SCS 3017 - LITERATURE SURVEY[1]

UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

DISTRIBUTED SOFTWARE TRANSACTIONAL MEMORY

Author:

Y.S. Horawalavithana

10002103

Supervisor:

Dr. D.N. Ranasinghe

December 10, 2013

[1]IEEE REF LATEX Mendeley JabRef - Word Count: 5000 apprx.

Declaration

I hereby declare that this literature survey report has been prepared by Y.S.Horawalavithana based on mainly the reference materials listed under the references of this report. No major components (sentences/ paragraphs etc.) of other publications are directly inserted in to this report with out being duly cited.

Signature	:	
Name	:	Y.S.Horawalavithana
Date	:	December 10, 2013

Abstract

The Emergence of many/ multi core systems suggest that learning parallel programming is becoming increasingly important for implementing real world applications requiring high throughput. Transactional Memory (TM) which can aid concurrent control in parallel computing has recently found its way to distributed system researches. Providing an alternative to lock based concurrency control in distributed systems, Distributed Software Transactional Memory (DSTM) combines the simplicity of TM with the scalability & failure resiliency by exploiting redundancy of distributed platforms. In this review paper we mainly address memory replication in DSTM which mostly provides serializability & opacity but may also incur high communication overhead among distributed nodes in achieving concurrency & fault tolerance. However we argue by presenting some currently available solutions, which point to DSTM being a serious challenge to existing technology for handling large transaction applications.

Keywords. Software Transactional Memory; Concurrent programming; Many-core systems; Replication

Acknowledgments

I would like to give special thanks to...

Dr. D.N. Ranasinghe, Senior Lecturer at UCSC
for being an always helpful & interested supervisor who offered invaluable advices throughout the survey

Dr. A.R. Weerasinghe, Senior Lecturer at UCSC
for giving me guidance to build my research skills

Prof. Bo Kagstrom, Director at HPC2N research lab, Umea
for being the professor who made me more interested in Distributed Systems

Lars Karlsson, PHD Fellow at UMIT research lab, Umea
for being always helpful in discussing various related research problems

Asst. Prof. Francisco Hernandez at Umea University
Primal Wijesekara, PHD Fellow at UBC, Canada
& *Charitha Madushanka*, at WASN research lab, Sri Lanka
for giving feedbacks on review drafts

My Family
for supporting me the many years of my studies

...As well as everyone who spent the time to listen to my question & helped me along the way of completing this survey

Contents

Acronyms

2PC 2 Phase Commit

AB Atomic Broadcast

DSTM Distributed Software Transactional Memory

DSM Distributed Shared Memory

DTM Distributed Transactional Memory

EC2 Amazon Elastic Compute Cloud

HPC High Performance Computing

HTM Hardware Transactional Memory

HyTM Hybrid Transactional Memory

MPI Message Passing Interface

P2P Peer-2-Peer

PGAS Partitioned Global Address Space

RMI Remote Method Invocation

RPC Remote Procedure Call

STM Software Transactional Memory

TLS Thread Level Speculation

TM Transactional Memory

UGAS Unified Global Address Space

URB Uniform Reliable Broadcast

1 Introduction

There'd been nearly dramatic shift in computer industry not to build a bigger or faster processor but putting more cores in a single processor chip, by making it smaller in order to get better throughput. If we want to run our applications faster, we need to learn parallel programming which get use the cores concurrently. That increasingly influences ordinary programmers to become concurrent programmers. But concurrent programming is bit too hard. Today's concurrent programming is heavily based on locks & condition variables which introduce variety of problems & trade-off in performance, scalability & software engineering [1].

1.1 Software Transactional Memory (STM)

Transactional Memory (TM) is a promising concept in simplifying shared memory multi core programming. TM ensures the correctness & performance over conventional locking schemes by replacing critical sections from an atomic block. Transactions will be executed atomically with respect to shared memory while TM synchronizes between concurrent transactions. It helps programmers to focus on high-level synchronization while ignoring the low level implementation details. TM avoids problems that locks & condition variables have introduced like priority inversion, convoying, and deadlocks [2] in concurrent programming. By implementing transactional semantics in software, STM offers flexibility for programmers in an architecture independent way.

Serial equivalence property(*serializability*) of transactions avoid many problems in concurrent systems. It preserves that the combined transactions which are separated in time, will produce the same effect as they had been performed one at a time. The key idea behind many STM systems is to ensure *serializability*.

When utilizing multi cores in concurrent programming, nested transactions serves well where it can perform parallel actions in each level of transactions' subtrees. Also the ability that sub-transactions can work independently as a set of nested transactions is potentially more robust in multi-core systems.

1.2 Motivation

There did exist important discussions in famous research debate on STM [1, 3] about it's own issues which are not presented in lock based concurrent systems. The former [1] identifies

that accessing shared memory object outside a transaction & the way transaction uses it's locks inside a transaction do lead to bottleneck in interaction with non-transactional codes. That shows it's not clear how to interact with codes which can't be wrapped inside an atomic block: transaction. In the presence of partial failures there was no guarantee that all transactions do make any progress which make doubts on general transaction semantics. But the latter clearly said that STM can still outperform sequential code using different hardware configurations & workloads as benchmarks which cleared many doubts. They've clearly motivated further researches in the field by increasing STM performance from using methods like manual instrumentation & transparent privatization etc. [3]

As TM heavily based on atomicity, any model implemented using that, should care more on strengthen the atomicity. While many Hardware Transactional Memory (HTM) systems provide strong atomicity, the researchers did pay their views to achieve it in STM as well. Many STM systems detect conflicts by treating non-transactional code as short transactions to increase performance. Also researchers suggest various methods that can be used dynamically to increase STM performance [4]. But there's a strong believe that Hybrid Transactional Memory (HyTM) which combines HTM & STM will be the strong candidate in boosting overall performance [5].

2 Distributed Model

When application requirements go beyond of limited scalability & high available terms, multi-core become insufficient in case we need a fault tolerance system. In case larger distributed systems or computer clusters can provide the required scalability & robustness, we still need to avoid distributed concurrency problems (e.g. distributed deadlocks[1]) when implementing those systems. We can extend the TM abstraction to distributed application in the same rationale where that TM makes attractive for multi-core systems.

2.1 DTM Design choices

The larger design space that comes with Distributed Transactional Memory (DTM), makes many choices to design distributed systems powered by distributed transactions. In case DTM proposed as a paradigm for distributed memory we can create a taxonomy in the way of memory abstraction. Unified Global Address Space (UGAS) is a simple programming model where the

[1] http://en.wikipedia.org/wiki/Deadlock#Distributed_deadlock

programmer can't control where the data is placed & the transactions are executed which suffers from optimization problem. In the other hand Partitioned Global Address Space (PGAS) gives implicit control to the programmer to optimize for locality but still remaining as a complicated programming model.

Talking about another design dimension, there're classical distributed execution models where DTM supported including a) *control flow*, where objects are statically assigned to distributed nodes & the transactions are invoked object operation through Remote Procedure Call (RPC). Securing atomicity it then performs 2 Phase Commit (2PC)[1] to ensure that all changes are successfully commit or unless they will be discarded at once. While control flow model reduces the network traffic among nodes, it increases the computation requirements at remote nodes. As a down side if the data patterns are dynamic in remote nodes, control flow suffers from poor data locality unless it can utilize standard caching mechanism b) *data flow* where objects are migrated to invoke immobile transactions but sometimes it's costly to locate & mobilize data c) *hybrid model* combines both above models [6]. As trade-offs are emerged in different execution models, DTM systems eagerly need another design choice.

2.2 Replication

The amount of time which a service is available, can be increased by maintaining copies of data between servers - *Replication*.Replicating TM provides a powerful mechanism for concurrent programmers to deal with machine failures and developing distributed lock based synchronization schemes.

. So if we have n number of servers where each server has a p probability of failing or becoming unreachable, then the availability of an object stored at each of these servers, can be increased if we have more failure-independent servers [7].

$$1 - probability(all\, managers\, failed\, or\, unreachable\,) = 1 - p^n$$

Caches don't necessarily keep collection of objects entirely at application level. In that case server replication performs better than common caching techniques in enhancing availability.

[1] http://wwwlgis.informatik.uni-kl.de/cms/fileadmin/courses/SS2008/Informationssysteme/Addons/2PC.pdf

Fault tolerance system must guarantee correct behavior of high available data. Data & service replication between computers do ensure that there will be correct servers which can outvote the wrong values produced by failure models.

2.2.1 Replica Schemes

There're replication schemes, where data can be replicated across nodes in full or partial way.

In full replication, data is replicated among all participating nodes where it gives a fault-tolerance system. These systems perform well in read only transactions where it can avoid remote communication by executing locally. Let's assume a system where data is not replicated & has only single object copy. In nodes' partial failures, objects held by failed nodes will be lost & the following transactions have never been committed. But in fully replicated systems where single computer is formed by a group of linked computers ensures that above trade-off will not be happened in nodes' partial failures [8]. But as number of nodes grow, the transmitted messages increase quadratic-ally in metric-space network where the communication cost between two neighbor nodes form a metric. So the broadcasting transactional primitives (read/write sets, memory differences) will not be scaled. To address this issue partially replicated object models come into act.

Someone can notice the heavy brute force replication in fully replicated systems where objects are replicated in each nodes. So each node is maintaining replicas of all objects. It can cause communication overhead where it increases locality while ensuring one-copy serializability [1]. It's quiet costly even though full replication model doesn't want to handle objects' requests & retrieve operations. To avoid this expensive replication, clusters of nodes can be created ensuring at least one object replica should be in each cluster. It may avoid the loss of objects since multiple nodes used to know about the object in the cluster. Partially replicated systems are more scalable because replicas only need to update data items where only they have local copies. But in the presence of larger work loads these systems don't perform well as they need to deal with expensive remote data accesses. Caching mechanisms were produced over time to deal with this issue by ensuring that they are not violating transactional semantics [9]. Also there're active schedulers [8, 10] where they schedule replica transactions to gain high performance.

[1] http://www.springerreference.com/docs/html/chapterdbid/64254.html

2.2.2 Implementation

Using atomic broadcast (Total order broadcast) client-server replication is easily implantable. But implementing atomic broadcast is the hard part which abstracted by the consensus problem [1]. Solving consensus depends on the system models synchronous & asynchronous. As concluded by recent research results [11], consensus problem is not solvable in asynchronous systems. Since considering worst case bounds at message transmission delays & process relative speeds, consensus is solvable in synchronous systems even where processes crash. Those worst case issues limit the definition of new transactional models. But there're models like partially synchronous systems & asynchronous systems augmented with failure detectors to solve consensus [12]. Based on atomic broadcast, several non-blocking transaction certification systems are developed to set the required state automatically after any node fails. Researches developed such systems where the performance differs from the crash-recovery algorithms [13].

As static group replication models touch the limitations, dynamic groups are emerged to break the shackles but introducing new problems. Modeling dynamic groups, updating group members & implementing group communication primitives are some of new problems to deal with. By determining alive nodes & agreeing on nodes' dynamically changing behavior, group membership service mechanism can be used to solve some kind of issues in the dynamic group model (synchronous)[12].

If we consider a system model where object replicas are stored in volatile memory, they each maintain a system state. While DSTM is more into service replication among nodes, it guarantees that modification to any data replicas should update their states on every nodes.

Implementation Techniques

Depending on the operation, the replica manager that receives the request should manage the coordination between other replica managers in the group if necessary. Also available copies of replication have to be validated & updated among the group to reach consistency. Quorum consensus [7] schemes are developed to reduce the number of replica managers required to perform an operation. In most practices, good techniques should enable operations within transactions even in the circumstances where all replicas are not reachable.

[1] http://www.cs.duke.edu/courses/fall07/cps212/consensus.pdf

3 Transaction Properties

DSTM is originated by the convergence between Distributed Shared Memory (DSM) & Database Replication. As common transaction property ACID reflects the characteristics where we can argue whether it's still required for DSTM as its' originates. ACID properties are separable in where we can have A Atomicity without having I Isolation or having both A and I without D - Durability. But the transaction's serializability possess in their completely. C consistency has been provided by the programmers where we can't guarantee to provide D Durability to user transactions. The participating nodes in a distributed transaction should either commit or abort securing the fundamental concept atomicity. By preserving other nodes from observing any operations before the transaction succeeds on specific object, DSTM serves Isolation as well. But Durability hasn't been considered as a useful purpose in system level activity because synchronizing accesses to a shared memory object of a single node will be done at times where that node is running.

If we think a cloud data center as an instance of distributed system, the providers should consider about high availability of data in general. Describing C Consistency, A Availability and P Partition tolerance by the CAP theorem [14], it says that they can't have them in simultaneously in a distributed system. So the data consistency provided by the cloud providers have early doubts in their minds about the support they've given as transactions which were conflicted with CAP theorem. That has led not to include transactions in their cloud platforms early on. Maintaining high availability & scalability in cloud data centers, CAP theorem reflects that providing transactions are conflicted with feasibility issues [15].

The consistency can be achieved via eventual consistency[1] where the changes are made in your program will be shown everywhere including all replicas given no time boundaries that can wait. But it has no guarantee to application programmers whether the consistency is really being achieved or not [16]. Recent NoSQL data stores apply above relaxed consistency approach to cover up node failures.

Application programmers should aware more about the way that the data is partitioned among nodes. In 2PC approach there are overheads in communicating with transactions because every node with data is a member of executing above conventional approach. As virtual disks provided by cloud infrastructure solves above problem partially but still remains prob-

[1] http://en.wikipedia.org/wiki/Eventual_consistency

lems in single node caching mechanism as it leverages with the application scalability. The Deuteronomy approach [17] strongly believes that it can cure above dilemma promising that transactional programming is a good way to access cloud data in the long run .

4 DSTM Drawbacks & Performance

Designing TM systems for distributed environments may differ from the traditional TM systems which are designed to single processor system. As DSTM systems are risen with largely growing cluster computing, many important aspects are there to take care more importantly. Not surprisingly many drawbacks are originated from those aspects [18]:

4.1 Drawbacks & Solutions

Co-ordination latency An efficient DSTM system should either reduce the information exchanged between distributed nodes or use a better communication mechanism to reduce the coordination latency. To reduce the inter-replica coordination cost the recent research had focused to use space efficient Bloom filter based encoding properties to reduce network traffic in the expense of user-tunable increase at the transaction abort rate [19]. Despite on nodes' failures above approach yields strong consistency properties & data high availability over classical replication schemes based on fine-grained locking & atomic commit. To develop better replication schemes [20] provides a leased based certification scheme based on cheaper Uniform Reliable Broadcast (URB)[1], by reducing more inter-replica coordination overhead. In particular to reduce replica communication overhead [19, 21] both have made their step to reduce the transaction's commit phase latency & cover transactions from many repeated aborts which can be happened due to high conflicts of workloads. Also to reduce the latency of accessing remote objects, symbolic prefetching techniques were introduced recently [22].

$$TM\,throughput\,ratio = \frac{Transaction\,processing\,time}{Replica\,coordination\,latency}$$

In case transactions haven't incurred either in disk access latencies or there're no overheads like common SQL statement parsing & plan optimization in database replication, transactions are often shorter in DSTM than other transactional systems. Due to expensive coordination latency above ratio can be extremely small & no generic DSTM prototypes were built regularly.

[1] http://disi.unitn.it/ montreso/ds/handouts/04-rb.pdf

To reduce this overhead, researches introduced another replication scheme which is overlapping phases of transactions processing while reducing overhead in updating transactions [23].

Also DSTM applications suffer from idle CPU time due to distributed coordination.Low CPU utilization is somewhat hard to ignore in large clusters with hundreds of nodes where the intention is to utilize the resources well.

Partial failures Failure of distributed node shouldn't stop the whole system. We addressed that array of DSTM models integrated with replication models may work even in partial failures. But robust DSTM systems deal with crash-recovery failures by storing application snapshots in stable storage or maintaining necessary data in volatile memory integrated with chosen algorithms. It may guarantee to provide fault-tolerant properties [13]

Interaction between replicas By ignoring false low level conflicts, one can achieve better performance in TM model. Many available content managers provide these services via detecting conflicts while keeping effective transactional time of each node [24]. Changing priorities among conflicted transactions, contention manager can achieve starvation freedom in all distributed nodes. Also elastic transaction models provide higher concurrency when implementing search structures. Instead of aborting & restarting transactions when conflicts are detected, elastic transactions split the work prior to the conflict & commit that [25].

No general protocol Many replication schemes can be integrated to improve DSTM performance. Even previously mentioned STM research debate ended up with giving all burden to the workload taken as benchmarks. That was led to design an autonomous fashioned framework where it can decide the protocol to apply at run time by looking the workload ahead. Recent works [26, 27] suggest that even machine learning approaches can be used to design effective learning techniques to predict the costs of different coordination primitives.

4.2 Performance Benchmark

As a class of Distributed System, Peer-2-Peer (P2P) systems serve as a good benchmark to investigate many DSTM drawbacks. Since P2P nodes can leave & join the network anytime, failure of a node becomes more complicated to detect. Also it leads to block the whole system as it may hold any lock. Even though data replication is strict consideration in P2P system, data hosts suffer from low memory bandwidth relative to large clusters. As network will not remain

synchronous as always, the coordination latency can become higher. Though large clusters do more care about computing high contention, P2P system uses sharing & collaboration mostly.

DSTM should be generic to all classes of Distributed Systems in its architectural concepts, but not the design choices; Applying DSTM to highly unreliable distributed environments like P2P with bad design choice won't lead to achieve any of Distributed System properties (performance, fault tolerance, scalability) in well manner. Solving issues like data security, asymmetric connections; control-flow model will be most suitable for P2P systems as it can computationally power up the remote nodes [28]. Replication is highly considered to implement in systems where consensus is solvable [12]

Even though DSTM may have drawbacks on the given design issue, it's important to understand the nature of distributed system & apply the right shape of DSTM. Many research works present designs of contention managers & cache-coherence protocols to achieve better DSTM characteristics.

5 In Today's World

Increasing demand of pay-only-for-what-you-use pricing model, cloud computing becomes so much popular, by shifting general computer processing power into another level. In case developers need to explore new ways that they can use to simplify the development of large scale parallel applications, many researches did find their paths to come up with new programming paradigm in recent time (e.g.: MapReduce, Sinfonia, Scope)

Given the huge value which transactions can provide, we simply can't avoid them in designing cloud platforms. Researches concluded that distributed transactions have performed well in the past, which is local to a data center [29]. Google and Microsoft have announced that they provide local transactions[30] while Amazon still rely on Oracle support in their cloud data centers [31]. Local transactions are now being considered as golden in user perspective where the early feasible issues are not seen any more with growing scalability issues. Even though Google was in flux about whether to use distributed transactions or not, it's good to say that they came with providing them recently via 2PC[30]

Google's web search indexing system was previously designed by MapReduce approach, but it was suffered from latency issues by discarding the work done at earlier runs. Maintaining large repositories of documents & updating them as soon as new documents crawled, distributed machines over the network need to access remote repositories concurrently. DTM model makes more attractive to cloud computing by using resources transparently in a distributed environment. DTM model is useful for batch consistency actions in single synchronization phase at commit time. As a way to help in incremental computation, Google came up with new DTM protocol called Prelocator [32] to modify Google web search index which reduce the average age of documents in search results by 50 %.

Dynamo is one of the storage service platform which is used by Amazon to serve large umber of transactions at user peak times using thousands of servers. The system is heavily depended on distributed transactions where replication has been done using consistent hashing techniques. The system follows weaker consistency model using object versioning where it's acceptable on application primitives [33].

Infinispan [34] from RedHat is another replication model like Dynamo where it's backed by P2P architecture to distribute consistent states across the network. Weaker consistency models

were practiced in Cloud-TM project [24] to find more efficient implementation of Infinispan. Unlike Dynamo, Walter [35] is a geo-replicated system where data is replicated asynchronously by agreeing with Parallel Snapshot Isolation [35] to achieve consistency.

Distributed concurrency control algorithms are developed in cluster environments with multi version capability. Most of them are implemented using DTM model by default [28]. While the algorithms ensure one-copy serializability, it can avoid common explicit multi-versioning. Giving independent capability to operate on their own data snapshots, it yields better result mostly in read only transactions.

To handle large number of transactions in the production environment, STM replication has been experimented & deployed as an academic system recently - FenixEdu [36].

Public clouds were investigated for High Performance Computing (HPC) recent time [37]. Even there're specialized programming languages like X10 or Fortress which provide DTM abstraction in HPC domain, they're not widely adaptable yet. Existing legacy codes which developed over decades is being the barrier to write new good softwares that scale to pet scale systems. Another reason may be that leading chip manufactures still need to be backward compliant with their architectures.

Recent results of executing DSTM applications in public cloud shows that if we can get concurrency characteristics of the environment, we can achieve scalability from the application [38]. But at current date there're no good scheduling method devised even for the popular Amazon Elastic Compute Cloud (EC2) [31].

6 Other Similar Approaches

As approaches to design complex systems in distributed environment, there're different kinds of variations were developed over last decades like RPC, DSM. In case DSM models don't require message passing, they hide the communication overhead which exist in RPC models. But later DSTM models are emerged with the simplicity of locking convention & the increasing of programmers' productivity. They outperform the relaxing memory consistency models which are introduced by software DSM models efficiently [39]. DSM models require explicit synchronization of locks, barriers where many DSTM models introduce them by transactions' semantics. Also DSTM models minimize the frequent to keep memory consistency at transactions. But as distributed environment becomes larger, performance is decreased in DSM specially cases where it uses home directory protocol [40].

There have been up & down results based on workloads when evaluating DSTM against Remote Method Invocation (RMI) which uses read/write locks to prevent deadlocks & live locks. RMI performs better in low number of read-only transactions as experimented [38]. As many DSTM models do keep system state at objects, it may not perform well in high proportion of read/write transactions but not in other side. In case that the distributed platform they used in [38] as benchmark, didn't have any large number of distributed instances could affect above results. Many variants of control flow models in DSTM were developed based on RMI which suffer problems introduced in section *DTM Design choices*

There're platform dependent approaches to parallelize the code which could minimize the heavy overhead of administrating cluster computers used in classical Message Passing Interface (MPI) models [41]. But overall those approaches suffered from node failures. Check pointing & roll back recovery techniques in MPI models, ensure the fault tolerance, but starting the application repeatedly in failure recoveries was led to heavy performance overhead. But recent MPI models migrate tasks to proactive nodes using compiler tools & libraries when anticipating node failures which can avoid above overhead too [42]. So at current date both MPI & DSTM replication models are good candidates in achieving user transparency over the presence of partial node failures in distributed environments.

7 Conclusion

In case large body of literature emerged with database replication, they can represent the source to inspire to develop better replication schemes for TM as they share the same notion of atomic transaction. As DSTM is originated by the convergence between DSM & Database Replication, the techniques used in both approaches can be used to build fault tolerant systems in DSTM where it replicates the memory state so that it can survive with partial failures of distributed nodes.

7.1 Future Work

Minimizing replica synchronization & conflict rate at run time, we can enhance the overall DSTM performance. In the development of high reliable & robust distributed systems using TM abstraction, researches are currently paying their views to use methods like abstract data types, higher level conflict detection and relaxation of consistency models.

Most STM systems use weak Isolation models to get high performance in multi-core systems. When we extend it to distributed systems, we want to reach strong Isolation where we should guarantee the correctness of transactional semantics. Future research will focus to get rid of this trade-off.

By exploiting DSTM semantics into Thread Level Speculation (TLS)[1] techniques, will support automatic, performance effective parallelization of sequential codes is another research believe.

There're recent studies on performing DSTM applications in public clouds. But the lack of good performance models did pull out most researches from their expected outcomes. Combining better scheduling methods in public clouds with good performance models, may have a significant impact on cloud application performance when using DSTM as a practice.

7.2 Suggestion

Many DSTM implementations were introduced over time including key DSTM features which are inter-operable with popular programming languages. That is a good sign to believe that DSTM is not just being a concept but in a stage where we can apply to real world problems.

[1] http://en.wikipedia.org/wiki/Speculative_multithreading

In case concurrent programmers are used to practice common technologies to cure parallel programming dilemma it's time to experiment these new models which can integrate with current work. We believe that DSTM can simplify the programmer to code and maintain even large-scale HPC software in public clouds while still providing good parallelism & concurrency control. The time has yet to come!

References

[1] C. Blundell, M. Michael, H. W. Cain, P. Wu, and S. Chiras, "software transactional memory : Why is it only a research toy?," no. September, 2008.

[2] T. Free and L. Is, "The Free Lunch Is Over : A Fundamental Turn Toward Concurrency in Software," pp. 1–7, 2009.

[3] A. Dragojevi and R. Guerraoui, "Why STM can be more than a Research Toy," *Communications of the ACM*, vol. 54, pp. 70–77, April 2011.

[4] M. Abadi, T. Harris, and M. Mehrara, "Transactional memory with strong atomicity using off-the-shelf memory protection hardware," *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming - PPoPP '09*, p. 185, 2008.

[5] M. Moir, "Hybrid Hardware / Software Transactional Memory," *US Patent*, no. 7395382, 2005.

[6] M. M. Saad, "Distributed Hybrid-Flow STM [Technical Report],"

[7] T. K. G. B. George Coulouris, Jean Dollimore, *Distributed Systems: concepts and design*. Addison-Wesley Pearson, 5 ed., 2012.

[8] J. Kim, B. Ravindran, and V. Tech, "Scheduling Transactions in Replicated Distributed Software Transactional Memory,"

[9] P. Romano, "Elastic, scalable and self-tuning data replication in the cloud-tm platform," *EWDCC '12 Proceedings of the 1st European Workshop on Dependable Cloud Computing*, vol. 5, 2012.

[10] J. Kim, R. Palmieri, and B. Ravindran, "Enhancing Concurrency in Distributed Transactional Memory through Commutativity," *Euro-Par 2013 Parallel Processing*, vol. 8097, pp. 150–161, 2013.

[11] K. Birman, *Reliable Distributed Systems*. New York: Springer, 1 ed., 2004.

[12] F. P. B. Charron-Bost and A. Schiper, *Replication: Theory and Practice*, vol. 5959. Springer, 2010.

[13] P. T. Wojciechowski and J. Koczak, "A Formal Model of Crash Recovery in Distributed Software Transactional Memory (Extended Abstract)," pp. 1–5.

[14] E. A. Brewer and U. C. Berkeley, "Inktomi at a Glance Distributed Systems Understanding Boundaries Where s the state ? (not all locations are equal) Santa Clara Cluster Delivering High Availability," pp. 1–12, 2000.

[15] D. Lomet, "Transactions : From Local Atomicity to Atomicity in the Cloud Atomicity and Transactions : A Personal Perspective," pp. 38–52, 2011.

[16] A. E. Abbadi, "Big Data and Cloud Computing : New Wine or just New Bottles ?," pp. 1647–1648.

[17] A. Fekete, D. Lomet, and G. Weikum, "Unbundling Transaction Services in the Cloud," pp. 1–10, 2009.

[18] L. Rodrigues and P. Romano, "Distributed Software Transactional Memories : A Summary of Research @ IST / INESC-ID," *IST@INESEC-ID*, pp. 19–20, 2012.

[19] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues, "D2STM: Dependable Distributed Software Transactional Memory," *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 307–313, Nov. 2009.

[20] N. Carvalho and P. Romano, "Asynchronous Lease-based Replication of Software Transactional Memory," no. 257784, 2008.

[21] I. to Reliable Distributed Programming, *Guerraoui, R., Rodrigues, L.:.* Springer, 2006.

[22] A. Dash and B. Demsky, "Automatically Generating Symbolic Prefetches for Distributed Transactional Memories,"

[23] M. Couceiro, I.-i. Ist, and P. Romano, "A Machine Learning Approach to Performance Prediction of Total Order Broadcast Protocols," *IST@INESEC-ID*.

[24] P. Romano, L. Rodrigues, and I.-i. Ist, "Cloud-TM : Harnessing the Cloud with Distributed Transactional Memories," *ACM SIGOPS Operating Systems Review*, vol. 44, pp. 1–6, April 2014.

[25] V. G. Pascal Felber and R. Guerraoui., "Elastic transactions," *International Symposium on Distributed Computing - DISC,*, pp. 93–107, 2009.

[26] M. Couceiro, I.-i. Ist, P. Romano, L. Rodrigues, and I.-i. Ist, "PolyCert : Polymorphic Self-Optimizing Replication for In-Memory Transactional Grids," no. May, 2011.

[27] V. H. P. Bernstein and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Pearson-Wesley Publishing Company, 1 ed., ISBN O-201-10715-5.

[28] A. P. Lacko, Peter, "Designing Software Transactional Memory for Peer 2 Peer Systems," *Foundations of Computing and Decision Sciences*, vol. 38, pp. 111–122, 2013.

[29] T. D. Group, "Nonstop sql, a distributed, high-performatlce, high-availability implementation of sql," *Tandem Technical Report*, April 1987.

[30] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, L. Jean-michel, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore : Providing Scalable , Highly Available Storage for Interactive Services," pp. 223–234.

[31] Amazon, "Amazon: Oracle & aws http://aws.amazon.com/oracle/," December 2013.

[32] D. Peng and F. Dabek, "Large-scale Incremental Processing Using Distributed Transactions and Notifications,"

[33] M. J. G. K. A. L. A. P. S. S. P. V. Giuseppe DeCandia, Deniz Hastorun and W. Vogels, "Dynamo: Amazons highly available key-value store," *SOSP '07 Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pp. 205–220, 2007.

[34] F. Marchioni and M. Surtani, "Infinispan data grid platform," *Packt Publishing*.

[35] M. K. A. J. L. Yair Sovran, Russell Power, "Transactional storage for geo-replicated systems," *SOSP '11 Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 385–400, 2011.

[36] P. Romano, N. Carvalho, and L. Rodrigues, "Towards distributed software transactional memory systems," *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware - LADIS '08*, p. 1, 2008.

[37] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, p. 395, 2010.

[38] Y. Yoshino and M. Aritsugi, "An Experiment on Performing DSTM Applications in a Public Cloud," *2012 41st International Conference on Parallel Processing Workshops*, pp. 179–187, Sept. 2012.

[39] P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy release consistency for software distributed shared memory," *Proceedings of the 19th annual international symposium on Computer architecture - ISCA '92*, pp. 13–21, 1992.

[40] M. M. Saad and B. Ravindran, "Snake : Control Flow Distributed Software Transactional Memory," *SSS'11 Proceedings of the 13th international conference on Stabilization, safety, and security of distributed systems*, pp. 238–252, 2011.

[41] P. Kraj, A. Sharma, N. Garge, R. Podolsky, and R. a. McIndoe, "ParaKMeans: Implementation of a parallelized K-means algorithm suitable for general laboratory use.," *BMC bioinformatics*, vol. 9, p. 200, Jan. 2008.

[42] P. G. Ivn Cores, Gabriel Rodrguez and M. J. Martn, "Failure avoidance in mpi applications using an application-level approach," *The British Computer Society*, 2012.