability of a speculative transaction history since the simultaneous coexistence within a polygraph of two sibling transactions, representative of non-conciliable serialization orders, can corrupt the polygraph by introducing cycles that might render it useless.

Another interesting finding highlighted in [9] is that, when considering different realistic application workloads, an optimal STR algorithm requires on average the exploration of at most 5 serialization orders per transaction when the number of Opt-delivered, but not yet TO-delivered, messages is lower than 15. This result highlights, on one hand, the factual importance of the STR's non-redundancy property. On the other hand, it shows that the additional amount of hardware resources required by an optimal STR algorithm is, in realistic settings, expected to be relatively low and, we argue, likely satisfiable by modern multi-core processors.

5. REFERENCES

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[2] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues. D^{2}STM: Dependable Distributed Software Transactional Memory. In *Proc. Int. Symp. on Dependable Computing (PRDC)*. IEEE Computer Society Press, 2009.

[3] D. Powell (ed.). *Special Issue on Group Communication*, volume 39. ACM, 1996.

[4] G. Hetyei and E. Reiner. Permutation trees and variation statistics. *Eur. J. Comb.*, 19(7):847–866, 1998.

[5] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *Proc. Int. Conf. on Distributed Computing Systems (ICDCS)*, IEEE Computer Society, 1998.

[6] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proc. Int. Conf. on Distributed Computing Systems (ICDCS)*, IEEE Computer Society, 1999.

[7] J. Mocito, A. Respicio, and L. Rodrigues. On statistically estimated optimistic delivery in large-scale total order protocols. *In Proc. Int. Symp. on Dependable Computing (PRDC)*, IEEE Computer Society Press, 2006.

[8] R. Palmieri, F. Quaglia, P. Romano, and N. Carvalho. Evaluating database-oriented replication schemes in software transactional memory systems. *In Proc. Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS)*, IEEE Computer Society Press, 2010.

[9] P. Romano, R. Palmieri, F. Quaglia, N. Carvalho and L. Rodrigues. On Speculative Replication of Transactional Systems, INESC-ID Tec. Rep. 38/2009, 2009.

[10] C. H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26(4):631–653, 1979.

[11] F. Pedone and A. Schiper. Optimistic atomic broadcast: a pragmatic viewpoint. *Theor. Comput. Sci.*, 291(1):79–101, 2003.

[12] 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)*, ACM press, 2008.