Partial Replication Policies for Dynamic Distributed Transactional Memory in Edge Clouds

Diogo Lima, Hugo Miranda, François Taïani

Partial Replication Policies for Dynamic Distributed Transactional Memory in Edge Clouds

Diogo Lima[1,3] Hugo Miranda[1] François Taïani[2]

[1]LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
dlima@lasige.di.fc.ul.pt, hamiranda@ciencias.ulisboa.pt

[2]Université de Rennes 1, IRISA, ESIR, Rennes, France
francois.taiani@irisa.fr

[3]Escola Superior de Hotelaria e Turismo do Estoril, Portugal

ABSTRACT

Distributed Transactional Memory (DTM) can play a fundamental role in the coordination of participants in edge clouds as a support for mobile distributed applications. DTM emerges as a concurrency mechanism aimed at simplifying distributed programming by allowing groups of operations to execute atomically, mirroring the well-known transaction model of relational databases.

In spite of recent studies showing that partial replication approaches can present gains in the scalability of DTMs by reducing the amount of data stored at each node, most DTM solutions follow a full replication scheme. The few partial replicated DTM frameworks either follow a random or round-robin algorithm for distributing data onto partial replication groups. In order to overcome the poor performance of these schemes, this paper investigates policies to extend the DTM to efficiently and dynamically map resources on partial replication groups. The goal is to understand if a dynamic service that constantly evaluates the data mapped into partial replicated groups can contribute to improve DTM based systems performance.

CCS Concepts

•**Computing methodologies** → **Distributed computing methodologies;** *Self-organization;*

Keywords

Edge Clouds, Partial Replication, Geographical Distribution

1. INTRODUCTION

Edge clouds are a decentralized and heavily distributed cloud infrastructure that encourages applications to be executed in close proximity to the input data. An increasing number of of services are being deployed on edge clouds in order to reduce latency and network traffic. However, services relying on edge clouds need to mediate the interaction between a variety of different actors and to coordinate the concurrent access to the same data by multiple applications, possibly executing on distinct sites.

Transactional Memory [6] is a concurrency mechanism aimed at simplifying the development of applications by allowing operations to execute in an atomic way. Analogous to database transactions, transactional memory defines a specific sequence of tasks that are considered a transaction, that either commits (and all changes produced by the tasks become visible) or aborts (and no change is visible). Originally proposed as a hardware architecture [6] for shared memory access, Transactional Memory was later extended to address parallelism in multiprocessor systems, under the term *Software Transactional Memory* (STM) [4, 5].

Distributed Transactional Memory (DTM) [1, 7, 10, 11, 15, 17, 18, 19] extends STM to distributed systems, addressing new issues such as data replication and node failures. It provides an additional abstraction level to the programmer, where traditional distributed programming details (e.g. socket management or data serialization) become transparent and integrated with concurrency in a unique and consistent approach.

In spite of recent studies showing that partial replication approaches can present gains in scalability by reducing the amount of data stored at each node [17], most DTM solutions follow a full replication scheme where all nodes in the system keep a replica of every object. However, intuition suggests that, as participants are more geographically dispersed, the latency and message concurrency to validate and commit transactions in a group of nodes increases, hampering system performance. This is the scenario expected to occur in large scale networks, where processes running on geographically dispersed edge clouds support some distributed application.

Motivated by recent commercial applications, in particular, large scale, augmented reality games (i.e Ingress and Pokemon Go), this paper assumes distributed applications running over multiple edge clouds share data. In such a scenario, the overall system performance depends of a policy capable of deploying the replicas of the data in proximity, in order to reduce the latency associated to each transaction. The paper proposes and evaluates a set of policies to extend the DTM to efficiently and dynamically map resources on partial replication groups. The goal is to understand

MECC '16 December 12, 2016 June, Trento, Italy

DOI: 3017116.3022872

if a dynamic service that continuously evaluates how data is mapped into partial replicated groups can contribute to improve the system's efficiency.

2. REPLICATION IN DTM

Replication of objects in DTMs can serve two purposes: to improve availability in the presence of faults and to improve performance by making the data locally available at the interested nodes.

2.1 Full Replication

The majority of DTM approaches adopt a full certification-based replication scheme: all nodes in the system keep a replica of every object, and transactions are locally executed, synchronizing objects state with the other replicas at commit time. This synchronization can be achieved either through a voting or non-voting certification. As observed in the DMV system [11], in the voting certification approach, a committing transaction needs to broadcast its updates to the other nodes and will only commit if they vote favorably.

In the non-voting approach, a communication round is saved as the decision can be taken locally and therefore, replicas do not need to reply to the transaction's issuer. The need to vote is replaced in a trade-off by the need to exchange more data during the update broadcast from the committing transaction [9]. In particular, the transaction owner must provide both the transaction's write and read sets. D^{2}STM [1] follows a variant of this non-voting certification-based replication, where bloom filters are used to reduce the size of broadcasted messages.

Since the non-voting certification approach allows replicas to independently validate transactions and every data item is replicated among all nodes, the crash of nodes in the system does not harm consistency. However, coordination of all nodes imposes a considerable communication overhead. Namely, broadcasting transactional read/write sets is inherently non-scalable, as the number of messages grow quadratically with the number of nodes present in the system [10].

2.2 Partial Replication

In partial replication, the full application's dataset is subdivided into n partitions and each partition is replicated in a group of m nodes. Partial replication is more scalable because committing transactions only need to reach the groups storing the data items accessed in the transaction.

To the best of our knowledge, SCORe [14] is the only partial replication protocol developed for DTM systems. SCORe combines the Two Phase Commit (2PC) algorithm [3] with Skeen's total order multicast [2] to form a commit protocol that ensures that only the replicas that maintain data accessed by a transaction participate in its outcome. SCORe relies on logical clocks that allow each node to keep two scalar timestamps: the *commitId* which is the timestamp of the last update transaction committed on that node, and the *nextId* which indicates the next timestamp the node will propose for a remote commit request.

At commit time, the transaction issuer multicasts a totally ordered validation message to all involved replicas. Every replica that receives this message validates the transaction If the validation is confirmed, the *nextId* is piggybacked on the reliably unicasted *vote* message and the transaction is stored by the voter in a local pending buffer. The transaction issuer then collects all *vote* messages (aborting the transaction in case one of the contacted node does not respond within a predefined timeout), sets the transaction's final commit timestamp as the maximum of the proposed *nextId* and multicasts back the *decide* message with the transaction's outcome and the *commitId*. If the outcome is positive, the receiving replicas buffer the transaction in a queue of stable transactions. A transaction T is finally committed only if there are no other transactions in both pending and stable buffer with a timestamp less that T's *commitId*.

In order to distribute data items among replication groups, SCORe uses a pseudo-random algorithm and therefore does not exploit data and node partitioning to its full extent. For example, as participants are more geographically disperse within a group, the increased latency and message concurrency induced to validate and commit transactions may hamper the system's performance.

This issue has been addressed with more judicious data distribution policies such as geographical distribution in distributed transactional SQL databases like CockroachDB [1], which allows the user to define replica locations. CockroachDB builds its SQL database on top of a transactional and strongly consistent sorted monolithic key-value store.

Replica location in CockroachDB is based on the types of failures a user wants to tolerate, e.g. replication in different servers within a datacenter to tolerate power failures, or different servers in different datacenters to tolerate large scale network or power outages. By contrast, because we target edge clouds, we aim at using replica location and data distribution to reduce latency and maintain system performance, by providing a dynamic service that constantly evaluates the data mapped into partial replicated groups.

This solution is also being studied in state machine replication [8], to achieve a dynamic scalable state machine replication designed to exploit workload locality. Data mapping onto partitions is managed by an oracle and state reconfiguration follows a simple rule: whenever a command requires data from different partitions, the client first multicasts *move* commands to the oracle and to the involved partitions so that all data is moved to a single partition and then the command is executed on that partition. Following this approach, the usage patterns will shape the data mapping and reduce multi-partition commands in a reactive manner. Differently from this work, we propose to explore policies that anticipate user requests and apply group changes proactively.

The geo-replicated storage system presented in [16] proposes to reduce latency and increase transaction throughput by reordering transactions so that those enclosed in a single partition are not delayed by global transactions involving multiple partitions. However, data placement into partitions is not addressed while we investigate data mapping policies in order to maximize the number of local transactions.

3. GEOGRAPHICAL-AWARE REPLICATION

In large scale networks, where servers and edge clouds are geographically dispersed among different regions, an erroneous data mapping may contribute to increase the latency and message concurrency to validate and commit transactions. In this paper, we assume that a dynamic resource mapping service (DRMS) is running in background and is

[1]https://www.cockroachlabs.com/

constantly evaluating and adapting the mapping of data on partial replication groups (PRG). The goal is to understand if there are policies that can contribute to maintain the system's efficiency, by determining at run-time the most suitable set of replicas for each object.

DRMS must thus trade-off between maximizing the number of transactions enclosed in a single PRG, while minimizing the amount of group changes and the geographical distance between the issuer of the transaction and the replicas of the data. Group changes should be avoided as they are known to be a time and resource consuming operations, due to the mandatory definition of synchronization points.

Anticipating the future utilization of data items is challenging and can only be efficiently determined if the application follows some usage pattern. Still, a number of distinct policies can be found. As a preliminary experiment, this paper evaluates the following partial replication policies for DRMS:

Static once an object is assigned to a PRG, it never changes. Transactions must access all the PRGs hosting each object;

Move to the Latest Transactions begin by transferring all participating objects to their PRG. Objects remain hosted on that PRG until they are requested by another transaction on another PRG;

Conservative Group Polling The DRMS keeps a usage counter associated to each object. After each transaction, the DRMS changes the objects if its usage on transactions issued by a different group is 1.5 times greater than the average usage of every object on every group. Transactions must access all the PRG hosting each object;

Aggressive Group Polling The DRMS keeps a usage counter associated to each object on each PRG. After each transaction, objects are moved to the PRG where they are more frequently used. Transactions must access all the PRG hosting each object.

The *Static* and the *Move to the Latest* represent the two extremes of the range of possible policies. In particular, *Static* completely avoids the movement of objects, forcing transactions to access all the PRGs involved. This would be the most convenient solution if the cost of moving objects was considerably higher than the cost of having transactions to access multiple PRGs. *Move to the Latest* mirrors the approach of [8] by assuming a negligible cost in moving objects from one PRG to another. In this scenario all transactions are guaranteed to be executed on a single partial replication group since the objects are previously moved, but is expected to be heavily penalized by an excessive number of group changes.

The *Conservative* and *Aggressive Group Polling* policies are more balanced approaches that avoid the permanent change of PRG by trying to identify object usage patterns in the system. Both keep information about where an object was used and how many times and only decide for group changes whenever a deviation from the average usage pattern is found.

It should be noted that all these policies seek to identify the most suitable (popular) location for each individual object, but that they ignore any potential correlation between objects manipulated by the same transaction. Exploiting such correlations is outside the scope of this paper and left as future work.

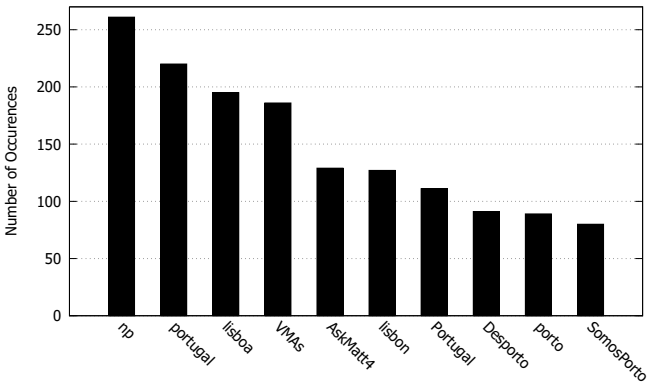

Figure 1: 10 more popular hash tags found

4. EVALUATION

To understand the impact of dynamic partial replication policies on system performance, we prepared a simple evaluation scenario based on Twitter, modeling each tweet as a transaction and each hash tag as one object accessed within the transaction. Twitter was preferred over traditional DTM benchmarks, such as Vacation from the STAMP suite [12] or TPC-W,[2] as a way to reinforce the location dependency and correlation between objects participating in a transaction. This is in contrast with the randomness of both STAMP and TPC-W in the selection of the objects participating in each transaction.

Data was collected between August the 17th and August the 29th 2016, using Twitter's Streaming API,[3] configured to collect the hash tags used in tweets performed within a 25km radius of four of the Portuguese biggest cities (Lisbon, Porto, Braga and Faro). In our experiment, each city represents the location of one Partial Replication Groups (PRG). Although the Twitter's Streaming API returns at most 1% of all tweets produced, the study performed in [13] claims that the distribution of geo-tagged tweets returned using the API is statistically representative of the hash tag diversity found in the complete set of tweets.

Overall, 5897 tweets containing one or more hash tags were collected. Lisbon was the most popular location, counting 62% of the tweets, followed by Porto (25%), Braga (8%) and Faro (5%). Most (66%) of the tweets had 1 hash tag. Tweets with 2 to 4 hash tags represented respectively 13%, 7% and 4% of the total. The largest number of hash tags observed in a tweet was 13. Figure 1 depicts an histogram of the 10 more popular hash tags among the total 6932 hash tags found. Of those, there are 5117 hash tags that only appear once.

4.1 Evaluation Setup

Evaluation was performed executing the implemented 4 policies, which is publicly available on-line[4] over the dataset.

[2]http://www.tpc.org/tpcw
[3]https://dev.twitter.com/streaming/overview
[4]https://github.com/dmslima/PRG_policies

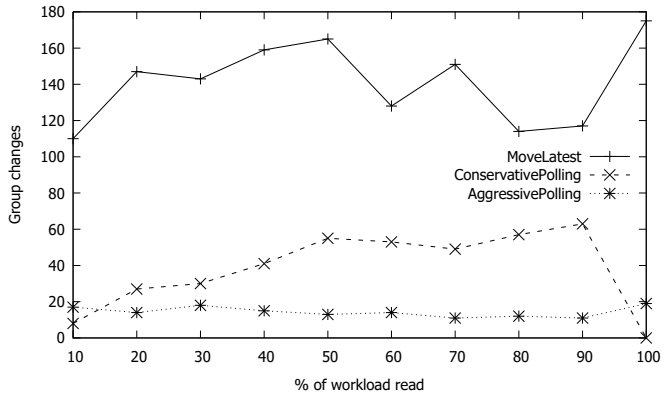

Figure 2: Number of group changes performed by the dynamic partial replication policies.

Additionally, both the *Random* and *Round-Robin* location insensitive algorithms were also included in the evaluation as control experiments. Transactions are processed in chronological order and consist in the list of objects accessed (i.e. the hash tags), with no distinction between read and write operations, and the ID of the Partial Replication Group (i.e. the city) where the transaction was performed.

Initially, each hash tag is affected to the partial replication group of the location where it was first tweeted, excluding the *Random* and *Round-Robin* simulations.

4.2 Experimental Results

The performance of each policy was evaluated using 3 distinct metrics, that reflect the trade-offs discussed above. Namely, the number of changes of objects between groups and the number of groups participating in a single transaction are expected to be as small as possible in order to reduce additional overhead to the DTM. The proportion of transactions enclosed in a single group complements the metrics above by showing the cases with the lowest overhead to the DTM.

To illustrate the adaptation capability of each policy, plots represent the metrics refreshed at every 10% of the transactions (∼590 transactions), counting exclusively the transactions performed in that period.

4.2.1 Group Changes

Group changes are time and resource consuming operations, so an efficient and dynamic partial replication data mapping policy should keep this value as low as possible. The amount of group changes performed in the system is depicted in Figure 2, omitting the *Static*, *Round-robin* and *Random* policies, not applicable to this case as objects permanently remain in the PRG where the first transaction using them occurred.

As expected, the *Move to the Latest* is heavily penalized by the volume of group changes performed, averaging around 140 group changes at each 10% of the workload read. Both *Polling* approaches show much lower volumes of group changes, with the *Aggressive Polling* making, on average, ∼10 times less groups changes than the *Move to the Latest* policy, and the *Aggressive Polling* making ∼4.5 times less changes. However, the *Conservative Polling* policy presenting a higher volume of group changes than the *Aggressive Polling* in almost every 10% slice of the evaluation is surprising since applying group changes based on an average of object usage should present less group changes than moving whenever a new PRG becomes the most frequent user of that object. One explanation to the this phenomenon may reside in the characteristics of the data collected with the presence of a considerable amount of *single shot hash tags*, i.e. hash tags that only appear once. If many of those are present in the workload, the global average usage of every object on every group tends to the decrease. This means that a larger set of objects start having its usage 1.5 times above the global average, which means that they are now considered to move groups.

4.2.2 Groups by Transaction

The goal of the average number of groups involved in a transaction metric is to assess if data is being efficiently correlated. A value of 1 represents an ideal situation in which the hash tags of a tweet are all located in the same partial replication group in which the transaction is executed. Conversely, the higher this average is, the more overhead will occur on each transaction, showing that the data mapping service is not correctly identifying data correlation and efficiently applying group changes.

Figure 3 depicts the average number of groups involved in a transaction with error bars showing the distance of the average to the standard deviation. We observe that, for the *Static* policy, this average remains around 1.2 during the entire set of transactions, while both *Random* and *Round-Robin* algorithms consistently remain above 2. This means that, even if objects changes are not allowed along the experiment, just by providing an initial location aware knowledge to object assignment into PRGs, there is already a significant reduction in the number of groups involved per transaction. Moreover, from the *Polling* policies, we can see that the *Aggressive Polling* policy is still able to reduce this average throughout the workload, while the *Conservative Polling* does reduce this average for the first 10% of the workload read, but then remains consistently above the *Static* policy for the rest of the evaluation. This indicates that the *Aggressive Polling* policy not only makes fewer group changes, but they are also more effective in comparison with the *Conservative Polling* policy. In addition, the latter is producing group changes that are becoming counter-productive when combined with the increase of the average number of groups involved in a transaction shows.

4.2.3 Single Group Transactions

To have a better understanding of this previous metric, we now focus on the proportion of transaction enclosed in a single group. i.e., those that have all the objects in the same PRG where the transaction was requested, in order to to better assess the policies' ability to improve the system's data mapping. Expectations are that the partial replication policies will increase this value close to 100%, which represents the best partial replication data mapping scenario.

The results, depicted in Figure 4, show that both location insensitive *Random* and *Round-Robin* algorithms are significantly outperformed by the remaining policies. Among the location-aware policies, the *Aggressive Polling* policy has a consistent gain over the *Static* policy, with approximately 5% more transactions enclosed in a single PRG on every slice of 10% of the workload. This figure also confirms the inef-

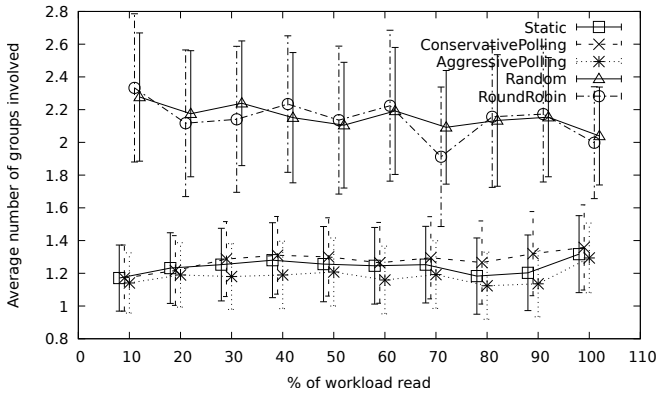

Figure 3: Average number of groups involved in a transaction.

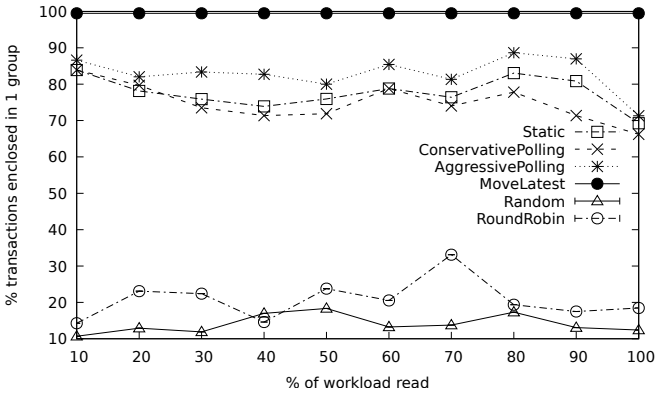

Figure 4: Average number of transactions enclosed on a single partial replication group.

fectiveness of the *Conservative Polling* policy, which is only able to marginally improve the percentage of transactions enclosed on a single PRG over the control test experiment (i.e. *Static*) at the first 20% of the workload.

One particular aspect observable in this figure is the sudden and equal decrease in the average number of transactions enclosed on a single PRG by all policies in the last 10% of the workload. This phenomenon is being caused by the *VMAs* hash tag. As observed in Figure 1, this is the fourth most popular hash tag with a total number of 186 occurrences. However, the first time it appears is at tweet 4791 out of 5897, issued in Lisbon, and all of the remaining 185 repetitions appear within the last 10% of the workload. More precisely, after being generated in Lisbon, this hash tag is used once in Porto and another in Braga before being repeated 12 times in a row in Lisbon. After these first 15 occurrences, the hash tag is evenly used by the PRGs of Lisbon and Porto and very sporadically in Braga and Faro. Which means that, for the *Static* policy, approximately half of the 170 remaining tweets using the hash tag *VMAs* are not enclosed on a single PRG. Considering that these all occur in the final 10% of the workload, the *VMAs* hash tag is singly contributing for half of the 30% of tweets not enclosed in a single PRG for that policy. Moreover, the same behavior is repeated for both of the *Polling* policies since those 12 initial repetitions pulling the hash tag to be kept in Lisbon is never reverted by the Porto RPG that does not generate enough tweets to become the most frequent group in object usage, but uses it as much as Lisbon. Thus, the object is never decided to move and the amount of transactions enclosed on a single PRG starts decreasing.

However, such behavior is expected in Twitter. The *VMAs* hash tag suggests a seasonal interest with global visibility that became popular, likely to be reproduced by other hash tags in the future, and represent a particular specificity of the workload chosen.

4.3 Discussion

A preliminary conclusion of the evaluation is that the adoption of a very conservative policy that maintains the objects on the PRG where they were first created can already present interesting results. Considering the Twitter's case, objects are mostly used in the location where they are created. This explains why the *Static* policy is able to present percentages of transactions enclosed on a single PRG already above 70%.

However, the results of the *Aggressive Polling* policy proves that there is still some room for improvements. On the other hand, the presence of *single shot hash tags* has lead to a significant decrease of the global object usage average, leading the *Conservative Polling* policy to increase group changes, where it was firstly expected to have less group changes than the *Aggressive Polling*. In fact, the latter not only produces less group changes, but also more effective ones as it is the only one able to improve the system's performance when compared to the *Static* configuration. Nevertheless, it should be noted that these results are susceptible to change with other datasets.

5. FUTURE WORK

The evaluation showed that more needs to be investigated in the field of dynamic partial replication policies with additional research challenges to be answered. For example, by further investigating possible correlation among distinct objects created by transactions. Moreover, the fact that a policy under-performed for this specific workload, does not mean it will not be more suitable than others for other workloads with different characteristics. We intend to gather larger and more diversified datasets to validate our results and provide a set of policies able to adapt to a variety of different workload characteristics. In addition, a uniform dataset, with meaningful correlation between objects and locations, possibly leveraging the logs of some popular application, would greatly contribute to support and compare the evaluation of such policies.

6. CONCLUSION

This paper discusses policies to extend the DTM to efficiently and dynamically map resources onto partial replication groups, in order to understand if a dynamic service that constantly evaluates the data mapped can contribute to maintaining a system's efficiency. The paper evaluated four partial replication policies on the data collected from the Twitter social network. We observed that the proposed policies can positively impact the performance of our case study, in comparison with off-the-shelf approaches that either always or never migrate objects from one partial repli-

cation group to another.

Results suggest that where the objects were first generated is often where they will be most frequently used. Further research must be performed to investigate deterministic workload characteristics and provide policies to adapt to the latter as well as a uniform test bed to evaluates such policies.

Acknowledgments

Work described in this paper was partially supported by Fundação para a Ciência e Tecnologia, Portugal, under project PTDC/EEI–ESS/5863/2014 - doit.

7. REFERENCES

[1] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues. D2STM: Dependable distributed software transactional memory. In *Proc. of the 2009 15th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'09)*, pages 307–313, 2009.

[2] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, Dec. 2004.

[3] J. Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481, 1978.

[4] M. Herlihy, V. Luchangco, and M. Moir. A flexible framework for implementing software transactional memory. *SIGPLAN*, 41(10):253–262, Oct. 2006.

[5] M. Herlihy, V. Luchangco, M. Moir, and W. N. Scherer, III. Software transactional memory for dynamic-sized data structures. In *Proc. of the 22nd Annual Symposium on Principles of Distributed Computing (PODC'03)*, pages 92–101, 2003.

[6] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300, May 1993.

[7] M. Herlihy and Y. Sun. Distributed transactional memory for metric-space networks. In *Proc. of the 19th Int'l Conference on Distributed Computing (DISC'05)*, pages 324–338, 2005.

[8] L. L. Hoang, C. E. Bezerra, and F. Pedone. Dynamic scalable state machine replication. In *46th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2016.

[9] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *Proc. of the 18th Int'l Conference on Distributed Computing Systems*, pages 156–163, May 1998.

[10] J. Kim and B. Ravindran. Scheduling transactions in replicated distributed software transactional memory. In *Proc. of the 13th IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 227–234, May 2013.

[11] K. Manassiev, M. Mihailescu, and C. Amza. Exploiting distributed version concurrency in a transactional memory cluster. In *Proc. of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06)*, pages 198–208, 2006.

[12] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford Transactional Applications for MultiProcessing. In *IEEE International Symposium on Workload Characterization*, pages 35–46, 2008.

[13] F. Morstatter, J. Pfeffer, H. Liu, and K. M. Carley. Is the sample good enough? comparing data from twitter's streaming API with twitter's firehose. *CoRR*, abs/1306.5204, 2013.

[14] S. Peluso, P. Romano, and F. Quaglia. Score: A scalable one-copy serializable partial replication protocol. In *Proc. of the 13th Int'l Middleware Conference (Middleware'12)*, pages 456–475, 2012.

[15] M. M. Saad and B. Ravindran. Hyflow: A high performance distributed software transactional memory framework. In *Proc. of the 20th Int'l Symposium on High Performance Distributed Computing (HPDC'11)*, pages 265–266, 2011.

[16] D. Sciascia and F. Pedone. Geo-replicated storage with scalable deferred update replication. In *Proceedings of the 2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops*, SRDSW '14, pages 26–29, Washington, DC, USA, 2014. IEEE Computer Society.

[17] J. a. A. Silva, T. M. Vale, R. J. Dias, H. Paulino, and J. a. M. Lourenço. Supporting multiple data replication models in distributed transactional memory. In *Proc. of the 2015 Int'l Conf. on Distributed Computing and Networking (ICDCN'15)*, pages 11:1–11:10, 2015.

[18] A. Turcu, B. Ravindran, and R. Palmieri. Hyflow2: A high performance distributed transactional memory framework in scala. In *Proc. of the 2013 Int'l Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools (PPPJ'13)*, pages 79–88, 2013.

[19] B. Zhang and B. Ravindran. Relay: A cache-coherence protocol for distributed transactional memory. *Principles of Distributed Systems*, pages 48–53, 2009.