Cloud-TM: Harnessing the Cloud with Distributed Transactional Memories

Paolo Romano
INESC-ID
romanop@gsd.inesc-id.pt

Luis Rodrigues
INESC-ID/IST
ler@ist.utl.pt

Nuno Carvalho
INESC-ID/IST
nonius@gsd.inesc-id.pt

João Cachopo
INESC-ID/IST
Joao.Cachopo@ist.utl.pt

ABSTRACT

One of the main challenges to harness the potential of Cloud computing is the design of programming models that simplify the development of large-scale parallel applications and that allow ordinary programmers to take full advantage of the computing power and the storage provided by the Cloud, both of which made available, on demand, in a pay-only-for-what-you-use pricing model.

In this paper, we discuss the use of the Transactional Memory programming model in the context of the cloud computing paradigm, which we refer to as Cloud-TM. We identify where existing Distributed Transactional Memory platforms still fail to meet the requirements of the cloud and of its users, and we point several open research problems whose solution we deem as essential to materialize the Cloud-TM vision.

1. INTRODUCTION

The cloud computing paradigm has been receiving increasing attention in the recent years. The cloud computing vision encompasses a general shift of computer processing, storage, and software delivery away from the desktop and local servers, across the network, and into next generation data centers hosted by large infrastructure companies such as Amazon, Google, Yahoo, Microsoft, or Sun. Just as the electric grid revolutionized access to electricity one hundred years ago, freeing corporations from having to generate their own power and enabling them to concentrate on their business differentiators, cloud computing is hailed as revolutionizing IT, freeing corporations from large IT capital investments and enabling them to plug into extremely powerful computing resources over the network [1].

In practice, cloud computing platforms, such as those offered by Amazon Web Services, AT&T's Synaptic Hosting, the HP/Yahoo/Intel Cloud Computing Testbed, and the IBM/Google cloud initiative, work differently than conventional Application Service Providers (ASP). Instead of owning, installing, and maintaining the software for their customers, cloud computing vendors typically maintain a hardware and middleware infrastructure, and provide to customers a virtual environment in which they can install their own software.

One of the main challenges that we, as a community, have to face to bring about the potential of cloud computing is the development of programming models and tools that simplify the design and implementation of applications for the cloud; without them, programmers targeting the cloud will not be able to take full advantage of the vast amount of computing power and storage available on demand, in a pay-only-for-what-you-use pricing model. This is a relevant research area and several novel programming models for simplifying large scale computations across shared-nothing clusters (e.g., MapReduce [12], Sinfonia [3], Scope [9], and Pig [24]) have been recently proposed by different research communities. Each of these models was designed with different goals in mind, and has its own weaknesses and strengths.

Another relevant research area that is currently garnering a wide interest from several communities proposes the adoption of *Transactional Memories* (TMs) to simplify the development of shared-memory concurrent programs [17, 30, 21]. Motivated by the recent trend that made multi-core and many-core CPUs the architecture-of-choice for mainstream computing, TMs represent an attractive solution to free programmers from the pitfalls of conventional explicit lock-based thread synchronization. Instead, TMs rely on the proven notion of atomic transactions to simplify concurrent programming [2]. By relishing the programmer from the burden of managing locks or other error-prone low-level concurrency control mechanisms, TMs have been shown to enable a significant boost in productivity, shortening development times, and increasing code reliability in complex concurrent applications [8].

Clearly, there are similarities and complementarities between the goals of the TM model and the MapReduce-like large scale parallel computing paradigms. The latter ultimately aims at bringing the power of parallel computing into the hands of ordinary users, whereas TMs permit to simplify the development of concurrent (though not distributed) applications by hiding the complexity of lock-based synchronization.

In this position paper we advocate a convergence between the TM programming model and the cloud computing paradigm, which we refer to as Cloud-TM. Like some recent Distributed Transactional Memory (DTM) approaches, e.g. [20, 7, 28, 11], the Cloud-TM paradigm enriches the traditional TM model to breach the boundaries of a single machine and to use transparently the resources available in a distributed environment. On the other hand, as we shall discuss, none of the existing DTM solutions is capable of taking full benefit from the existing and future cloud computing platforms' infrastructures (e.g., very large clusters built on top of cheap, failure-prone commodity hardware). Furthermore, even if existing DTM solutions free programmers from the burden of dealing with distributed lock-based synchronization, they still require that programmers design and code explicitly parallel applications, thus failing to meet one of the primary requirements of cloud programmers: *simplicity.* In fact, rather than expert architects of robust and scalable distributed applications, typical programmers of cloud computing applications are used to develop sequential code. So, approaches such as MapReduce, which automate the parallelization and fault-management in a distributed program but that require the developer to master an unfamiliar programming model and to redesign pre-existing programs to benefit from cloud features, are not the final solution.

In the remainder of this paper, we first critically analyze the current mainstream programming models for large scale parallel systems, highlighting their main strengths and weaknesses. Next, we outline several open research challenges that need to be addressed to materialize our vision of Cloud-TM. Finally, we overview some of our recent results towards the achievement of this ambitious goal.

2. PROGRAMMING MODELS FOR THE CLOUD

In this section we overview some of the most popular models for programming parallel and concurrent systems, critically highlighting their main pros and cons when adopted in the context of a cloud computing infrastructure.

MapReduce-like approaches: MapReduce [12] is a functional programming model that permits automatic parallelization and execution on large scale clusters. By forcing developers to adhere to a restricted, although neatly defined, programming model, the MapReduce's runtime system is able to take care automatically of issues such as input data partitioning, scheduling the program's execution across multiple machines, and handling of nodes' failures. Inspired in the *map* and *reduce* primitives present in Lisp, MapReduce computations are structured into two sequential phases. A first mapping phase, which operates over each "record" in the input and outputs a set of intermediate key/value pairs. Then, a reducing phase, where all the values that share the same key are processed and combined based on some application level logic.

MapReduce is being extensively used in large scale Google data centers to analyze in parallel huge data sets in domains such as web log and graph analysis. Its automatic parallelization and fault-tolerance features have attracted the attention of a growing community of enthusiastic users that have developed a complete open source porting of the original proprietary system—the Apache Hadoop project. Nevertheless, it is nowadays widely recognized that, depending on the nature of the problem to be addressed, casting a known solution algorithm into the functional MapReduce programming model might be far from being trivial, possibly forcing to fragment the computation into a sequence of MapReduce tasks or inducing unnatural additional steps that can lead to significant performance drawbacks with respect to a conventional explicit parallel scheme [26]. So, more recently, a number of MapReduce extensions, e.g. Cascading, Pig [24] or DryadLINQ [34], have been proposed in the form of additional APIs that expose simpler SQL-like programming interfaces (hence not forcing developers to "think" in MapReduce), which are then automatically mapped to an underlying MapReduce implementation. On the other hand, the actual efficiency of MapReduce is currently a matter of a controversial debate [1, 13], with several well-known scientists critically highlighting several performance issues of MapReduce and its derivatives, based both on a comparison with the mechanisms supported by modern parallel databases, as well as on benchmarking results showing MapReduce to be about an order of magnitude slower than alternative systems[33].

PGAS approaches: Another alternative programming paradigm that has recently emerged as an attractive way to program large parallel systems is the so-called Partitioned Global Address Space (PGAS), e.g. [6], embodied in languages such as Unified Parallel C (UPC), Co-Array Fortran, and Titanium.

PGAS is considered as a sweet spot between a pure shared-memory style emulated in software, such as software distributed shared memory (DSM) [4], and an explicit message-passing style, such as MPI. Like shared memory programming, and unlike MPI, PGAS languages provide a global address space, allowing threads to refer to remote memory directly, instead of via message-passing calls.

By letting the programmer explicitly control data locality and communication, the PGAS model permits to achieve better performance and scalability than classical software DSM. Yet, this comes at the cost of a lower-level, more complex, programming interface. This makes PGAS approaches mainly targeted towards high-performance computing applications, rather than the reference paradigm for general purpose cloud computing platforms.

DTM approaches: Whereas TMs have garnered a huge research interest over the last 6 years, most of the research efforts in this area have been in the context of non-distributed, cache coherent, shared-memory systems, e.g. multi-cores and SMPs architectures. By contrast, the problem of how to extend the TM abstraction across the boundaries of a single machine and to employ transactions as a first class abstraction for large scale distributed programming has only very recently started to be addressed; we are aware of the existence of a handful of DTM platforms only [20, 7, 23, 11, 3].

One may argue that this may be, in some sense, due to the skepticism of the research community towards distributed shared memory (DSM) approaches. In fact, two decades of research on DSMs have clearly highlighted that DSMs are capable of achieving good performances only if programmers embrace relaxed consistency models [19]. Unfortunately, relaxed consistency models are typically challenging for ordinary programmers, because they are forced to understand all the subtleties of complicated consistency properties to avoid endangering correctness. As highlighted by the experimental evaluation of several of the aforementioned DTM platforms [3, 11, 7], DTMs use the transaction's abstraction not only as way to simplify parallel programming but also as a natural means to aggregate communication efficiently, avoiding the performance pitfalls proper of DSM systems without sacrificing programming simplicity. In fact, unlike strongly consistent DSMs, which require expensive remote synchronizations at each single memory access, atomic transactions allow for optimistic implementations that permit to batch any consistency action within a single synchronization phase taking place at commit time [3, 20, 11]. This approach amortizes communication overheads across a (possibly large) number of memory accesses, with clear benefits in terms of performance.

Taking a closer look at existing DTM platforms, it is relatively easy to draw a line between solutions, such as those in [20, 23, 11], which were designed for being deployed in small scale clusters, and those, such as Cluster-STM [7] and Sinfonia [3], which were architected to scale up to several hundreds of nodes.

In the former group of solutions, all are based on full replication schemes and, when fault-tolerance is addressed (which is actually the case only for [11]), this is done by using Group Communication Systems that provide support for Virtual Synchrony and Atomic Broadcast. These strategies have shown encouraging performance results when employed in clusters of at most 10 nodes, but it is very unlikely that they sustain the scalability challenges of the largest cloud computing environments.

Cluster-STM [7] and Sinfonia [3], on the other hand, were shown to achieve impressive scalability levels when deployed in typical data centers' settings. Even though these two solutions are affected by a number of rather constraining limitations (e.g., in Sinfonia (mini-)transactions need to predeclare statically the sequence of memory accesses to be performed, and Cluster-STM supports at most one thread per machine and provides no fault-tolerant guarantees), they are characterized by a common trait. They both expose an explicitly partitioned address space to the programmers and delegate to the application level the role of implementing any form of caching.

This has the main advantage of enabling expert developers to use deep knowledge of the application domain to define highly optimized memory layouts and introduce carefully-thought optimizations. Unsurprisingly, the (excellent) performances of both these platforms were demonstrated considering applications that were carefully designed and optimized to maximize access locality and minimize remote memory accesses. For instance, the cluster file system built on top of Sinfonia places, whenever possible, an inode, its chaining list, and its file data in the same memory node. Likewise, the implementation of the SSCA2 Kernel 4 graph analysis algorithm in Cluster-STM retrieves the adjacency lists of multiple vertexes stored by remote nodes via a single message, rather than by sending out one message per vertex.

On the other hand, it is natural to wonder how likely it would be for an ordinary developer, not so experienced with distributed programming, to end up coding applications that deliver poor performance in such a "minimalistic" system, or whether it would be, instead, desirable to demand some additional assistance from the runtime environment.

3. TOWARDS A CLOUD-TM

At the light of our previous analysis, in this section we characterize a novel programming model, named Cloud-TM, which is explicitly tailored to match the requirements of cloud computing applications. Cloud-TM aims at extending the imperative programming model of DTM systems with a set of additional mechanisms ultimately aimed at simplifying the life of cloud programmers, by masking most of the complexity associated with the design and implementation of large scale parallel application.

Rather than presenting a set of complete solutions, in the following we highlight a number of gaps in existing DTM platforms and outline several research challenges whose addressing we deem as essential to materialize the Cloud-TM model.

C.1) Hide Complexity: As already discussed, one the main drawbacks of MapReduce-inspired systems is that their functional programming paradigm may force the programmers to structure their applications in an unnatural and/or inefficient way. Conversely, imperative programming paradigms, such as those supported by the DTM model, typically lend themselves to tackle problems that do not fit nicely in the MapReduce model [26].

On the down-side, however, the (D)TM model requires that programmers explicitly design and develop parallel algorithms, because it has none of the auto-parallelization features that are one of the strengths of MapReduce-inspired systems.

Motivated by MapReduce's success in achieving automatic code parallelization, we argue that a Cloud-TM platform should be able to achieve automatic parallelization of sequential TM programs. This is clearly a very ambitious goal, given that automatic parallelization of *generic* sequential programs by compilers remains a grand challenge after decades of research. On the other hand, (D)TM environments, thanks to the built-in availability of isolated, atomic transactions, are particularly well suited to support thread-level speculation (TLS) techniques [16, 31, 22]. In a (D)TM environment, in fact, TLS can be simply and efficiently implemented by transactifying the portions of (sequential) code to be executed speculatively, and by exploiting the transactions' atomicity and isolation semantics to detect code dependencies and rollback to consistent states invalid (and possibly unbounded) computations.

Transparent fault-masking is another highly-valued feature of MapReduce that, unfortunately, is not satisfactorily supported by most of the existing DTM platforms. The solutions in [20, 23, 7], for instance, do not provide any fault-tolerance mechanism at all. This is clearly a major lack in a platform for large scale distributed computing where node failures represent the norm rather than the exception.

C.2) Coping with Workload Heterogeneity: Despite the vast body of literature developed in the area of (cache-coherent) TMs, the search for a TM solution capable to perform optimally in presence of any arbitrary workload has turned out to be inconclusive, motivating recent efforts aimed at developing polymorphic schemes that are able to self-tune to match the actual workload characteristics [15, 27]. This problem is even more exacerbated in a DTM environment, where additional complex mechanisms need to be used to ensure global consistency. In fact, it is well known (see [10]) that the performance of transactional data replication schemes is strongly affected by the workload characteristics (e.g. probability of conflict among remote/local transactions, access locality, average size of transactions' read set and write set, etc.). Manually identifying the optimal configuration in the huge design space of a DTM is at least impractical and time-consuming, or even impossible in the case of dynamic workloads.

On the other hand, the problem of adjusting automatically the policies used for, e.g. enforcing replica consistency, regulating local and remote transactional conflicts, or governing the local caching strategies, remains an open research question that demands gaining further insights on the modelling of the performance impacts of the workload characteristics on these mechanisms, as well as on their mutual correlations. We believe that a first and fundamental step in this direction is the development of novel workload monitoring and characterization methodologies aimed at automatically constructing probabilistic models of the transactions' data access patterns. Finally, an open challenge to enable the adoption of these self-tuning mechanisms is to conceive dynamic self-reconfiguration schemes that permit to switch efficiently between heterogeneous distributed management strategies, while preserving correctness during transitions or even despite the simultaneous coexistence of different schemes.

C.3) Maximize Locality: Unlike a non-distributed cache-coherent TM system, the cost of executing a transaction in a cloud computing environment depends on the location of the datasets to be accessed: Whether they reside on the same local node, on a remote node attached to the same network switch, on a remote node within a different network branch of the same cluster, or on a remote node belonging to some geographically distributed cluster (large cloud computing providers, such as Amazon S3 cloud storage service, own data centers spread throughout the world).

Hence, in cloud computing environments, the effectiveness of any parallel programming platform, and in particular of a DTM system, is strictly dependent on the ability to keep these cost factors into account and to tune its internal consistency mechanisms in such a way that they minimize the cost of interprocess communications.

At the light of these considerations, we argue that a Cloud-TM platform should not only offer self-tuning mechanisms for dynamically rearranging the data and code placement policies, but also first-class language constructs to guide these self-optimizing schemes and, whether required, to permit applications to control exactly the actual physical memory layout and caching policies to be adopted.

C.4) Automatic Resource Provisioning: Given that the processing power and storage available on demand in a cloud are provided according to a pay-only-for-what-you-use pricing model, the main dilemma that a customer of a cloud computing infrastructure is constantly faced with is: "What costs should I sustain to meet my performance and, more in general, Quality of Service criteria?". Providing effective tools and methodologies for guiding, and ideally automatizing, the provisioning of the computing resources to be hired represents a key enabling factor to materialize the cloud computing promise of elasticity in the face of changing conditions, and, ultimately, to consolidate its business model.

Even though a number of approaches have been recently proposed to compute dynamically the resource's demand of applications deployed in virtualized data centers, see e.g. [18, 32], existing solutions are not designed to keep into account relevant aspects of DTM platforms, such as the relation between the number of involved replicas, the data caching strategy, or the probability of logical contention. To take into account these aspects, we need novel ad hoc cost/performance forecasting models.

C.5) ACI vs ACID: Unlike classic ACID transactions of the database world, existing TM solutions normally do not provide the durability property for their transactions, because they are meant to synchronize accesses to the shared-memory of a single machine only while the machine is running. So, most TMs do not need to persist their data either to disk or to external systems. In a cloud computing environment, however, an application may be running over a cluster of several machines, and, so, its data must be sent among those machines.

Furthermore, the unitary computing resources that are made available on demand in a cloud computing environment normally include an attached persistent storage. In the Amazon's Elastic Compute Cloud (EC2), for instance, computing resources are normally apportioned in small, large, and extra large virtual private server instances, where even the smaller server instance is equipped with 160GB of storage. Hence, not taking advantage of the local disk storage for recovery, performance (e.g. to store larger local caches and reduce network bandwidth consumption), and scalability purposes (i.e. to implement a classical, though distributed, hierarchical memory system) would represent not only a missed opportunity, but also a wasted cost.

Thus, we argue that a Cloud-TM platform must deal with the issues of data serialization/deserialization, typically at the transactional boundaries. Even though this is a well-studied topic in the database world, it is largely unaddressed in the TM world.

4. SOME PRELIMINARY RESULTS

Even though there is a long road ahead before the Cloud-TM model is fully materialized, at INESC-ID/IST we have a number of research lines addressing some of the research challenges outlined in the previous section.

On the front of the design and development of DTM platforms, our recent work in developing D^2STM [11] has allowed us to gain valuable insights on DTM systems providing not only strong consistency guarantees, but also transparently ensuring failure resilience (see challenge C.1). Targeting relatively small scale clusters, and supporting exclusively full data replication, the D^2STM (Dependable Distributed STM) system is designed to operate in environments that are certainly less challenging than large scale cloud computing infrastructures. Nevertheless, our experimentation with the D^2STM system has allowed us to highlight both encouraging signals as well as potential performance pitfalls. On the positive side, we could observe that the replica coordination overhead can be significantly amortized, at the middleware layer, through the usage of optimistic certification-based approaches and of efficient message encoding techniques. Our experimental results show in fact that, in clusters of a ten of nodes, D^2STM can achieve linear speed-ups and significantly outperform a non-replicated TM. On the other hand, we could also verify how hard it is in practice to achieve satisfactory performance unless two conditions are met: (1) that the application-level code is adequately designed to reduce the sources of contention and to batch significant portions of computation within the same transaction, and (2) that the runtime environment is able to adapt in a timely manner its data replication strategies to cope with heterogeneous and dynamically varying workloads.

Given that the manual identification of an optimal tuning of a distributed application is a heavy burden for the programmers, some of our latest research efforts has been devoted at building tools aimed exactly at simplifying such a task. On one hand, we are developing stochastic techniques for identifying and predicting the data access patterns of transactional applications [14]. We regard these methodologies not only as valuable instruments to support the developers in the applications' optimization (see challenge C.3), but also as an essential building block of any autonomic runtime system that adaptively tunes its internal mechanisms to better match the workload characteristics (see challenge C.2). On the other hand, we are investigating the possibility of exploiting the transactional semantics provided by a (Distributed) Software Transactional Memory to support thread-level speculation techniques that achieve automatic, performance-effective parallelization of sequential programs [5] (see challenge C.1). Our prototype, named JaSPEx [5], albeit being currently capable of automatically parallelizing Java code meant to be executed on a single multi-core machine, is showing some very promising preliminary results. Thus, its integration within a DTM platform appears as a natural direction of our future work.

Finally, in [29] we have recently introduced a novel replication technique, which we call Speculative Transactional Replication (STR), that aims at maximizing the performance of applications characterized by "problematic" data access patterns, i.e. exhibiting a high contention level and very fine grained transactions, and for which there is no (or very little) room for application-level code optimizations (see challenge C.2). The idea underlying STR is rather simple—to seek maximum overlap between the replica coordination and transaction processing phases. This is accomplished with two complementary approaches: (1) by triggering the latter well before the former is concluded (i.e. as soon as transactions are optimistically delivered [25]), and (2) by speculatively exploring multiple transaction serialization orders instead of waiting for the final outcome of the coordination phase. This permits to avoid underutilizing the computing resources locally available at each replica, and to increase the maximum parallelism level globally achievable by the system. The work in [29] allowed us to lay the ground for the design and implementation of STR systems by introducing a set of desirable correctness and optimality properties for a speculatively replicated transactional system, as well as by presenting the first optimal STR algorithm. Integrating the STR within a working DTM prototype represents an important part of our future work.

5. CONCLUDING REMARKS

It is our belief that the success of the cloud computing paradigm will strongly depend on whether it will be possible to identify adequate parallel programming models that are able to achieve two distinct, and often contrasting, goals: (1) simplifying the development of large scale parallel applications, so as to bring the power of parallel computing into the hands of ordinary programmers, and (2) simultaneously sustaining the harsh scalability challenges characterizing cloud platforms.

In this paper, we have advocated the potential of Cloud-TM, a programming model based on a DTM platform specifically architected for large scale, elastic cloud computing environments. Based on our experience in designing and implementing DTM systems, and at the light of a critical analysis of several mainstream programming models for large scale parallel computing, we have identified several critical gaps in existing DTM systems, which make them not ripe for successful cloud deployment.

Finally, we have identified several open research directions that we deem as essential for materializing the Cloud-TM model, and overviewed some of our most recent results towards the fulfilment of this goal.

Acknowledgments

This work was partially supported by the FCT project Pastramy (PTDC/EIA/72405/2006).

6. REFERENCES

[1] Abadi, D. J. Data management in the cloud: Limitations and opportunities. IEEE Data Eng. Bulletin, 32(1), March 2009.

[2] Adl-Tabatabai, A.-R., Kozyrakis, C., and Saha, B. Unlocking concurrency. *ACM Queue 4*, 10 (2007), 24–33.

[3] Aguilera, M. K., Merchant, A., Shah, M., Veitch, A., and Karamanolis, C. Sinfonia: a new paradigm for building scalable distributed systems. In *Proc. Symposium on Operating Systems Principles (SOSP)* (New York, NY, USA, 2007), ACM, pp. 159–174.

[4] Amza, C., Cox, A. L., Dwarkadas, S., Keleher, P. J., Lu, H., Rajamony, R., Yu, W., and Zwaenepoel, W. Threadmarks: Shared memory computing on networks of workstations. *IEEE Computer 29*, 2 (1996), 18–28.

[5] Anjo, I., and Cachopo, J. Jaspex: Speculative parallel execution of java applications. In *Proc. of the Simpósio de Informática (INFORUM)* (Lisbon, Portugal, Sept. 2009).

[6] Barton, C., Casçaval, C., Almási, G., Zheng, Y., Farreras, M., Chatterje, S., and Amaral, J. N. Shared memory programming for large scale machines. In *Proc. of Programming Language Design and Implementation (PLDI)* (New York, NY, USA, 2006), ACM, pp. 108–117.

[7] Bocchino, R. L., Adve, V. S., and Chamberlain, B. L. Software transactional memory for large scale clusters. In *Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP)* (New York, NY, USA, 2008), ACM, pp. 247–258.

[8] Cachopo, J. *Development of Rich Domain Models with Atomic Actions.* PhD thesis, Tech. Univ. of Lisbon, 2007.

[9] Chaiken, R., Jenkins, B., Larson, P.-A., Ramsey, B., Shakib, D., Weaver, S., and Zhou, J. Scope: easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow. 1*, 2 (2008), 1265–1276.

[10] Ciciani, B., Dias, D. M., and Yu, P. S. Analysis of concurrency-coherency control protocols for

distributed transaction processing systems with regional locality. *IEEE Trans. Softw. Eng. 18*, 10 (1992), 899–914.

[11] Couceiro, M., Romano, P., Carvalho, N., and Rodrigues, L. D^{2}STM: Dependable Distributed Software Transactional Memory. In *Proc. 15th Pacific Rim International Symposium on Dependable Computing (PRDC)* (2009), IEEE Computer Society Press.

[12] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Comm. ACM 51*, 1 (2008), 107–113.

[13] DeWitt, D., and Stonebraker, M. Mapreduce: A major step backwards, `http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html`.

[14] Garbatov, S., Cachopo, J., and Pereira, J. Data access pattern analysis based on bayesian updating. In *Proc. of the Simpósio de Informática (INFORUM)* (Lisbon, Portugal, Sept. 2009).

[15] Guerraoui, R., Herlihy, M., and Pochon, B. Polymorphic Contention Management. In *Proc. of the International Symposium on Distributed Computing (DISC)* (2005), pp. 303–323.

[16] Hammond, L., Willey, M., and Olukotun, K. Data speculation support for a chip multiprocessor. *SIGOPS Operating Systems Review 32*, 5 (1998), 58–69.

[17] Herlihy, M., Eliot, J., and Moss, B. Transactional memory: Architectural support for lock-free data structures. In *Proc. of the International Symposium on Computer Architecture (ISCA)* (1993), pp. 289–300.

[18] Kalyvianaki, E., Charalambous, T., and Hand, S. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proc. of the 6th International Conference on Autonomic Computing (ICAC)* (New York, NY, USA, 2009), ACM, pp. 117–126.

[19] Keleher, P., Cox, A. L., and Zwaenepoel, W. Lazy release consistency for software distributed shared memory. In *Proc. of the Int. Symposium on Computer architecture (ISCA)* (New York, NY, USA, 1992), ACM, pp. 13–21.

[20] Kotselidis, C., Ansari, M., Jarvis, K., Lujan, M., Kirkham, C., and Watson, I. Distm: A software transactional memory framework for clusters. In *Proc. 37th International Conference on Parallel Processing (ICPP)* (Sept. 2008), pp. 51–58.

[21] Kumar, S., Chu, M., Hughes, C. J., Kundu, P., and Nguyen, A. Hybrid transactional memory. In *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming* (New York, NY, USA, 2006), ACM Press, pp. 209–220.

[22] Liu, W., Tuck, J., Ceze, L., Ahn, W., Strauss, K., Renau, J., and Torrellas, J. POSH: a TLS compiler that exploits program structure. In *Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP)* (New York, NY, USA, 2006), ACM, pp. 158–167.

[23] Manassiev, K., Mihailescu, M., and Amza, C. Exploiting distributed version concurrency in a transactional memory cluster. In *Proceedings of the Symposium on Principles and practice of parallel programming* (New York, NY, USA, 2006), ACM, pp. 198–208.

[24] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. Pig latin: a not-so-foreign language for data processing. In *Proc. of the Int. Conference on the Management of Data (SIGMOD)* (New York, NY, USA, 2008), ACM, pp. 1099–1110.

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

[26] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proc. of the International Symposium on High-Performance Computer Architecture (HPCA)* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 13–24.

[27] Riegel, T., Fetzer, C., and Felber, P. Automatic data partitioning in software transactional memories. In *Proc. of the Symposium on Parallelism in Algorithms and Architectures (SPAA)* (New York, NY, USA, 2008), ACM, pp. 152–159.

[28] Romano, P., Carvalho, N., and Rodrigues, L. Towards distributed software transactional memory systems. In *Proc. of the WShop on Large-Scale Distributed Systems and Middleware (LADIS 2008)* (Watson Research Labs, Yorktown Heights (NY), USA, Sept. 2008). (invited paper).

[29] Romano, P., Palmieri, R., Quaglia, F., Carvalho, N., and Rodrigues, L. On speculative replication of transactional systems. Tech. Rep. 38/2009, INESC-ID, July 2009.

[30] Shavit, N., and Touitou, D. Software transactional memory. In *Proc. of the Symposium on Principles of Distributed Computing (PODC)* (Ottawa, 1995), ACM Press.

[31] Steffan, J. G., Colohan, C. B., Zhai, A., and Mowry, T. C. A scalable approach to thread-level speculation. In *Proc. of the International Symposium on Computer Architecture (ISCA)* (New York, NY, USA, 2000), ACM, pp. 1–12.

[32] Xu, J., Zhao, M., Fortes, J., Carpenter, R., and Yousif, M. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the International Conference on Autonomic Computing (ICAC)* (2007), IEEE Computer Society, p. 25.

[33] Yoon, E. Hadoop Map/Reduce Data Processing Benchmarks. Hadoop Wiki., http://wiki.apache.org/hadoop/DataProcessingBenchmarks.

[34] Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P. K., and Currey, J. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In *Proc. of the Symposium on Operating System Design and Implementation (OSDI)* (2008).