Brief Announcement: Replacement - Handling Failures in a Replicated State Machine

Leander Jehl, Tormod Erevik Lea, and Hein Meling

University of Stavanger, Norway
{leander.jehl,tormod.e.lea,hein.meling}@uis.no

1 Introduction

State machine replication is a common approach for building fault-tolerant services. A Replicated State Machine (RSM) typically uses a consensus protocol such as Paxos [1] to decide on the order of updates and thus keep replicas consistent. Using Paxos, the RSM can continue to process new requests, as long as *more than half of the replicas* remain operational. If this bound is violated, however, the current RSM is forced to stop making progress indefinitely. To avoid scenarios in which the number of failures exceeds the bound, it is beneficial to immediately instantiate failure handling, if this can be done without causing a significant disruption to request execution.

This can be done by reconfiguration, which is a general method to replace one set of replicas with another. Classical reconfiguration relies on the RSM to decide on a reconfiguration command [2]. For this, the old configuration must have a majority of operational replicas and a single correct leader. The latter can only be guaranteed if the replicas are sufficiently synchronized.

In this paper, we present Replacement [3], a reconfiguration algorithm specialized for replacing a faulty replica with a new one. Also Replacement requires a majority of operational replicas. However, different from traditional reconfiguration techniques, failure handling with Replacement does not rely on consensus. Thus, by using Replacement, faulty replicas can be replaced even during times of asynchrony, e.g. when clocks are not synchronized and the network experiences unpredictable delays, or when multiple replicas are competing for leadership. This is useful, since replacing slow or overloaded replicas can restore synchrony and replaced replicas can no longer compete for leadership.

In [4] we showed that reconfiguration without consensus is possible. However, the algorithm presented in [4] (ARec), has to stop the state machine during reconfiguration. Replacement, our new method, includes minor adjustments to the Paxos algorithm that allow the RSM to make progress, while replicas disagree on the current configuration. It thus avoids the increased client latency and temporary unavailability, caused by ARec.

2 Contribution

Replacement is similar to the round change in Paxos. A replacement request, specifying an old replica and its replacements, is propagated to all replicas, which then send PROMISE messages to the new replica. The new replica can determine a correct state and start running Paxos, after collecting a quorum of promises. The following ideas are key to Replacement.

New state only for the new replica. To ensure that no different values can get chosen before and after the replacement, we guarantee that a value, accepted by a majority before replacement, is still accepted by a majority after replacement. For this, it is enough if the new replica stores any possibly accepted value. Therefore, in Replacement, only the new replica needs to wait for promises, while the other replicas can continue to run Paxos.

Vector Timestamps. In Replacement, replicas use a vector clock to timestamp the current configuration. By attaching this vector clock to messages, we can detect and discard messages from replaced replicas. Thus Replacement can allow replicas, that are not replaced, to continue running Paxos in the same round. This is different from other reconfiguration methods [4,5] which enforce a round change in Paxos, and thus discard all messages from the previous round.

Combining Replacements. Every replacement has a unique timestamp and if two concurrent replacements are issued for the same replica, the one with the higher timestamp will be executed. However, if two concurrent replacements are issued for different replicas, both replacements will be executed, possibly in different orders. Thus, replacements for different replicas can be issued by different agents, without the risk that some replacement is lost due to concurrency with another, unrelated replacement.

Since replacements for different replicas are executed concurrently without any order or priority, concurrent replacements can block each other. We solve this with simple coordination among the replacing processes, which is only necessary if a majority of the replicas are replaced concurrently.

Evaluation. Our evaluation shows that using ARec causes longer repair times and temporary unavailability, compared to classical reconfiguration. Replacement performs on par with classical reconfiguration in a synchronous setting, but also allows failure handling in times of asynchrony.

References

1. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. 16(2), 133–169 (1998)
2. Lamport, L., Malkhi, D., Zhou, L.: Reconfiguring a state machine. SIGACT News 41(1), 63–73 (2010)
3. Jehl, L., Meling, H.: Towards fast and efficient failure handling for paxos state machines. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops (ICDCSW), pp. 98–102 (2013)
4. Jehl, L., Meling, H.: Asynchronous Reconfiguration for Paxos State Machines. In: Chatterjee, M., Cao, J.-N., Kothapalli, K., Rajsbaum, S. (eds.) ICDCN 2014. LNCS, vol. 8314, pp. 119–133. Springer, Heidelberg (2014)
5. Lamport, L., Malkhi, D., Zhou, L.: Vertical paxos and primary-backup replication. In: PODC, pp. 312–313 (2009)

Brief Announcement: The Power of Scheduling-Aware Synchronization*

Panagiota Fatourou[1] and Nikolaos D. Kallimanis[2]

[1] FORTH-ICS & University of Crete, Greece
faturu@csd.uoc.gr
[2] FORTH-ICS, Greece
nkallima@ics.forth.gr

We present a new *combining-based* synchronization technique, called Hydra[1], that enables batching, on a single node, of the synchronization requests initiated by threads running on the same core. The technique results in highly-increased *combining degree* (which is the average number of requests that each *combiner* serves), and significantly reduces the number of expensive synchronization *primitives* (like CAS, Swap, Fetch&Add, etc.) performed. We prove that the performance power of Hydra is tremendous when employed in an environment supporting cheap context switching, like user-level threads. Hydra outperforms by far all previous state-of-the-art synchronization algorithms. We experimentally show that the throughput of Hydra is higher than that of CC-Synch, a state-of-the-art (blocking) synchronization protocol presented in PPoPP '12, by more than *an order of magnitude.* Hydra's throughput is surprisingly close to the ideal and this is achieved without increasing the average latency in serving each request.

We also study a simple variant of P-Sim [2], called PSimX, with highly upgraded performance; PSimX is wait-free. The performance of PSimX, albeit lower than that of Hydra, is also close to the ideal. By employing user-level threads in other synchronization protocols, the exhibited performance advantage is much lower than that of Hydra and PSimX. Based on PSimX, it is easy to implement useful *wait-free primitives* (e.g. multi-word `CAS`) at a surprisingly low cost.

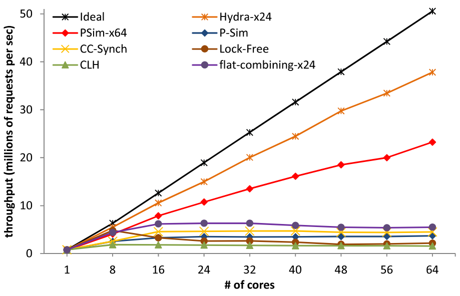
Based on Hydra and PSimX, we implement and experimentally evaluate implementations of concurrent queues and stacks. These implementations outperform by far all current state-of-the-art concurrent queue and stack implementations, respectively. Although the current versions of Hydra and PSimX have been tested in an environment supporting user-level threads, they can also run on top of any threading library, preemptive or not (including kernel threads).

Protocol Description. Hydra maintains a linked list of nodes. Each node of this list stores announced requests of active threads running on the same core c. The first thread p among those running on c, that wants to apply a request,

* This work has been supported by the ARISTEIA Action of the Operational Programme Education and Lifelong Learning which is co-funded by the European Social Fund (ESF) and National Resources through the GreenVM project.

[1] Lernaean Hydra was an ancient monster of Greek mythology that possessed many heads. In Hydra, a processing core (the body) possesses a lot of user threads (the heads).

Alg	throughput	Variant's throughput	speedup
CC-Synch	4.18	4.60	1.10
DSM-Synch	4.10	4.58	1.12
P-Sim	3.90	23.2	5.94
Lock-Free	2.00	1.87	0.94
CLH	1.58	1.7	1.08
FC	2.99	5.51	1.84
OyamaAlg	1.72	2.8	1.63



tries to store a pointer to a node nd in an array A. Other threads running on c may simultaneously compete on the same position of A, so CAS is used.

If p successfully stores nd in A, it records its request in nd, initiates a recording period by informing the other threads running on c that they can start recording requests in nd, and calls Yield. To apply a request, some other thread executing on c, discovers that a recording period is active and records its request in nd. Then, it calls Yield until some combiner serves its request.

Fair scheduling results in the reactivation of p at some later point. Then, p ends the current recording period, executes a Swap to append nd in the shared list, and decides whether it should become a combiner. If p does not become a combiner, it repeatedly calls Yield until a combiner either serves its request or informs p that it is the new combiner. Otherwise, it first serves its own request and then traverses the list and serves the requests recorded in the list nodes, in order, until either it has traversed all elements of the list or it has served up to some constant number of requests. Finally, p informs the process owning the next to traverse node in the list that it is the new combiner.

Performance Evaluation. We evaluated Hydra and PSimX in a 64-core machine consisting of four AMD Opteron 6272 processors (Interlagos). For our experiments, we consider the Fetch&Multiply benchmark used in [1,2]. The figure presents the throughput for the original versions of the evaluated algorithms and their variants where the best number of user level threads per core was employed for each algorithm; all algorithms other than P-Sim and flat-combining (FC) do not exhibit any serious performance gains when employing user level threads. We experimentally compare Hydra with CC-Synch [1], P-Sim [2], flat-combining (Hendler *et. al*, SPAA'10), OyamaAlg (Oyama *et. al*, PDSIA'99), a blocking implementation based on (CLH or MCS) spin-locks, and a simple lock-free implementation. Hydra outperforms CC-Synch by a factor of up to 11 without sacrificing the good latency ensured by CC-Synch. The performance advantages of Hydra over all other algorithms are even higher.

Although the current versions of Hydra and PSimX employ user-level threads, they can also run on top of any threading library, preemptive or not. Hydra and PSimX are *linearizable*. The full paper is provided in [3].

References

1. Fatourou, P., Kallimanis, N.D.: Revisiting the combining synchronization technique. In: Proc. of the 17th ACM Symp. on Principles and Practice of Parallel Programming, pp. 257–266. ACM (2012)
2. Fatourou, P., Kallimanis, N.D.: Highly-Efficient Wait-Free Synchronization. Theory of Computing Systems 53(4), 1–46 (2013)
3. Fatourou, P., Kallimanis, N.D.: The Power of Scheduling-Aware Synchronization. Technical Report TR 442, FORTH ICS, Hellas (2014)

Brief Announcement: Assignment of Different-Sized Inputs in MapReduce*

Foto Afrati[1], Shlomi Dolev[2], Ephraim Korach[2], Shantanu Sharma[2], and Jeffrey D. Ullman[3]

[1] National Technical University of Athens, Greece
[2] Ben-Gurion University of the Negev, Israel
[3] Stanford University, USA

Reducer Capacity. An important parameter to be considered in MapReduce algorithms is the "reducer capacity." A *reducer* is an application of the reduce function to a single *key* and its associated list of *value*s. The *reducer capacity* is an upper bound on the sum of the sizes of the *value*s that are assigned to the reducer. For example, we may choose the reducer capacity to be the size of the main memory of the processors on which the reducers run. We assume that all the reducers have an identical capacity, denoted by q.

Motivation and Examples. We demonstrate a new aspect of the reducer capacity in the scope of several special cases. One useful special case is where an output depends on *exactly* two inputs. We present two examples where each output depends on exactly two inputs and define two problems that are based on these examples.

Similarity-join. Similarity-join is used to find the similarity between any two inputs, *e.g.*, Web pages or documents. A set of m inputs (*e.g.*, Web pages) $WP = \{wp_1, wp_2, \ldots, wp_m\}$, a similarity function $sim(x, y)$, and a similarity threshold t are given, and each pair of inputs $\langle wp_x, wp_y\rangle$ corresponds to one output such that $sim(wp_x, wp_y) \geq t$. It is necessary to compare all pairs of inputs when the similarity measure is sufficiently complex that shortcuts like locality-sensitive hashing are not available. Therefore, it is mandatory to compare every two inputs (Web pages) of the given input set (WP).

Skew join of two relations $X(A, B)$ and $Y(B, C)$. The join of relations $X(A, B)$ and $Y(B, C)$, where the joining attribute is B, provides the output tuples $\langle a, b, c\rangle$, where (a, b) is in X and (b, c) is in Y. One or both of the relations X and Y

* More details appear in the technical report 14-05, Department of Computer Science, Ben-Gurion University of the Negev, Israel, 2014. This work was partially supported by the project Handling Uncertainty in Data Intensive Applications, co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning," under the program THALES, the Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant 428/11), the Israeli Internet Association, and the Ministry of Science and Technology, Infrastructure Research in the Field of Advanced Computing and Cyber Security.

may have a large number of tuples with the same B-value. A value of the joining attribute B that occurs many times is known as a *heavy hitter*. In skew join of $X(A, B)$ and $Y(B, C)$, all the tuples of both the relations with the same heavy hitter should appear together to provide the output tuples.

Problem Statement. We define two problems where exactly two inputs are required for computing an output, as follows: (i) *All-to-All problem.* In the *all-to-all* (*A2A*) problem, a set of inputs is given, and each pair of inputs corresponds to one output. Computing common friends on a social networking site and similarity join are examples. (ii) *X-to-Y problem.* In the *X-to-Y* (*X2Y*) problem, two disjoint sets X and Y are given, and each pair of elements $\langle x_i, y_j \rangle$, where $x_i \in X, y_j \in Y, \forall i, j$, of the sets X and Y corresponds to one output. Skew join and outer product or tensor product are examples.

The *communication cost*, *i.e.*, the total amount of data transmitted from the map phase to the reduce phase, is a significant factor in the performance of a MapReduce algorithm. The communication cost comes with tradeoff in the degree of parallelism however. Higher parallelism requires more reducers (hence, of smaller reducer capacity), and hence a larger communication cost (because the copies of the given inputs are required to be assigned to more reducers). A substantial level of parallelism can be achieved with fewer reducers, and hence, yield a smaller communication cost. Thus, we focus on minimizing the total number of reducers, for a given reducer capacity q. A smaller number of reducers results in a smaller communication cost.

Tradeoffs. The following tradeoffs appear in MapReduce algorithms and in particular in our setting: (i) a tradeoff between the reducer capacity and the total number of reducers, (ii) a tradeoff between the reducer capacity and parallelism, and (iii) a tradeoff between the reducer capacity and the communication cost.

Mapping Schema. A mapping schema is an assignment of the set of inputs to some given reducers under the following two constraints: (i) a reducer is assigned inputs whose sum of the sizes is less than or equal to the reducer capacity, and (ii) for each output, we must assign the corresponding inputs to at least one reducer in common. The following two problems are proved to be NP-compete:

The *A2A Mapping Schema Problem.* An instance of the *A2A mapping schema problem* consists of a set of m inputs whose input size set is $W = \{w_1, w_2, \ldots, w_m\}$ and a set of z reducers of capacity q. A solution to the *A2A mapping schema problem* assigns every pair of inputs to at least one reducer in common, without exceeding q at any reducer.

The *X2Y Mapping Schema Problem.* An instance of the *X2Y mapping schema problem* consists of two disjoint sets X and Y and a set of z reducers of capacity q. The inputs of the set X are of sizes $w_1, w_2, \ldots, w_m$, and the inputs of the set Y are of sizes $w'_1, w'_2, \ldots, w'_n$. A solution to the *X2Y mapping schema problem* assigns every two inputs, the first from one set, X, and the second from the other set, Y, to at least one reducer in common, without exceeding q at any reducer.

Brief Announcement: Scheduling Multiple Objects in Distributed Transactional Memory*

Costas Busch[1], Maurice Herlihy[2], Miroslav Popovic[3], and Gokarna Sharma[1]

[1] Louisiana State University, USA
[2] Brown University, USA
[3] University of Novi Sad, Serbia

Distributed Transactional Memory Model. We consider transactional memory implementations in distributed networked systems, where we provide several performance bounds and impossibility results. A network is modeled as a weighted graph G and each transaction resides at a node and requires one or more shared objects for read or write. We focus on the data-flow model where objects are mobile and the time for an object to traverse an edge is equal to the weight of the edge. In order to guarantee consistency, an object can have only one writable copy in the network at any moment of time. A transaction which is about to execute requires that all requested objects are available at its node.

An execution schedule specifies which transactions execute at any moment of time. The schedule also determines the network paths that the objects will follow while moving from one transaction node to another. We evaluate an execution schedule with two performance metrics: *communication cost*, which is the total distance traversed by all the objects, and *execution time*, which is the total time to execute all transactions. For simplicity, we assume that once a transaction has obtained all requested objects its actual computation time is instantaneous, which implies that the execution time for a set of transactions depends only on the edge traversal times of the requested objects along the followed paths.

Most of the previous works on distributed transactional memory focused on analyzing problem instances with only one shared object. Herlihy and Sun [1] provide a distributed directory approach and the first formal bounds for low doubling dimension metrics. Sharma *et al.* [3] generalize their approach for general network topologies. Zhang *et al.* [4] examine the special case of the work-conserving model with multiple objects and the relation to object TSP tours.

Contributions. We give a comprehensive set of bounds for problem instances where transactions require multiple objects. We assume batch problems where all transactions and their requested objects are known before execution starts. We provide offline schedules for the transactions that have near optimal communication cost. We also provide non-trivial bounds for the execution time, and explore trade-offs between communication cost and execution time. We continue with a description of our detailed contributions.

Communication cost. We first observe that the problem of minimizing the communication cost is NP-hard with a reduction from the graph TSP problem.

* This work is supported by the National Science Foundation grant CCF-1320835.

We then give an upper bound for the communication cost. We use a universal TSP tour to schedule the transactions. A universal TSP tour [2] defines a traversal order for the network nodes so that any subsequence of nodes is also an approximate TSP tour for the respective nodes. By executing the transactions in the order according to the universal TSP tour we guarantee that each object follows an approximate TSP tour of the nodes with the transactions that request the object. The overall schedule has communication cost within $O(\log^4 n/\log\log n)$ factor from optimal, where n is the number of nodes. We obtain better bounds for planar graphs and networks with low doubling metrics.

Execution time. The problem of optimizing the execution time is NP-hard, and it is also hard to approximate it within any factor smaller than the number of transactions (reduction from vertex-coloring). We give an $O(\Delta)$ approximation algorithm for the execution time, where Δ is the maximum number of conflicts between transactions. This bound is obtained with a greedy coloring of a weighted conflict graph of transactions.

An interesting question is whether there are efficient schedules with execution time close to the optimal TSP tours of the objects. We answer this question to the negative, namely, there is a problem instance where each shortest object walk has length $O(n^{5/6})$, while any execution schedule requires time $\Omega(n)$. The same instance has $O(\log n)$ objects per transaction and $\Delta = O(n^{2/3}\log n)$; thus, the $\Omega(n)$ execution time does not follow trivially from other problem parameters. This problem instance demonstrates a significant asymptotic gap between the objects' optimal TSP tour lengths and the execution time.

Time and communication trade-offs. We give a problem instance where it is impossible to simultaneously optimize execution time and communication cost. In this problem instance a lower bound for the execution time is $\Omega(n^{2/3})$ and a lower bound for the communication cost is $\Omega(n)$. We provide two schedules, one with optimal execution time $O(n^{2/3})$, and another schedule with optimal communication cost $O(n)$. We observe that the first schedule has sub-optimal communication cost, while the second schedule has sub-optimal execution time. In fact, any schedule that achieves optimal execution time must have suboptimal communication cost $\Omega(n^{4/3})$. Furthermore, any schedule with optimal communication cost must have suboptimal execution time $\Omega(n)$.

References

1. Herlihy, M., Sun, Y.: Distributed transactional memory for metric-space networks. Distributed Computing 20(3), 195–208 (2007)
2. Jia, L., Lin, G., Noubir, G., Rajaraman, R., Sundaram, R.: Universal approximations for TSP, Steiner tree, and set cover. In: STOC, pp. 386–395 (2005)
3. Sharma, G., Busch, C., Srivathsan, S.: Distributed transactional memory for general networks. In: IPDPS, pp. 1045–1056 (2012), To appear in Distributed Computing
4. Zhang, B., Ravindran, B., Palmieri, R.: Distributed transactional contention management as the traveling salesman problem. In: Halldórsson, M.M. (ed.) SIROCCO 2014. LNCS, vol. 8576, pp. 54–67. Springer, Heidelberg (2014)

Brief Announcement: Relaxing Opacity in Pessimistic Transactional Memory

Konrad Siek and Paweł T. Wojciechowski

Institute of Computing Science
Poznań University of Technology
60-965 Poznań, Poland
{konrad.siek,pawel.t.wojciechowski}@cs.put.edu.pl

Since in the Transactional Memory (TM) abstraction transactional code can contain any operation (rather than just reads and writes), greater attention must be paid to the state of shared variables at any given time. Thus strong safety properties are important in TM, such as opacity [2], virtual world consistency [3], or TMS1/2 [1]. They regulate what values can be read, even by transactions that abort. In comparison to these, properties like serializability allow inconsistent views, so they are relatively weak. However, strong properties virtually preclude early release as a technique for optimizing TM. Early release is a mechanism that allows transactions to read from other transactions, even if the latter are still live. This can increase parallelism, and it is useful in high contention (see e.g., [4]). Thus, we introduce last-use opacity, a safety property that relaxes opacity.

Opacity consists of three core guarantees: serializability, preservation of real-time order, and consistency. We concentrate on the latter, which stipulates that non-local read operations (i.e. those that read values written by other transactions than the current one) must only read values from committed or commit-pending transactions. *Last-use opacity* relaxes this consistency criterion to only provide last-use consistency [7] and recoverability. Then, a transaction can read from another live transaction, if the latter will no longer access the variable in question. Plus, transactions must commit or abort in the order in which they access shared variables. These conditions are defined as follows:

Definition 1 (Commit-pending Equivalence). *Transaction T_i in history H is commit-pending–equivalent with respect to variable x if (a) T_i is live, and (b) there is a read or write operation op on x in $H|T_i$, s.t. for any history H_c for which H is a prefix ($H_c = H \cdot H'$) op is the last read or write on x in $H_c|T_i$.*

Definition 2 (Last-use Consistent Operation). *Given a history H, a transaction T_i and a read operation $op_r = \mathrm{r}(x)v$ on variable x returning v in subhistory $H|T_i$, we say op_r is last-use–consistent as follows: (a) If op_r is local then the latest write operation on x preceding op_r writes value v to x; (b) If op_r is non-local then either $v = 0$ or there is a non-local write operation op_w on variable x writing v in $H|T_k$ ($k \neq i$) where T_k is committed, commit-pending, or commit-pending–equivalent with respect to x.*

Definition 3 (Recoverable Last-use Consistency). *History H is recoverable last-use–consistent if (a) every read operation in $H|T_i$, for every transaction*

T_i in H is last-use–consistent, and (b) for every pair of transactions T_i, T_j such that $i \neq j$ and T_j reads from or writes after T_i, then T_i aborts or commits before T_j aborts or commit, and if T_i aborts, then T_j also aborts.

Relaxing consistency necessarily leads to some inconsistent views to be accepted. Hence, while last-use opacity prevents overwriting (releasing x and writing to it afterwards), it does not prevent zombie transactions—ones that view inconsistent state and are forced to abort. This happens if transaction T_i reads from T_j which, for whatever reason, later aborts. Even if T_i eventually aborts, it operates on stale data and, therefore, can behave unexpectedly. However, this can be rendered harmless by, e.g. sandboxing [5], or enforcing invariants.

On the other hand, using last-use opacity yields performance benefits, especially in high contention. In Fig. 1 we compare two variants of the same distributed TM [6]: last-use–opaque LSVA and opaque OSVA. In all benchmarks LSVA is able to process transactions faster, due to its ability to release early.

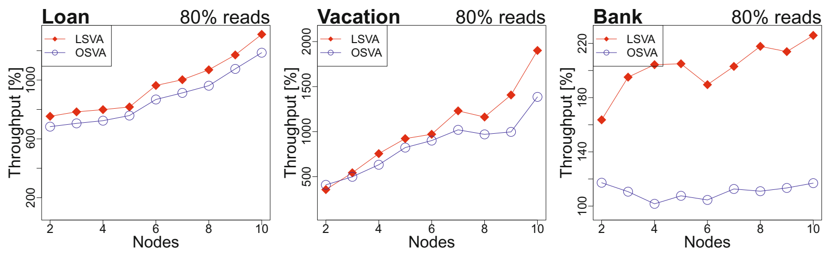


Fig. 1. Percentage improvement relative to a lock-based implementation

Acknowledgments. The project was funded from National Science Centre funds granted by decision No. DEC-2012/06/M/ST6/00463.

References

1. Doherty, S., Groves, L., Luchangco, V., Moir, M.: Towards formally specifying and verifying transactional memory. Formal Aspects of Computing 25 (September 2013)
2. Guerraoui, R., Kapałka, M.: On the Correctness of Transactional Memory. In: Proc. PPoPP 2008 (February 2008)
3. Imbs, D., de Mendivil, J.R., Raynal, M.: On the Consistency Conditions or Transactional Memories. Tech. Rep. 1917, IRISA (December 2008)
4. Ramadan, H.E., Roy, I., Herlihy, M., Witchel, E.: Committing Conflicting Transactions in an STM. In: Proc. PPoPP 2009(February 2009)
5. Scott, M.: Transactional Semantics with Zombies. In: Proc. WTTM 2014 (July 2014)
6. Siek, K., Wojciechowski, P.T.: Atomic RMI: A Distributed Transactional Memory Framework. In: Proc. HLPP 2014 (July 2014)
7. Siek, K., Wojciechowski, P.T.: Zen and the Art of Concurrency Control: An Exploration of TM Safety Property Space with Early Release in Mind. In: Proc. WTTM 2014(July 2014)

Brief Announcement: A Practical Transactional Memory Interface[⋆]

Shahar Timnat[1], Maurice Herlihy[2], and Erez Petrank[1]

[1] Computer Science Department, Technion
[2] Computer Science Department, Brown University

Transactional memory (TM) is becoming an increasingly central concept in parallel programming. Recently, Intel introduced the TSX extensions to the x86 architecture, which include RTM: an off-the-shelf hardware that supports hardware transactional memory. However, there are several reasons for a developer to avoid using hardware transactional memory. First, HTM is only available for some of the computers in the market. Thus, a code that relies on HTM only suits a fraction of the available computers. Second, RTM transactions are "best effort" and are not guaranteed to succeed. Thus, to work with HTM, a *fall-back* path must also be provided, in case transactions repeatedly fail. Namely, developing software using HTM requires three code bases: one based on transactions, a second one for platforms that do not support HTM, and a third code base to handle transaction failures.

We propose a new programming discipline for highly-concurrent linearizable objects that takes advantage of HTM when it is available, and still performs reasonably (around X0.6) when it is not available. We suggest designing data structures using an operation similar to the well-known MCAS(Multi-word Compare And Swap) operation. The MCAS operation executes atomically on several shared memory addresses. Each address is associated with an *expected-value* and a *new-value*. An execution of MCAS succeeds and returns true iff the data in all the addresses is equal to the expected value. In such a case, the data in each address is replaced with the new value. If any of the specified addresses contains data that is different from the expected value, then false is returned and the data in the shared memory remains unchanged. MCAS execution is not supported by common hardware, but there exists an algorithm that implements this operation using standard single-word CASes [4]. Alternatively, MCAS can be easily implemented using transactional memory or by locks.

We propose an extended interface of MCAS called MCMS (Multiple Compare Multiple Swap), in which we also allow addresses to be compared without being swapped. The extension may seem redundant, because, in effect, comparing an address without swapping it is identical to a regular MCAS in which this address' expected value equals its new value. However, when implementing the MCMS using transactional memory, it is ill-advised to write a new (identical) value to replace an old one, since this may cause unnecessary transaction aborts.

In order to study the usability of the MCMS operation, we designed two algorithms that use it. One for the linked-list data structure, and one for the binary search tree. The MCMS tree is almost a straightforward MCMS-based version of the lock-free binary

[⋆] This work was supported by the United States - Israel Binational Science Foundation (BSF) grant No. 2012171. Maurice Herlihy was supported by NSF grant 1331141.

search tree by Ellen et al. [1]. But interestingly, attempting to design a linked-list that exploits the MCMS operation yielded a slightly new algorithm that turns out very efficient also when used with locks. The main idea is to mark a deleted node in a different and useful manner. Instead of using a mark on the reference (like Harris [3]), or using a mark on the reference and additionally a backlink (like Fomitchev and Ruppert [2]), or using a separate mark field (like the lazy linked-list [5]), we mark a node deleted by setting its pointer to be a back-link, referencing the previous node in the list. This approach works excellently with transactions, but can also be used with locks. In fact, a lock-based version of this new algorithm outperforms all known linked-list implementations.

We present three simple fall-back alternatives to enable progress in case RTM executions repeatedly fail. The simplest way is to use locks, in a similar manner to *lock-elision*. The second approach is to use CAS-based MCMS ([4]) as a fall-back. The third alternative is a copying scheme, where a new copy of the data structure is created upon demand to guarantee progress. Both the linked-list and tree algorithm outperform their lock-free alternatives when using either a lock-based fall-back path or a copying fall-back path. The list algorithm performs up to X1.8 faster than Harris's linked-list, and the tree algorithm performs up to X1.2 faster than the tree of Ellen et al. A fall-back path that relies on an MCMS fall-back path is at times a bit faster (up to X1.1) and at times a bit slower than the lock-free alternatives, depending on the specific benchmark and configuration.

Another important advantage of programming with MCMS is that the resulting algorithms are considerably simpler to design and debug compared to standard lock-free algorithms that build on the CAS operation. The stronger MCMS operation allows lock-free algorithms to be designed without requiring complicated "helping" operations typically of lock-free algorithms.

References

1. Ellen, F., Fatourou, P., Ruppert, E., van Breugel, F.: Non-blocking binary search trees. In: PODC (2010)
2. Fomitchev, M., Ruppert, E.: Lock-free linked lists and skip lists. In: PODC 2004, pp. 50–59 (2004)
3. Harris, T.L.: A pragmatic implementation of non-blocking linked-lists. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 300–314. Springer, Heidelberg (2001)
4. Harris, T.L., Fraser, K., Pratt, I.A.: A practical multi-word compare-and-swap operation. In: Malkhi, D. (ed.) DISC 2002. LNCS, vol. 2508, pp. 265–279. Springer, Heidelberg (2002)
5. Heller, S., Herlihy, M., Luchangco, V., Moir, M., Scherer III, W.N., Shavit, N.: A lazy concurrent list-based set algorithm. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 3–16. Springer, Heidelberg (2006)

Brief Announcement: On Dynamic and Multi-functional Labeling Schemes

Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Noy Rotbart

Department of Computer Science, University of Copenhagen
Universitetsparken 5, 2100 Copenhagen
{soerend,knudsen,noyro}@di.ku.dk

1 Introduction

A labeling scheme is a method of distributing the information about the structure of a graph among its vertices by assigning short *labels*, such that a selected function on pairs of vertices can be computed using only their labels. In their seminal paper, Kannan et al. [1] introduced adjacency labeling schemes for trees using at most $2 \log n$ bits for each of the functions adjacency, siblings and ancestry. Alstrup, Bille and Rauhe [2] established a lower bound of $\log n + \log \log n$ for the functions siblings, connectivity and ancestry along with a matching upper bound for the first two. For adjacency, a $\log n + O(\log^* n)$ labeling scheme was presented in [3]. A $\log n + O(\log \log n)$ labeling scheme for ancestry was established only recently by Fraigniaud and Korman [4].

Cohen, Kaplan and Milo [5] considered *dynamic labeling schemes*, where the encoder receives n leaf insertions and assigns unique labels that must remain unchanged throughout the labeling process. In this context, they showed a tight bound of $\Theta(n)$ bits for any dynamic ancestry labeling scheme. In light of this lower bound, Korman, Peleg and Rodeh [6] introduced dynamic labeling schemes, where node re-label is permitted and performed by message passing. In this model they are able to maintain a compact labeling scheme for ancestry, while keeping the number of messages small. Additional results in this setting include conversion methods for static labeling schemes [7], as well as specialized distance [7] and routing [8] labeling schemes.

2 Our Contributions

In the full version [9] we first stress the importance of the lower bound achieved by Cohen et al. [5] by showing that it extends to routing, NCA, and distance. In contrast, we observe that for the dynamic setting, we can achieve efficient labeling schemes for the functions adjacency, sibling, and connectivity without the need of relabeling. More precisely, we observe that the original $2 \log n$ adjacency labeling scheme due to Kannan et al. [1] is in fact suitable for the dynamic setting. Moreover, the original labeling scheme also supports sibling queries and a slightly modified scheme is shown to work for connectivity. Our findings reveal an exponential gap between ancestry and the functions mentioned for the dynamic setting.

We then present various families of insertion sequences for which labels of size $2\log n$ are required for each of the functions. This suggest that in the dynamic setting the original labeling schemes are in fact optimal, and contrast the static case, where adjacency labeling schemes requires strictly fewer bits than both sibling and connectivity. We prove the lower bound by showing a family of n insertion sequences that requires $O(n^2)$ distinct labels, as illustrated in Fig. 1.

Many other graph families enjoy (static) adjacency labeling schemes of size $O(\log n)$. Among those, we mention graphs with bounded arboricity, graphs of bounded treewidth and interval graphs. We show simple lower bounds of $\Omega(n)$ for dynamic adjacency labeling schemes for those families.

Multi-functional labeling schemes. In this context, we show the following results. First, we prove that $3\log n$ bits are necessary and sufficient for any dynamic labeling scheme supporting adjacency and connectivity. Interestingly, the same gap appears in the static setting where we prove that $\log n + 2\log\log n$ bits are sufficient and necessary for any unique labeling scheme supporting both connectivity and siblings/ancestry, in contrast to $\log n + \log\log n$ [2] for each function individually.

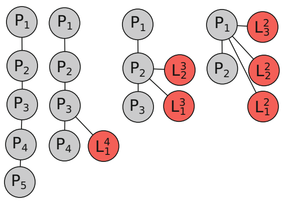


Fig. 1. The lower bound construction for adjacency dynamic labeling schemes. The red nodes are the ones that must be labeled with distinct labels.

References

1. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. SIAM Journal on Discrete Mathematics, 334–343 (1992)
2. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. SIAM J. Discret. Math. 19(2), 448–462 (2005)
3. Alstrup, S., Rauhe, T.: Small induced-universal graphs and compact implicit graph representations. In: FOCS 2002, pp. 53–62 (2002)
4. Fraigniaud, P., Korman, A.: An optimal ancestry scheme and small universal posets. In: STOC 2010, pp. 611–620 (2010)
5. Cohen, E., Kaplan, H., Milo, T.: Labeling dynamic xml trees. SIAM Journal on Computing 39(5), 2048–2074 (2010)
6. Korman, A., Peleg, D., Rodeh, Y.: Labeling schemes for dynamic tree networks. Theory of Computing Systems 37(1), 49–75 (2004)
7. Korman, A.: General compact labeling schemes for dynamic trees. Distributed Computing 20(3), 179–193 (2007)
8. Korman, A.: Compact routing schemes for dynamic trees in the fixed port model. In: Garg, V., Wattenhofer, R., Kothapalli, K. (eds.) ICDCN 2009. LNCS, vol. 5408, pp. 218–229. Springer, Heidelberg (2009)
9. Dahlgaard, S., Knudsen, M.B.T., Rotbart, N.: Dynamic and multi-functional labeling schemes, arXiv preprint arXiv:1404.4982

Brief Announcement: Update Consistency in Partitionable Systems

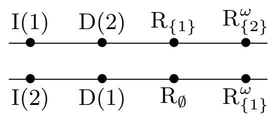
Matthieu Perrin, Achour Mostéfaoui, and Claude Jard

LINA – University of Nantes, 2 rue de la Houssinière, 44322 Nantes Cedex 3, France
{matthieu.perrin,claude.jard,achour.mostefaoui}@univ-nantes.fr,

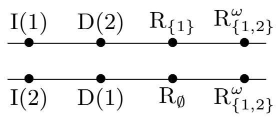
Data replication is essential to ensure reliability, availability and fault-tolerance of massive distributed applications over large scale systems such as the Internet. However, these systems are prone to partitioning, which by Brewer's CAP theorem [1] makes it impossible to use a strong consistency criterion like atomicity. Eventual consistency [2] guaranties that all replicas eventually converge to a common state when the participants stop updating. However, eventual consistency fails to fully specify shared objects and requires additional non-intuitive and error-prone distributed specification techniques, that must take into account all possible concurrent histories of updates to specify this common state [3]. This approach, that can lead to specifications as complicated as the implementations themselves, is limited by a more serious issue. The concurrent specification of objects uses the notion of *concurrent events*. In message-passing systems, two events are concurrent if they are enforced by different processes and each process enforced its event before it received the notification message from the other process. In other words, the notion of concurrency depends on the implementation of the object, not on its specification. Consequently, the final user may not know if two events are concurrent without explicitly tracking the messages exchanged by the processes. A specification should be independent of the system on which it is implemented.

We believe that an object should be totally specified by two facets: its abstract data type, that characterizes its sequential executions, and a consistency criterion, that defines how it is supposed to behave in a distributed environment. Not only sequential specification helps repeal the problem of intention, it also allows to use the well studied and understood notions of languages and automata. This makes possible to apply all the tools developed for sequential systems, from their simple definition using structures and classes to the most advanced techniques like model checking and formal verification.

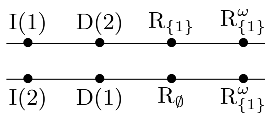
Eventual consistency (EC) imposes no constraint on the convergent state, that very few depends on the sequential specification. For example, an implementation that ignores all the updates is eventually consistent, as all replicas converge to the initial state. We propose *update consistency* (UC), a new consistency criterion in which the convergent state must be obtained by a total ordering of the updates that contains the sequential order of each process. Another equivalent way to approach it is that, if the number of updates is finite, it is possible to remove a finite number of queries such that the remaining history is sequentially consistent. Unlike Fig. 1a, Fig. 1b presents an eventually consistent history, as both processes read $\{1, 2\}$ once they have converged. However, it is not update



(a) Not EC and not UC



(b) EC but not UC



(c) EC and UC

Fig. 1. Three histories for a set of integers, with different consistency criteria. An event labeled ω is repeated infinitely often.

consistent: in any linearization of the updates, a deletion must appear as the last update, so this history cannot converge to state $\{1, 2\}$. State $\{1\}$ is possible because the updates can be done in the order $\mathrm{I}(2), \mathrm{D}(1), \mathrm{I}(1), \mathrm{D}(2)$, so Fig. 1c, is update consistent. As update consistency is strictly stronger than eventual consistency, an update consistent object can always be used instead of its eventually consistent counterpart.

We can prove that update consistency is universal, in the sense that every object has an update consistent implementation in a partitionable system, where any number of crashes are allowed. The principle is to build a total order on the updates on which all the participants agree, and then to rewrite the history *a posteriori* so that every replica of the object eventually reaches the state corresponding to the common sequential history. Any strategy to build the total order on the updates would work. For example, this order can be built from a timestamp made of a Lamport's clock [4] and the id of the process that performed it. The genericity of the proposed algorithm is very important because it may give a substitute to composability. Composability is an important property of consistency criteria because it allows to program in a modular way, but it is very difficult to achieve for consistency criteria. A same algorithm that pilots several objects during a same execution allows this execution to be update consistent. This universality result allows to imagine automatic compilation techniques that compose specifications instead of implementations.

References

1. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News 33, 51–59 (2002)
2. Vogels, W.: Eventually consistent. Queue 6, 14–19 (2008)
3. Burckhardt, S., Gotsman, A., Yang, H., Zawirski, M.: Replicated data types: Specification, verification, optimality. In: Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 271–284. ACM (2014)
4. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21, 558–565 (1978)

Brief Announcement: Breaching the Wall of Impossibility Results on Disjoint-Access Parallel TM

Sebastiano Peluso[1,2,3], Roberto Palmieri[1], Paolo Romano[2], Binoy Ravindran[1], and Francesco Quaglia[3]

[1] Virginia Tech, Blacksburg, VA, USA
{peluso,robertop,binoy}@vt.edu
[2] IST/INESC-ID, Lisbon, Portugal
{peluso,romanop}@gsd.inesc-id.pt
[3] Sapienza University, Rome, Italy
{peluso,quaglia}@dis.uniroma1.it

Abstract. Transactional Memory (TM) implementations guaranteeing disjoint-access parallelism (DAP) are desirable on multi-core architectures because they can exploit low-level parallelism. In this paper we look for a breach in the wall of existing impossibility results on DAP TMs, by identifying the strongest consistency and liveness guarantees that a DAP TM can ensure while maximizing efficiency in read-dominated workloads. Along the path of designing this protocol, we report two impossibility results related to ensuring real-time order in a DAP TM.

Keywords: Transactional Memory, Disjoint-Access Parallelism, Real-Time Order.

1 Overview of the Achieved Results

A property that is deemed as crucial for the scalability of a TM is its ability to avoid any contention on shared objects, also called *base objects*, among transactions that access disjoint data sets – *disjoint-access parallelism* (or DAP) [1]. Also, since many real-world workloads are often read-dominated, another aspect with strong impact on performance of TM algorithms is optimizing the processing of read-only transactions. In this sense, two main properties are regarded as particularly important for read-only transactions: wait-freedom, i.e. transactions are never blocked or aborted (WFRO), and invisible reads, i.e. read operations never update any datum or base object (IRO). We succinctly denote their union as WFIRO.

Given the set of impossibility results related to implementing TM algorithms that guarantee different variants of the DAP property, as well as alternative consistency and liveness criteria [1,2,3], in this paper we find a breach in this wall of impossibility results, seeking an answer to the following question: what are the strongest *consistency* and *liveness* guarantees that a TM can ensure while remaining scalable — by ensuring DAP — and maximizing efficiency in read-dominated workloads — by having WFIRO? Our search space considers the Cartesian product of the consistency criteria specified by Adya's hierarchy [4] and of a set of liveness properties that comprises both TM-specific criteria [5], as well as classical progress criteria, i.e. obstruction-, lock- and wait-freedom.

Along the path that leads us to answer the above question, we also prove two novel impossibility results. If one selects *any* consistency criterion that ensures Real Time Order (RTO), i.e. by ensuring that transactions appear as executed without reversing the partial order defined by non-concurrent transactions, and independently of the isolation guarantees for concurrent transactions, it is impossible to ensure also WFRO, obstruction-free update transactions and the weakest form of DAP [1]. Further, even assuming weakly progressive update transactions [5], we are still faced with an impossibility result if we want IRO.

These results highlight the necessity of relaxing RTO to implement a scalable TM that maximizes the efficiency of read-only transactions by jointly guaranteeing DAP and WFIRO. This leads us to introduce a weaker variant of RTO, named *Witnessable Real Time Order* (WRTO), which demands that the RTO is enforced only among transactions exhibiting (transitive) data conflicts.

By adopting WRTO, we design a WFIRO TM that guarantees the strongest variant of DAP [2], strong progressiveness [5] and a consistency criterion whose semantics is very close to those provided by popular safety properties for TM, such as Opacity. This consistency criterion, known as Extended Update Serializability (EUS) [4,6] guarantees the serializability of the history of committed update transactions. Further, EUS ensures that all transactions (also transactions that eventually abort) observe a snapshot producible by some equivalent serialization of the history of (committed) update transactions.

Acknowledgments. This work is supported in part by US NSF under grant CNS-1217385 and by FCT via grants PEst-OE/EEI/LA0021/2013 and EXPL/EEI-ESS/0361/2013.

References

1. Attiya, H., Hillel, E., Milani, A.: Inherent Limitations on Disjoint-Access Parallel Implementations of Transactional Memory. J. Theory Comput. Syst. 49(4), 698–719 (2011)
2. Guerraoui, R., Kapalka, M.: On Obstruction-free Transactions. In: 20th Annual Symposium on Parallelism in Algorithms and Architectures, pp. 304–313. ACM, New York (2008)
3. Bushkov, V., Dziuma, D., Fatourou, P., Guerraoui, R.: The PCL Theorem. Transactions cannot be Parallel, Consistent and Live. In: 26th Annual Symposium on Parallelism in Algorithms and Architectures, pp. 178–187. ACM, New York (2014)
4. Adya, A.: Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions. PhD Thesis. MIT (1999)
5. Guerraoui, R., Kapalka, M.: The Semantics of Progress in Lock-based Transactional Memory. In: 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 404–415. ACM, New York (2009)
6. Peluso, S., Ruivo, P., Romano, P., Quaglia, F., Rodrigues, L.: When Scalability Meets Consistency: Genuine Multiversion Update-Serializable Partial Data Replication. In: 32nd IEEE International Conference on Distributed Computing Systems, pp. 455–465. IEEE Computer Society, Washington, DC (2012)

Brief Announcement: COP Composition Using Transaction Suspension in the Compiler

Hillel Avni[1] and Adi Suissa-Peleg[2]

[1] Ben Gurion University
hillel.avni@gmail.com
[2] Harvard University
adisuis@seas.harvard.edu

Abstract. Combining a number of transactions into a single atomic transaction is an important transactional memory (TM) feature supported by many software TM (STM) implementations. This composition, however, typically results in long transactions with an increased contention probability.

In consistency oblivious programming (COP), the read-only prefix of a data structure operation is performed outside of a TM transactional context. The operation is then completed by using a transaction that verifies the prefix output and performs updates. In STM, this strategy effectively reduces much of the overhead and potential contention.

In this work we emphasize the importance of *transaction-suspension*, which enables performing non-transactional memory accesses inside a transaction. Suspension not only simplifies the use of COP, but also enables the composition of a sequence of COP-based operations into a single transaction. We add transaction-suspension support to GCC-TM, and integrate COP into TM applications. We also support TM-Safe memory reclamation in transactions with COP operations, by adding privatization before a transaction abort to the GCC-TM library.

Introduction. Consistency Oblivious Programming (COP) [2], is a programming methodology for improving a TM-based data structure performance. In COP, the read-only prefix (ROP) of a data structure operation is performed in a non-transactional context. The operation is then completed by using a transaction that verifies the ROP output and performs updates. COP-based data structures effectively reduce much of the TM instrumentation overhead and potential contention.

The ROP may observe inconsistent states, and must avoid crashing as a result. It is the responsibility of the programmer to keep the ROP from hitting infinite loops or uninitialized pointers. Another type of crash may be caused by an ROP code segment that accesses a memory location after it was released by a concurrent transaction. To prevent this scenario we modified the privatization algorithm in the STM.

A useful feature supported by many STM implementations is transactions composability, the ability to combine a number of transactional atomic blocks to be executed in a single transaction. This fosters the use of TM-based data structures, and facilitates the creation of non-trivial atomic transactions that access different data structures.

In this paper, we introduce a methodology that uses GCC-TM, the GNU C Compiler (GCC) [1] STM implementation, to support efficient and natural composition of COP operations. Our methodology is based on *transaction-suspension*, which enables executing non-transactional, non-instrumented instructions inside a transactional block. In order to support a suspension of a transaction in GCC-TM, we mark functions with the TM-Pure attribute[1] [4], that omits the instrumentation of these functions when called from transactions. We apply our methodology to the linked list and red-black tree, that are part of the data-structures library which is used by the STAMP applications. Our results show that this mechanism reduces 80% of the aborts caused by conflicts.

COP Composing Using Suspended Transactions. When using transaction-suspension, a COP operation, OP, embedded in a transaction T, goes through the following steps:

$\mathrm{T}_{start} \rightarrow$Any code$\rightarrow T_{suspend} \rightarrow \mathrm{OP}_{rop} \rightarrow \mathrm{T}_{resume} \rightarrow \mathrm{OP}_{verify} \rightarrow \mathrm{OP}_{updates} \rightarrow$Any code$\rightarrow \mathrm{T}_{end}$

OP_{verify} should verify the validity of the data gathered during the ROP code. This code is executed locally and must be concise, so that it does not introduce additional overhead.

In addition, note that OP_{rop} can be executed several times in non-transactional, suspended mode, and only if verification failure persists, it should fallback to transactional mode. If the transactional execution of the ROP, i.e., the fallback, aborts, the transaction naturally aborts.

The only way to compose COP operations without transaction-suspension, is the one proposed by [5], i.e., execute all ROP parts of the composed operations before starting the transaction, then, inside the transaction, verify their output and complete the transactions updates. This method allows composition only if an operation is not writing data that may later be accessed by another COP operation in the same transaction.

Safe Memory Reclamation. Two important functions that are TM-Safe [4], i.e., can be executed inside a transaction, are *malloc* and *free*. These functions are made safe by privatization. If transaction T wrote to memory, then before it commits, it waits for the termination of the transactions that started before its commit [3]. As a side effect of privatization, in case T detaches some memory block from a data structure and successfully commits, then T can free that block.

On the other hand, if T allocates some block of memory, M, and then aborts, it can free M without privatization. The reason is that the pointer to the tentative memory block is not exposed to other transactions.

[1] A function that is marked with the TM-Pure attribute is executed as a non-transactional code block. The TM-Pure attribute is supported by the GCC-TM implementation.

This is violated when COP is involved. If the non-transactional ROP code block traverses the data structure, it may acquire a pointer to a newly allocated memory block, and upon an abort of T and freeing M, the ROP may try to access unmapped memory. To prevent this scenario, we added privatization to writing transactions that are about to perform rollback. If a transaction is a read-only transaction, it can free its tentative memory blocks unconditionally. If, however, the transaction updated some memory location, it has to perform privatization as if it was successfully committed. Our evaluation showed that this privatization has a negligible impact on performance.

With the suspended mode, and rollback privatization, malloc and free become also COP safe. The reason is that memory is not recycled as long as there is a transaction in progress, and the COP operations are always encapsulated in transactions. One restriction is that allocation cannot take place in a ROP, because, in case the validation fails, the allocated memory will not be freed, as we do not abort the transaction in this case. However, as the ROP code typically avoids from writing to memory, it does not need to allocate or free memory.

References

1. Gcc version 4.7.0 (April 2012), `http://gcc.gnu.org/gcc-4.7/`
2. Afek, Y., Avni, H., Shavit, N.: Towards consistency oblivious programming. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 65–79. Springer, Heidelberg (2011)
3. Dice, D., Matveev, A., Shavit, N.: Implicit privatization using private transactions. In: TRANSACT (2010)
4. Riegel, T.: Software Transactional Memory Building Blocks. PhD thesis, Technischen Universitat Dresden, geboren am 1.3.1979 in Dresden (March 2013)
5. Xiang, L., Scott, M.L.: Composable partitioned transactions. In: WTTM (2013)

Brief Announcement: Non-blocking Monitor Executions for Increased Parallelism*

Wei-Lun Hung, Himanshu Chauhan, and Vijay K. Garg

The University of Texas at Austin
{wlhung@,himanshu@,garg@ece.}utexas.edu

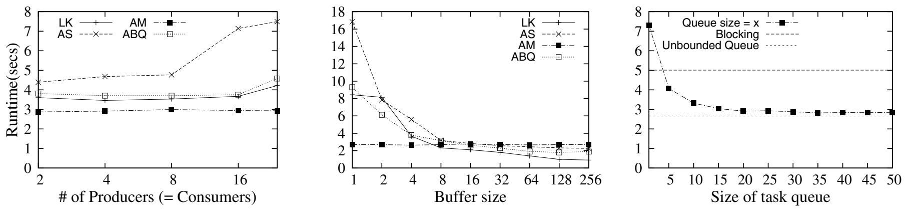
Motivation and Approach: Monitors are a prevalent programming technique for thread synchronization in shared-memory parallel programs. The current design of monitors uses the *wait/notification* mechanism that blocks threads from executing without exclusive access to critical sections. We explore the idea of allowing non-blocking executions of monitor methods to improve the collective worker thread throughput and cache-locality in multi-threaded programs.

Our proposed framework, called *ActiveMonitor*, uses the concept of *futures* [1,2] to provide non-blocking monitors by creating: (i) an *executor* for every monitor object (similar to remote-core-locking [3]), and (ii) *tasks* — equivalent to monitor methods — that are submitted to the executors. Our framework handles these steps automatically. The framework allows the programmer to use the keyword 'nonblocking' in signatures of monitor methods to make their execution non-blocking. Non-blocking methods return a *future* reference, which can be used to retrieve the result of method invocation. We re-interpret linearizability in this context, and enforce two rules to guarantee correctness: **(a)** all the tasks submitted to one monitor executor are processed in FIFO order. **(b)** tasks corresponding to a worker thread's invocations of methods on different monitors are processed in program order (of the worker thread). See [4] for details.

Evaluation: We present the performance evaluation of our approach for two monitor-based problems in Java. In our benchmark, each worker (thread) performs 512000 operations on shared data protected by monitors. We vary the number of workers from 2 to 24 on a 24-way machine, and measure the time required for all the workers to complete their operations.

1. Bounded-Buffer Problem: Every producer's put invocation is non-blocking, and every consumer's take is blocking. Items are plain objects. We also compare runtimes of Java's ArrayBlockingQueue based implementation (denoted by ABQ). We collect runtimes by varying: (a) number of workers for a fixed buffer-size (=4). (b) buffer-size for fixed number of producers/consumers (=16 each). (c) limit on non-blocking tasks allowed for fixed buffer-size (=4), and 16 producers and consumers each. Fig. 1 shows the results of these three experiments. Across all results, we use these legends for implementation techniques: LK: Java Reentrant locks, AS: *AutoSynch* [5], AM: *ActiveMonitor* (this paper).
2. Sorted Linked-List Problem: Worker threads insert or remove, with equal

* Supported in part by NSF Grants CNS-1346245, CNS-1115808, and Cullen Trust.

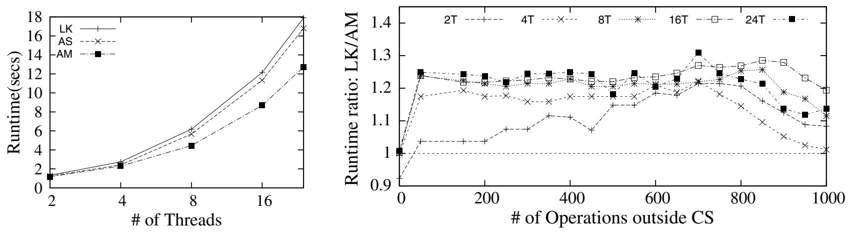


(a) Buffer-size = 4 **(b)** Varying buffer-size **(c)** Varying tasks-queue size

Fig. 1. Runtimes (mean values across 25 runs) for bounded-buffer

probability, random integer values on a pre-populated linked-list of integers that is sorted in non-decreasing order. Both insert and remove operations are non-blocking. Each worker thread also performs some local operations outside the critical section (CS) between successive updates to the list. We collect the runtimes by varying: (a) number of workers, keeping local operations outside CS/worker fixed at 250. (b) number of workers as well as number of local operations outside CS. The results of these two experiments are shown in Fig. 2.

See [4] for extended evaluation on other monitor problems, details of CPU and memory consumption, and comparison with other implementation techniques.



(a) Varying # of workers **(b)** Varying # of workers, and ops outside CS

Fig. 2. Results (mean values across 25 runs) for sorted linked-list

References

1. Halstead, R.H.: Multilisp: A language for concurrent symbolic computation. ACM Trans. Program. Lang. Syst. 7(4), 501–538 (1985)
2. Kogan, A., Herlihy, M.: The future(s) of shared data structures. In: PODC (2014)
3. Lozi, J.-P., et al.: Remote core locking: Migrating critical-section execution to improve the performance of multithreaded applications. In: USENIX Annual Technical Conference, pp. 65–76 (2012)
4. `http://arxiv.org/abs/1408.0818`
5. Hung, W.-L., Garg, V.K.: AutoSynch: An Automatic-signal Monitor Based on Predicate Tagging. In: PLDI, pp. 253–262 (2013)

Brief Announcement: Agreement in Partitioned Dynamic Networks

Adam Sealfon and Aikaterini Sotiraki

Massachusetts Institute of Technology
{asealfon,katesot}@mit.edu

Abstract. In the dynamic network model, the communication graph is assumed to be connected in every round but is otherwise arbitrary. We consider the related setting of *p-partitioned dynamic networks*, in which the communication graph in each round consists of at most p connected components. We explore the problem of k-agreement in this model for $k \geq p$. We show that if the number of processes is unknown then it is impossible to achieve k-agreement for any k and any $p \geq 2$. Given an upper bound N on the number of processes, we provide algorithms achieving k-agreement in $p(N-p-1)+1$ rounds for $k = p$ and in $O(N/\epsilon)$ rounds for $k = \lceil(1+\epsilon)p\rceil$.

Keywords: distributed algorithms, dynamic networks, agreement, partitioned networks.

Dynamic graphs are a model for distributed algorithms which were introduced by Kuhn, Lynch and Oshman [1]. In this paper we explore the capabilities and limitations of a modification to the dynamic graph model addressing additional challenges arising in wireless communication.

In the dynamic graph model, the network is assumed merely to be connected in each round, with no additional assumptions about consistency from round to round. We weaken this assumption further, allowing the network to consist of more than one connected component. Formally,

Definition 1. *A dynamic graph $G = (V, E)$ is said to be* p-partitioned *if at each round t, it consists of at most p connected components.*

Processes communicate in synchronous rounds using local broadcast. The edges in each round are chosen by an adaptive adversary.

In this setting, many of the problems previously considered in the dynamic network model cannot be solved. In particular, tasks such as token dissemination, leader election and consensus which cannot be solved in partitioned static networks clearly are also impossible in partitioned dynamic networks. We consider the problem of k-agreement for constant k, which can be solved in static p-partitioned networks as long as $k \geq p$. The conditions for k-agreement are the following:

1. Agreement: All decision values are in W, where W is a subset of the initial values with $|W| = k$.

2. Validity: Any decision value is the initial value of some process.
3. Termination: All processes eventually decide.

We show that k-agreement is not possible in the setting of p-partitioned dynamic networks if the number of processes is unknown, but that it can be achieved for any $k \geq p$ given an upper bound on the number of processes. Our results are qualitatively different from the case of ordinary dynamic networks, for which there are known consensus protocols which do not assume knowledge of the size of the network [2].

Theorem 1. *For all $p \geq 2$, $k \geq 1$ there is no algorithm which will solve k-agreement on p-partitioned dynamic graphs given no information about the size of the network.*

Theorem 2. *For any $p \geq 1$, we can solve p-agreement in $p(N - p - 1) + 1$ rounds on any p-partitioned dynamic graph, where N is a known upper bound on the number of vertices.*

Theorem 3. *For any $\epsilon > 0, p \geq 1$, we can solve $\lceil (1+\epsilon)p \rceil$-agreement in $O(N/\epsilon)$ rounds on any p-partitioned dynamic graph, where N is a known upper bound on the number of vertices.*

Our results apply to both undirected and directed graphs. More details and the complete proofs can be found in [3].

It would be interesting to consider whether it is possible to achieve agreement in fewer rounds in a p-partitioned dynamic network. Our algorithms solve $\lceil (1 + \epsilon)p \rceil$-agreement in $O(N/\epsilon)$ rounds and p-agreement in $p(N-p-1)+1$ rounds. It is unclear whether this dependence on p is intrinsic or whether p-agreement can be achieved in $O(N)$ rounds regardless of p. It would also be interesting to explore whether p-agreement can be achieved in fewer rounds with high probability against a nonadaptive adversary.

We have shown that it remains possible to solve nontrivial problems under the weaker assumption that the network at each round consists of at most p connected components. It remains open what additional problems can be solved in this model.

Acknowledgments. We would like to thank Mohsen Ghaffari and Nancy Lynch for helpful discussions. This material is based upon work supported in part by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374.

References

1. Kuhn, F., Lynch, N., Oshman, R.: Distributed Computation in Dynamic Networks. In: Proc. 42nd ACM Symp. on Theory of Computing, STOC (2010)
2. Oshman, R.: Distributed Computation in Wireless and Dynamic Networks. PhD Thesis (2012)
3. Sealfon, A., Sotiraki, A.: Agreement in Partitioned Dynamic Networks. CoRR, abs/1408.0574 (2014)

Brief Announcement: The 1-2-3-Toolkit for Building Your Own Balls-into-Bins Algorithm

Pierre Bertrand[1] and Christoph Lenzen[2]

[1] Ecole Normale Suprieure Cachan
Avenue du prsident Wilson, 94230 Cachan
pierre.bertrand@ens-cachan.fr
[2] MPI for Informatics
Campus E1 4, 66123 Saarbrcken
clenzen@mpi-inf.mpg.de

Abstract. We examine a generic class of simple distributed balls-into-bins algorithms and compute accurate estimates of the remaining balls and the load distribution after each round. Each algorithm is classified by (i) the load that bins accept in a given round and (ii) the number of messages each ball sends in a given round. Our algorithms employ a novel ranking mechanism resulting in notable improvements. Simulations independently verify our results and their high accuracy.

1 Problem and Algorithm

Consider a distributed system of n anonymous balls and n anonymous bins, each having access to (perfect) randomization. Communication proceeds in synchronous rounds, each of which consists of the following steps.

1. Balls perform computations and send messages to bins.
2. Bins receive them, perform computations, and respond to received messages.
3. Each ball may commit to a bin, inform it, and terminate.

The main goals are to minimize the maximal number of balls committing to the same bin, the number of rounds, and the number of messages. This fundamental load balancing task has a wide range of applications, cf. [5].

Today, we understand the asymptotics of this problem very well [3,4,6]. However, lower and upper bounds have in common that they are not very precise. Arguably, with running time bounds like, e.g., $\Theta(\log\log n/\log\log\log n)$ or $\log^* n + \mathcal{O}(1)$, the involved constants are essential. In this work, we provide a simple, yet accurate analysis of a general class of algorithms. We introduce a novel *ranking* mechanism, resulting in superior performance.

Concretely, in each round $i \in \mathbb{N}$, the following steps are executed.

1. Each ball sends $M_i \in \mathbb{N}$ messages to uniformly independently random (u.i.r.) bins. These messages carry ranks $1, \dots, M_i$.
2. A bin of current load ℓ responds to (up to) $L_i - \ell$ balls, where smaller ranks are preferred. Ties are broken by choosing u.i.r.
3. Each ball that receives a response commits to the responding bin to which it sent the message of smallest rank.

2 Techniques and Results

Applying Chernoff's bound, it is not hard to show that the number of bins with a given load and the number of remaining balls are strongly concentrated around the expected values. With high probability, the error resulting from assuming that these expected values are matched exactly is hence negligible. Using this argument (and the union bound) repeatedly, we can infer that it suffices to compute expected values, approximating the true distribution by expected values. We complement the derived analytical results by simulations, confirming that the deviations are indeed very small. Moreover, we use the simulations to compare to other algorithms from the literature.

Table 1. Evaluated specific scenarios (analytical and simulation results match)

goal	rounds	max. load	messages	exp. fraction of balls left	L	M
small load	3	2	$< 5.5n$	$< 6 \cdot 10^{-7}$	$(2,2,2)$	$(2,5,5)$
few rounds	2	3	$< 5.5n$	$< 6 \cdot 10^{-10}$	$(2,3)$	$(2,5)$
few messages	3	3	$< 3.5n$	$< 5 \cdot 10^{-8}$	$(2,3,3)$	$(1,2,2)$
safe termination	3	3	$< 3.85n$	$< 6 \cdot 10^{-19}$	$(2,2,3)$	$(1,4,5)$

Our simulations also show that the proposed algorithms compare favorably with all previous ones from the literature. The full paper, comprising a discussion of related work, the derivation of the analytical bounds, and details on the simulation results, is available on arxiv [2]. The used code can be found online [1].

Acknowledgements. Christoph Lenzen has been supported by the Deutsche Forschungsgemeinschaft (DFG, reference number Le 3107/1-1).

References

1. Bertrand, P.: Python scripts used for simulations and computations (2014), `http://people.mpi-inf.mpg.de/~clenzen/babi/`
2. Bertrand, P., Lenzen, C.: The 1-2-3-Toolkit for Building Your Own Balls-into-Bins Algorithm. Computing Research Repository abs/1407.8433 (2014)
3. Even, G., Medina, M.: Parallel Randomized Load Balancing: A Lower Bound for a More General Model. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 358–369. Springer, Heidelberg (2010)
4. Lenzen, C., Wattenhofer, R.: Tight Bounds for Parallel Randomized Load Balancing: Extended Abstract. In: Proc. 43rd Symposium on Theory of Computing (STOC), pp. 11–20 (2011)
5. Mitzenmacher, M., Richa, A., Sitaraman, R.: The Power of Two Random Choices: A Survey of the Techniques and Results. In: Handbook of Randomized Computing, vol. 1, pp. 255–312. Kluwer Academic Publishers, Dordrecht (2001)
6. Stemann, V.: Parallel Balanced Allocations. In: Proc. 8th Symposium on Parallel Algorithms and Architectures (SPAA), pp. 261–269 (1996)

Brief Announcement: k-Selection and Sorting in the SINR Model

Stephan Holzer[1,⋆], Sebastian Kohler[2], and Roger Wattenhofer[2]

[1] Massachusetts Institute of Technology (MIT), Cambridge, USA
holzer@mit.edu
[2] ETH Zurich, Zurich, Switzerland
sebastian.kohler@alumni.ethz.ch, wattenhofer@ethz.ch

Abstract. We study algorithms and lower bounds for k-selection and sorting in the signal-to-interference-plus-noise-ratio (SINR) model. For the problem of finding the k-th smallest value in the network, we provide a $\mathcal{O}(\log^2 n)$ algorithm based on the aggregation trees presented in [2]. We argue that any algorithm using this approach has runtime $\Omega(\log^2 n/\log\log n)$. We show that sorting can be done in time $\Theta(n)$.

1 Model and Preliminaries

In the SINR model [1,5] we consider a set $V := \{v_1, v_2, \ldots, v_n\}$ of $n := |V|$ nodes in the Euclidean plane. Each node $v \in V$ has a unique ID $\mathrm{id}_v \in \{1, \ldots, n\}$ and is given an arbitrary input value $x_v \in [W]$ for some $W \in \mathcal{O}(poly(n))$. Time is slotted into discrete *time steps* of equal length and every node wakes up at the same time. Local computation does not count towards the complexity-measure as we are interested in communication complexity. Communication bandwidth is limited, only one message containing $\Theta(1)$ values from $[n]$ and $\Theta(1)$ values from $[W]$ can be sent/received by a node in a single time step. In each time step, each node $v \in V$ can choose an arbitrary *transmission power* $P_v \geq 0$. A message sent by a node s is received by node r if $P_r = 0$ and the received SINR at r exceeds a constant threshold $\beta > 1$, i.e., the SINR condition $\frac{P_s/d(s,r)^\alpha}{\sum_{s' \in V\setminus\{s\}} P_{s'}/d(s',r)^\alpha + N} \geq \beta$ is satisfied. Here, $\alpha > 2$ is the constant *path-loss exponent* and $N \geq 0$ is the *ambient noise*. The positions of the nodes, their IDs, n and W are known to all nodes. A probabilistic event A happens *with high probability* if $\Pr[A] \geq 1 - 1/n$.

2 Algorithms and Lower Bounds

We start with a sketch of an algorithm for k-selection (finding the k-th smallest value in the network). We use the construction of a minimum-latency aggregation schedule (MLAS) presented in Section 7.1 in [2], which is based on the fact that in our model $\Omega(n)$ links of a minimum spanning tree of V can be scheduled in a

⋆ Part of this work has been done at ETH Zurich. At MIT the author is supported by the following grants: AFOSR Contract Number FA9550-13-1-0042, NSF Award 0939370-CCF, NSF Award CCF-1217506, NSF Award number CCF-AF-0937274.

single time step. While the tree in [2] is stated to be directed towards the root, we can also obtain such a tree with *bidirectional links* using the bidirectional version of the *amenability* in [2]. Using these trees and a canonically derived link scheduling technique we call *level schedule* we show how to shrink $O(\log n)$ times the range in $[W]$ that contains the k^{th}-largest element by a constant factor, each time using $O(\log n)$ time steps to find a new range.

Theorem 1. *The k-selection problem can be solved in $\mathcal{O}(\log^2 n)$ time steps on an aggregation tree with a level schedule.*

Algorithm and proof of this theorem can be found in [3]. It has been shown in [2] that any distributive aggregation function can be computed in $\mathcal{O}(\log n)$ time steps in the SINR model with an MLAS. A matching lower bound [2] extends to k-selection, as finding the minimum is a special case of k-selection (i.e., k-selection with $k = 1$). Thus we could still hope for a quadratic speedup. However, this (if it is possible) requires new techniques since we show that by using a level schedule this can not be achieved.

Theorem 2. *The number of time steps required to solve the k-selection problem w.h.p. in an aggregation tree with a level schedule is in $\Omega(\log^2 n/\log\log n)$.*

The formal proof can be found in [3]. The theorem is proved with two reductions. First, solving the k-selection problem cannot be harder than solving the k-selection problem w.r.t. a subset of V. Second, it can be shown that in every MLAS aggregation tree as constructed in [2], there exist two disjoint subsets of V of size $\Omega(\sqrt{n})$ with the property that sending a message from one set to the other requires $\Omega(\log n/\log\log n)$ time steps. Any algorithm that solves the k-selection problem on an aggregation tree with a level schedule can therefore be used to build an algorithm that is $\Omega(\log n/\log\log n)$ times faster in the setting of the two-party k-selection problem (see [4]). This combined with a lower bound of $\Omega(\log n)$ for the two-party k-selection problem [4] proofs Theorem 2.

Finally we study sorting. We say that data in a network is sorted when each node $v \in V$ knows the id_v-th smallest input value in the network. An $O(n)$ algorithm and the full proof of Theorem 3 can be found in [3].

Theorem 3. *Assume $\alpha > 0$. Every (possibly randomized) algorithm in the SINR model for sorting has runtime $\Omega(n)$ in the worst case.*

Acknowledgements: We like to thank Magnus Halldórsson.

References

1. Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Trans. Inf. Theory 46(2), 388–404 (2000)
2. Halldórsson, M.M., Mitra, P.: Wireless connectivity and capacity. In: Proc. 23rd SODA 2012, pp. 516–526 (2012)

3. Kohler, S.: New algorithms for fundamental problems in wireless networks. Master's thesis, ETH Zürich, Department ITET, Zürich, Switzerland (2012)
4. Kuhn, F., Locher, T., Wattenhofer, R.: Tight bounds for distributed selection. In: Proc. 19th SPAA 2007, pp. 145–153 (2007)
5. Moscibroda, T., Wattenhofer, R., Weber, Y.: Protocol Design Beyond Graph-Based Models. In: Workshop on Hot Topics in Networks (2006)

Brief Announcement: Distributed 3/2-Approximation of the Diameter

Stephan Holzer[1,⋆], David Peleg[2,⋆⋆], Liam Roditty[3,⋆⋆⋆], and Roger Wattenhofer[4]

[1] Massachusetts Institute of Technology (MIT), Cambridge, USA
holzer@mit.edu
[2] Weizmann Institute, Rehovot, Israel
david.peleg@weizmann.ac.il
[3] Bar-Ilan University, Ramat-Gan, Israel
liamr@macs.biu.ac.il
[4] ETH Zurich, Zurich, Switzerland
wattenhofer@ethz.ch

Abstract. We present an algorithm to 3/2-approximate the diameter of a network in time $\mathcal{O}(\sqrt{n \log n} + D)$ in the CONGEST model. We achieve this by combining results of [2,6] with ideas from [7]. This solution is a factor $\sqrt{\log n}$ faster than the one achieved in [4] and uses a different approach. Our different approach is of interest as we show how to extend it to compute a $(3/2 + \varepsilon)$-approximation to the diameter in time $\mathcal{O}(\sqrt{(n/(D\varepsilon)) \log n} + D)$. This essentially matches the $\Omega(\sqrt{(n/D)\varepsilon} + D)$ lower bound for $(3/2 - \varepsilon)$-approximating the diameter [1].

1 Model and Basic Definitions

The CONGEST model [5] is a message passing model with limited bandwidth. We are interested in the number of communication rounds required by a distributed algorithm to solve a problem in the CONGEST model. Thus we neglect internal computations subsequently. We denote the number $|V|$ of nodes of a network by n. The (hop-)distance of nodes u and v in G is denoted by $d(u, v)$. A k-dominating set for a graph G is a subset $\mathcal{DOM}$ of vertices with the property that for every $v \in V$ there is a node $u \in \mathcal{DOM}$ at distance of at most k to v. The diameter $D := \max_{u,v \in V} d(u, v)$ of a graph G is the maximum distance between any two nodes of the graph.

⋆ Corresponding author. Part of this work has been done at ETH Zurich. Work at MIT supported by grants: AFOSR Contract Number FA9550-13-1-0042, NSF Award 0939370-CCF, NSF Award CCF-1217506, NSF Award number CCF-AF-0937274.

⋆⋆ Supported in part by grants from the Israel Science Foundation, the United-States - Israel Binational Science Foundation and the Israel Ministry of Science.

⋆⋆⋆ Work supported by the Israel Science Foundation (grant no. 822/10).

2 Results

Theorem 1. *Algorithm 1 computes a* $3/2$*-approximation of the diameter w.h.p. in* $\mathcal{O}(\sqrt{n \log n} + D)$ *time.*

We present Algorithm 1 which is inspired by [7] and can be implemented in a distributed way. Details of this implementation and a proof of Theorem 1 will appear in a journal version that merges [2] and [6]. Here, $C_k(w)$ denotes the set of k closest vertices to w visited by a (partial) breadth first search (BFS) starting in w that stops after visiting k nodes (ties are broken arbitrarily, e.g. by lexicographical order in the tree's topology).

Algorithm 1. Computes a 3/2-approximation to the diameter of G

```
1: each node $v$ joins set $S$ with probability $\sqrt{\log(n)/n}$;
2: compute a BFS from each node in $S$;
3: for every $v \in V$, compute $p_S(v) :=$ the closest node in $S$ to $v$;
4: $w := \arg\max_{v \in V} d(v, p_S(v))$;
5: compute a BFS tree from $w$ as well as $C_s(w)$;
6: for every $v \in C_s(w)$, compute a BFS tree from $v$;
7: return the maximum depth of any BFS tree that was computed;
```

Theorem 2. *For any* $0 < \varepsilon \leq 1/3$*, a* $(3/2 + \varepsilon)$*-approximation of the diameter can be computed w.h.p. in* $\mathcal{O}\left(\sqrt{n/(D\varepsilon)\log n} + D\right)$ *time.*

Details of the algorithm and a proof of Theorem 2 will appear in a journal version that merges [2] and [6]. We only sketch the main insight, which is to modify Algorithm 1 by using ideas of an algorithm to $(1 + \varepsilon)$-approximate the diameter presented in [2]. First we obtain a 2-approximation D' of D by computing the depth of a BFS from the node with smallest ID. Next we compute a $\Theta(\varepsilon D')$-dominating set $\mathcal{DOM}$ of size $\mathcal{O}(n/(\varepsilon D'))$ using [3]. Now we execute Algorithm 1 restricted to the nodes in $\mathcal{DOM}$, where 1) nodes join S with a probability $\sqrt{\log(n)/|\mathcal{DOM}|}$ instead of $\sqrt{\log(n)/n}$, and 2) nodes not in $\mathcal{DOM}$ implicitly participate in the algorithm (mainly by forwarding messages). Executing Algorithm 1 on this $\Theta(\varepsilon D')$-dominating set affects the approximation ratio only by $\Theta(\varepsilon)$. This reduction of the number of vertices to $\mathcal{O}(n/(\varepsilon D'))$ yields the speedup.

References

1. Frischknecht, S., Holzer, S., Wattenhofer, R.: Networks Cannot Compute Their Diameter in Sublinear Time. In: Proc. 23rd SODA 2012, pp. 1150–1162 (2012)
2. Holzer, S., Wattenhofer, R.: Optimal distributed all pairs shortest paths and applications. In: Proc. 31st PODC 2012, pp. 355–364 (2012)
3. Kutten, S., Peleg, D.: Fast distributed construction of small k-dominating sets and applications. Journal of Algorithms 28(1), 40–66 (1998)

4. Lenzen, C., Peleg, D.: Efficient distributed source detection with limited bandwidth. In: Proc. 32nd PODC 2013, pp. 375–382 (2013)
5. Peleg, D.: Distributed computing: A locality-sensitive approach. Society for Industrial and Applied Mathematics, Philadelphia (2000)
6. Peleg, D., Roditty, L., Tal, E.: Distributed algorithms for network diameter and girth. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 660–672. Springer, Heidelberg (2012)
7. Roditty, L., Williams, V.V.: Fast approximation algorithms for the diameter and radius of sparse graphs. In: Proc. 45th STOC 2013, pp. 515–524 (2013)

Brief Announcement: Space-Optimal Silent Self-stabilizing Spanning Tree Constructions Inspired by Proof-Labeling Schemes

Lélia Blin[1,⋆] and Pierre Fraigniaud[2,⋆⋆]

[1] LIP6-UPMC, University of Evry-Val d'Essonne, France
[2] CNRS and University Paris Diderot, France

Abstract. We present a general roadmap for the design of space-optimal polynomial-time silent self-stabilizing spanning tree constructions. Our roadmap is based on sequential greedy algorithms inspired from the design of proof-labeling schemes.

Context and Objective. One desirable property for a self-stabilizing algorithm is to be *silent*, that is, to keep the individual state of each process unchanged once a legal state has been reached. Silentness is a desirable property as it guarantees that self-stabilization does not burden the system with extra traffic between processes whenever the system is in a legal state. Designing silent algorithms is difficult because one must insure that the processes are able to collectively decide *locally* of the legality of a global state of the system, based solely on their own individual states, and on the individual states of their neighbors. This difficulty becomes prominent when one takes into account an important complexity measure for self-stabilizing algorithms: *space complexity*. Keeping the memory space limited at each process reduces the potential corruption of the memory, and enables to maintain several redundant copies of variables (e.g., for fault-tolerance) without hurting the efficiency of the system.

Our objective is to compute some spanning tree T of G. Typically, the tree T is rooted at some node r, and it is distributedly encoded at each node v by the identify of v's parent $p(v)$ in T. (The root r has $p(r) = \perp$). We are interested in all kinds of spanning trees, but will mostly focus our attention to two specific kinds of spanning trees: minimum-weight spanning trees (MST), and minimum-degree spanning trees (MDST). Constructing MSTs is a classical problem in the distributed computing setting. In the case of MDSTs, we aim at designing an algorithm which, for any given (connected) graph G, constructs a spanning tree T of G whose degree is minimum among all spanning trees of G. Our interest in MDSTs is motivated by resolving issues arising in the design of MAC protocols for sensor networks under the 802.15.4 specification. It is also worth pointing out that MDSTs arise in many other contexts, including electrical circuits, communication networks, as well as in many other areas. Since Hamiltonian-path is NP-hard, we actually slightly relax our task, by focussing on the construction

⋆ Additional supports from ANR project IRIS.
⋆⋆ Additional support from ANR project DISPLEXITY, and INRIA project GANG.

of spanning trees whose degree is within +1 from the minimum degree OPT of any spanning tree in the given graph.

It is known that, for every task with a proof-labeling scheme on k-bit labels, there is a silent self-stabilizing algorithm for that task using registers on $O(k + \log n)$ bits in n-node networks [2]. However, the convergence time of the generic algorithm in [2] may be exponential.

Our Results. We present a general roadmap for the design of *space-optimal polynomial-time* silent self-stabilizing constructions of spanning trees optimizing different kinds of criteria, under the state model[1]. Following our roadmap, we were able to design space-optimal algorithms for both MST and MDST constructions. Our MST algorithm uses registers of size $O(\log^2 n)$ bits in n-node networks, which is known to be optimal. While there exists more compact MST algorithms, these algorithms designed for minimizing the size of the memory are not silent. Our MDST algorithm is an additive approximation algorithm. It returns a spanning tree with degree at most OPT + 1. It uses registers of $O(\log n)$ bits, which is know to be optimal. It exponentially improves the previous best known (OPT + 1)-approximation algorithm, which is not silent, yet is using $\Omega(n \log n)$ bits of memory per node, and is converging in the same number of rounds. Both our algorithms converge in a number of rounds polynomial in n, and perform polynomial-time computation at each node. In fact, our MDST algorithm constructs a special kind of trees, named *FR-trees* after Fürer and Raghavachari. Indeed, we show that verifying whether a given tree is an arbitrary trees of degree $\leq$ OPT + 1 cannot be done in polynomial time, unless NP = co-NP. Instead, we show that there is a proof-labeling scheme for FR-trees using labels on $O(\log n)$ bits.

Our roadmap relies on a collection of ingredients. The first ingredient is the design of sequential *greedy* algorithms guided by *proof-labeling schemes.* The second ingredient is a *redundant* proof-labeling scheme for spanning trees, enabling the design of silent *loop-free* self-stabilizing algorithms for permuting tree edges with non-tree edges. The third ingredient is the design of a silent algorithm for the construction of the $O(\log n)$-bit label informative-labeling scheme for nearest common ancestor (NCA) from the literature, in order to identify the fundamental cycles. The two latter ingredients are used for implementing the sequential algorithms of the first ingredient in a distributed silent self-stabilizing manner.

More details are available in [1,2].

References

1. Blin, L., Fraigniaud, P.: Polynomial-Time Space-Optimal Silent Self-Stabilizing Minimum-Degree Spanning Tree Construction. Tech. Rep. arXiv 1402.2496 (2014)
2. Blin, L., Fraigniaud, P., Patt-Shamir, B.: On Proof-Labeling Schemes versus Silent Self-Stabilizing Algorithms. In: 16th SSS (2014)

[1] Recall that, in the state model, every node has read/write access to its own public variables, and read-only access to the public variables of its neighbors in the network G connecting the node.

Brief Announcement: Secure Anonymous Broadcast

Mahnush Movahedi, Jared Saia, and Mahdi Zamani

Dept. of Computer Science, University of New Mexico, Albuquerque, NM, USA 87131
{movahedi,saia,zamani}@cs.unm.edu

Consider a network of n parties, where there is a private and authenticated communication channel between every pair of parties. In anonymous broadcast, one or more of the parties want to anonymously send messages to all parties. This problem is used in many applications such as anonymous communication, private information retrieval, distributed auctions, and multi-party computation. To the best of our knowledge, known techniques for solving this problem either scale poorly with n or are vulnerable to jamming attacks [2,4], collisions [9], or traffic analysis [3].

We propose a decentralized algorithm for anonymous broadcast whose communication and computation scale linearly (up to a polylogarithmic factor) with the number of parties and is not vulnerable to jamming, collision, and traffic analysis. Our protocol is information-theoretically secure, does not require any trusted party, and is load-balanced. The protocol can tolerate up to a $(1/6 - \epsilon)$ fraction statically-scheduled *Byzantine* parties, for some positive constant ϵ. We assume the communication is *synchronous*, and we do *not* require reliable broadcast channels.

Similar to DC-Nets [2,4,9], we use *Multi-Party Computation (MPC)* for achieving anonymity with traffic analysis resistance. In MPC, a set of n parties, each having a secret input, compute a known function over their inputs without revealing the inputs to any party. In DC-Nets, all parties participate in a multi-party sum protocol with a zero input except one that participates with an input equal to a message it wants to broadcast. At the end, the sum result, which is equal to the nonzero input, is revealed to all parties anonymously.

Unlike DC-Nets, we let parties participate in a *multi-party shuffling* protocol, where the parties collaborate with each other to randomly shuffle their inputs. To achieve scalability, we perform local communications and computations in logarithmic-size groups of parties called *quorum*s, where we ensure that the fraction of dishonest parties in each quorum is guaranteed not to exceed a certain value. We create a set of quorums using the polylogarithmic Byzantine agreement protocol of Braud-Santoni et al. [1]. We prove the following theorem in [7].

Theorem. *For any $\epsilon > 0$, there exists an unconditionally-secure n-party protocol tolerating $t < (1/6 - \epsilon)n$ dishonest parties such that, with high probability, each honest party sends its message to all parties anonymously. The protocol has $O(\log n)$ rounds of communication and requires each party to send $\tilde{O}(1)$ bits and compute $\tilde{O}(1)$ operations for delivering one anonymous bit.*

Protocol Overview. To perform multi-party shuffling, we assign to each message a uniform random value, and then obliviously sort the messages according to the random values. We implement this in a decentralized fashion by evaluating the sorting circuit of [6] over secret-shared[1] inputs. A sorting circuit consists of *comparator* gates each with two inputs and two outputs, where the outputs are determined by comparing the inputs. Our protocol first builds a set of quorums in a one-time setup phase and then, assigns each comparator to a quorum that is responsible for computing the functionality of the comparator over secret-shared values using the secure comparison protocol of [8]. Then, the circuit is evaluated level-by-level, passing the outputs of one level as the inputs to the next level. Once the local computation is finished in each quorum, the result is forwarded to the next quorum via one-to-one communication with parties of the next quorum. Finally, at the highest level, the shuffled messages are reconstructed and sent back to all parties via a binary tree structure.

When forwarding a secret-shared value from one quorum to another, we need to ensure that no coalition of dishonest parties from the quorums involved can learn anything about the secret value. To this end, we first generate a random polynomial that passes through the origin, and then add it to the polynomial that corresponds to the shared secret. The result is a new polynomial that represents the same secret but with *fresh* random coefficients (see [7] for a formal definition).

One issue with the *shuffling-via-sorting* technique described above is that if the random values are not distinct, then the resulting distribution can deviate from the uniform distribution. In [7], we show that by choosing the random values from a sufficiently large domain, we can prevent such collisions with high probability. Namely, we prove that a domain of size $\Omega(kn^2 \log n)$ elements guarantees a uniform random shuffle with probability $1 - 1/n^k$, for any constant $k > 0$.

References

1. Santoni, N., Guerraoui, R., Huc, F.: Fast Byzantine agreement. In: PODC 2013, pp. 57–64 (2013)
2. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
3. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: USENIX Security Symposium 2004, p. 21 (2004)
4. Golle, P., Juels, A.: Dining cryptographers revisited. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 456–473. Springer, Heidelberg (2004)
5. Katz, J., Koo, C.-Y., Kumaresan, R.: Improving the round complexity of VSS in point-to-point networks. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 499–510. Springer, Heidelberg (2008)
6. Leighton, T., Plaxton, C.G.: A (fairly) simple circuit that (usually) sorts. In: FOCS 1990, pp. 264–274 (1990)
7. Movahedi, M., Saia, J., Zamani, M.: Secure Anonymous Broadcast. ArXiv e-prints, 1405.5326 (May 2014)
8. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007)
9. Zamani, M., Saia, J., Movahedi, M., Khoury, J.: Towards provably-secure scalable anonymous broadcast. In: USENIX Free and Open Comm. on Internet, FOCI (2013)

[1] We use the verifiable secret sharing scheme of [5] that builds upon Shamir's scheme.

Brief Announcement: Privacy-Preserving Location-Based Services

Mahnush Movahedi and Mahdi Zamani

Dept. of Computer Science, University of New Mexico, Albuquerque, NM, USA 87131
{movahedi,zamani}@cs.unm.edu

Nowadays, mobile users frequently ask *Location-Based Services (LBS)* to find points of interest near them, to receive information about traffic along their route, or to receive customized advertising. Unfortunately, shared location data can be used by others (e.g., providers and governments) for precise surveillance and hence, compromising user privacy. As far as we know, current privacy-preserving LBS protocols have at least one of the following drawbacks: (1) assumption of trusted third parties [2,6], (2) vulnerability to global attacks such as traffic analysis [5,6], (3) inaccurate query results due to spatial cloaking [2,5,6], and (4) insecurity against malicious behaviors [2,5,6].

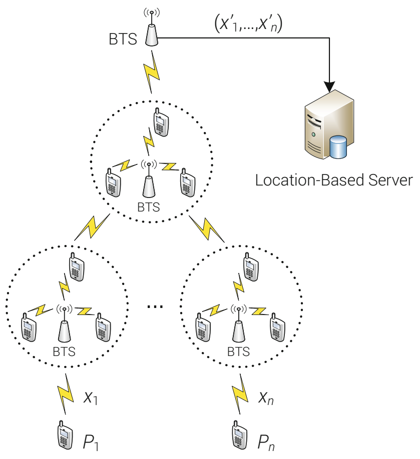
In this paper, we propose an efficient protocol for privacy-preserving LBS that is secure against malicious attacks as well as global attacks. Our protocol scales well with the number of clients and is load-balanced. Load-balancing is crucial since mobile devices usually have very limited resources. Moreover, unlike the majority of previous work which rely on centralized trusted servers, our construction is fully-decentralized. Our protocol provides polylogarithmic per-client communication and computation costs with respect to the number of clients and achieves the highest location accuracy by avoiding location cloaking.

Theorem. [1] *Consider n clients in a fully-connected synchronous network with private channels, where each client has a locational query to send to a server. There exists an n-party cryptographic protocol tolerating up to $t < (1/6 - \epsilon)n$ malicious clients such that, with high probability, each honest client sends its query to the server anonymously. The protocol requires each client to send $\tilde{O}(1)$ bits and compute $\tilde{O}(1)$ operations in $O(\log n)$ rounds of communication.*

Protocol Overview. Consider n parties $P_1, P_2, ..., P_n$ each having a locational query x_i, for all $i \in [n]$. The parties want to anonymously send their queries to a location-based server and receive the query results back. Our high-level idea is to perform a *multi-party shuffling* among all clients ensuring that their inputs remain private, and no adversary can trace the messages to their corresponding senders. To this end, we adopt and implement the distributed shuffling technique of [7] with cryptographic assumptions for achieving location privacy. This protocol first builds a set of *quorums*[1] in a one-time setup phase, and then uses the quorums in an online phase for shuffling client queries: a uniform random value is assigned to each message, and then the messages are sorted obliviously according to the random values using a sorting circuit [7]. Figure 1 (left) depicts

[1] A quorum is a group of $O(\log n)$ parties, where the fraction of malicious parties in each quorum is guaranteed not to exceed a certain value.

F. Kuhn (Ed.): DISC 2014, LNCS 8784, pp. 569–571, 2014.



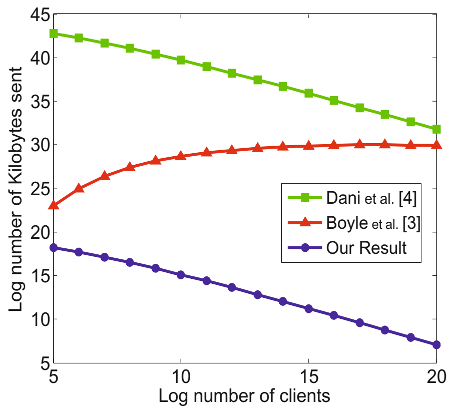


Fig. 1. Our architecture (left) and our simulation results (right)

our protocol architecture based on the algorithm of [7]. Each circle depicts a quorum of mobile users who connect to their local base station. Once the local computation is finished in each quorum, the result is forwarded to the next quorum via one-to-one communication with clients of the next quorum. Finally, at the highest level, the shuffled queries are reconstructed and sent to the LBS. Once the queries are processed, the server broadcasts the results to the parties (see [1] for a precise protocol description).

To study the feasibility of our scheme and compare it to previous work, we implemented a simulation of our protocol and two other protocols [3,4] that can be used for shuffling n queries randomly in a similar setting. As far as we know, these protocols have the best scalability with respect to the network size among other works. Figure 1 (right) shows the simulation results obtained for various network sizes between 2^5 and 2^{20} (between 32 and about 1 million). We observe that our protocol performs significantly better than others (see [1] for a complete simulation setup and discussion on the results).

References

1. Full version of this paper, `http://cs.unm.edu/~zamani/papers/lbs-full.pdf`
2. Bamba, B., Liu, L., Pesti, P., Wang, T.: Supporting anonymous location queries in mobile environments with PrivacyGrid. In: WWW 2008, pp. 237–246 (2008)
3. Boyle, E., Goldwasser, S., Tessaro, S.: Communication locality in secure multi-party computation: How to run sublinear algorithms in a distributed setting. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 356–376. Springer, Heidelberg (2013)
4. Dani, V., King, V., Movahedi, M., Saia, J.: Quorums quicken queries: Efficient asynchronous secure multiparty computation. In: Chatterjee, M., Cao, J.-N., Kothapalli, K., Rajsbaum, S. (eds.) ICDCN 2014. LNCS, vol. 8314, pp. 242–256. Springer, Heidelberg (2014)

5. Ghinita, G., Kalnis, P., Skiadopoulos, S.: Prive: Anonymous location-based queries in distributed mobile systems. In: WWW 2007, pp. 371–380 (2007)
6. Mokbel, M.F., Chow, C.-Y., Aref, W.G.: The new casper: Query processing for location services without compromising privacy. In: VLDB 2006, pp. 763–774 (2006)
7. Movahedi, M., Saia, J., Zamani, M.: Secure Anonymous Broadcast. ArXiv e-prints, 1405.5326 (May 2014)

Author Index