Elastic Scaling for Transactional Memory: From Centralized to Distributed Architectures

D. Didona* P. Felber† D. Harmanci† P. Romano* J. Schenker†

1 Context and Motivation

Transactional memory (TM) [**?**] has been widely studied over the last decade as it provides a scalable and easy-to-use alternative to locks. Over the last years, a wide body of literature has been published on TM, and several variants have been developed, including hardware-based (HTM), software-based (STM), and distributed (DTM) [**?**]. One of the key results highlighted by existing research is that, independently of the nature of the synchronization scheme adopted by a TM platform, its actual performance is strongly workload dependent and affected by a number of complex, often intertwined factors (e.g. duration of transactions, level of data contention, ratio of update vs read-only transactions).

This work is based on the belief that most workloads have a *natural degree of parallelism*, i.e., there is a workload-specific threshold below which adding more threads will improve transaction throughput, and over which new threads will not help and might even degrade performance because of higher contention and aborts rates, even if sufficiently many cores are available.

In this position paper we discuss on the importance of adapting the concurrency level to the workload (which we call elastic scaling) in various application settings. Note that related problems have been already addressed in previous research. For instance, Felber et al. [**?**] tackled the problem of how to tune at run time the number of "locks" and their coverage of the whole address space in the TINYSTM library. Wang et al. [**?**] exploit machine learning techniques to select which STM implementation to adopt on the basis of the application workload. Ansari et al. [**?**] adapt the parallelism of STM applications in order to maintain the transaction abort rate under a predefined level. In the area of replicated relational databases, recent works [**?**, **?**] have proposed mechanisms for supporting elastic scaling, namely automatically adapting, in face of varying workloads, the number of nodes the platform is deployed onto. So far, however, limited attention has been devoted to dynamically identifying the optimal degree of parallelism for a (D)TM platform, namely the degree of local (i.e., number of active threads) and possibly global (i.e., number of nodes in a DTM) concurrency that maximizes the throughput of complex (D)TM applications.

In this paper we present experimental results obtained considering two extreme scenarios: on shared-memory systems with a low-level STM library written in C, and in distributed systems with a high-level DSTM infrastructure written in Java.

We first show that realistic benchmarks exhibit widely different performance depending on the degree of parallelism, and that adapting the number of threads at runtime can improve performance of some applications over any execution with a fixed number of threads. By applying small modifications to the benchmarks and the underlying STM runtime in a shared-memory system, one can straightforwardly optimize the concurrency level using exploration-based on-line optimization techniques, e.g., using hill climbing or gradient descent algorithms.

In distributed settings, however, the cost of testing configurations with a different number of threads (i.e., nodes) is prohibitive, as it requires transferring state, generates additional traffic, and takes orders of magnitude more time than in centralized settings. Therefore, in such settings, one should instead rely on modeling techniques to predict the expected gains from adding or removing nodes for adapting the concurrency level.

We argue that, to cover the complete spectrum of TM systems, one should combine exploration-based and model-driven methods: exploration-based techniques allows us to perform accurate local observations, while models help predicting the evolution of performance at a large scale. In other words, on-line exploration can improve accuracy of models, while models can improve scalability of on-line techniques. We present experimen-

*Instituto Superior Técnico / INESC-ID, Portugal.
†University of Neuchâtel, Switzerland.

tal results that back our claims in both centralized and distributed settings, and open the way to further research in adaptive mechanisms for elastic scaling of TM within and across nodes.

2 Exploration-based Scaling

Ideally, it is desired to run applications at their natural degree of parallelism, i.e., a point where each thread does "sufficient" useful work without inducing "too much" contention. The exact definition of both quantities varies depending on the context and it is generally not obvious to find this natural degree of parallelism for a given application. For workloads where contention due to data synchronization does not change throughout the application execution, the best level of parallelism can be found offline by repeatedly restarting the application with different sets of parameters. However, the contention a workload generates may vary during the lifetime of the application, i.e., the natural degree of parallelism represented by the workload varies as the application executes. Hence, a general solution to this problem would need to track the workload generated by the application on-line.

Before dwelling on the actual exploration algorithm, let us briefly consider the benefits of such adaptive techniques on the application `intruder`, part of the widely used STAMP benchmark suite [**?**]. This application emulates a signature-based network intrusion detection system and exhibits a workload that evolves over time.

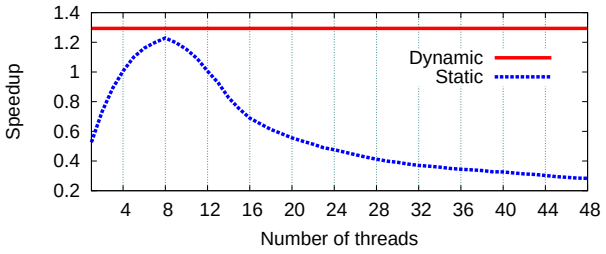


Figure 1: Speedup of the `intruder` benchmark as compared to sequential (non-STM) version, using static and dynamically evolving numbers of threads.

Figure **??** indicates the performance of the benchmark when executed with varying number of threads (dashed line), or when dynamically changing the number of threads (plain straight line corresponding to a constant value). One can observe that performance is significantly better when dynamically adapting concurrency than with any fixed number of threads. Note that the experiment was run on a 48-core machine, i.e., the number of physical cores was not the limiting factor.

Our exploration-based approach performs on-line monitoring of key performance metrics. It allows us to find the natural degree of parallelism of an application by running it with an iterative algorithm controlling its concurrency level. The algorithm terminates when all the work to be performed by the application is accomplished. Each iteration of the algorithm has three phases, as illustrated in Figure **??**:

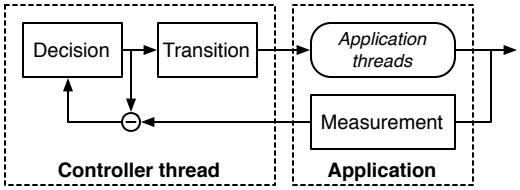


Figure 2: The principle of the exploration-based algorithm is akin a feedback control loop. The three phases are shown in rectangles with solid lines.

- **Measurement phase:** In this phase, the application runs with a fixed number of threads. Key performance metrics (numbers of commits and aborts) are measured during a certain time period. The commit rate gives an indication of raw transaction throughput, while the abort rate is a good measure of contention.

- **Decision phase:** In this phase the algorithm decides between two actions: increasing or decreasing the number of threads. If the last measurement phase shows improvements in terms of commit rate, the action performed in the previous iteration is repeated (addition or removal of threads); otherwise, it is reversed. The decision taken in this phase corresponds to a *hill climbing* technique maximizing transaction throughput, i.e., commit rate. The technique explores configurations in the vicinity of the current one by dynamically adding or removing threads, until a (local) maximum is reached. Even when reaching such a point, the configuration is tested for adapting to possible variations in the workload that would shift the optimal configuration(s) [1].

- **Transition phase:** An external controller thread adds or removes threads to/from the application according to outcome of the decision phase.

For faster adaptation to the workload, we tune the duration of the measurement phase such that we have sufficiently many samples (i.e., commits) to take sound decisions but without wasting too much time. In this way, the algorithm reacts fast by quickly collecting measurements with applications composed by short transactions while

[1] One should note at this point that none of the benchmarks we experimented with (STAMP applications and various micro-benchmarks) exhibits multiple maxima when observing throughput as a function of the number of threads, up to the hardware limit of our 48-core test machine.

it will take more time to adapt, but will still take correct decisions, for applications with long transactions.

Inserting the application inside the iterative algorithm required us to introduce (i) code observing performance, for the measurement phase, and (ii) a *controller thread* that performs decision and transition phases to modify the parameters of the application based on the measured performance. This extra thread controls the main execution loop of the application and can add or remove as many transactional threads as required during run time.

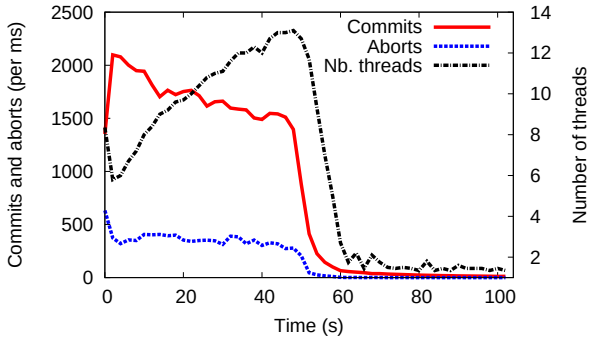


Figure 3: Evolution of the number of threads with the `intruder` benchmark using exploration-based scaling.

Figure **??** shows the behavior of the exploration-based algorithm with the `intruder` application. As one can observe, the number of threads (values averaged over 2-second periods for clarity) increases in the first half of the execution to reach 13, then drops sharply to account for changes in the workload. The last part of the execution uses only few threads, which reduces the commit throughput but limits contention and avoids most aborts.

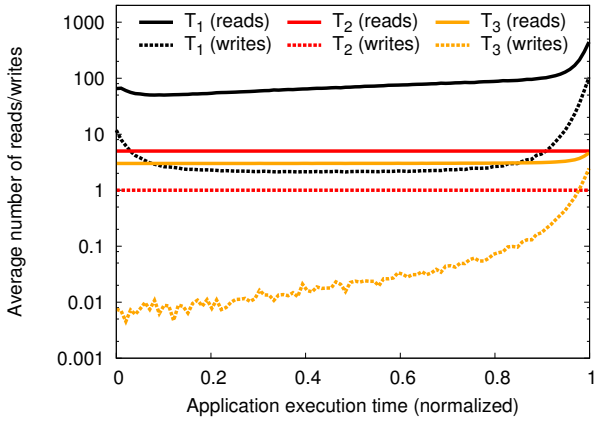


Figure 4: Variations in transaction sizes during execution of the `intruder` benchmark.

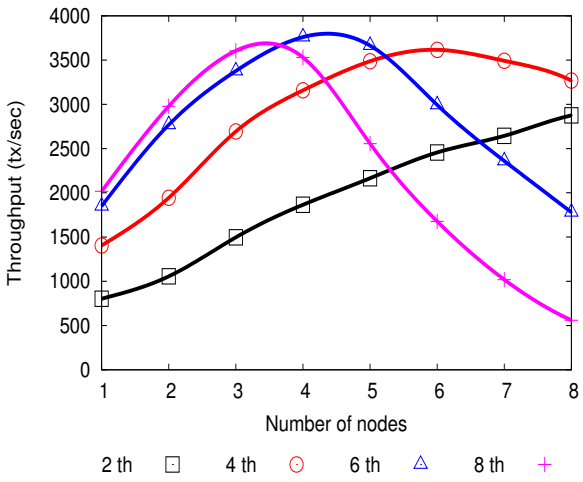
To better understand what triggers such changes in the workload, we show in Figure **??** the variations in transaction lengths, as reported by the size of the read and write sets, during the execution of `intruder`. Values are averaged over groups of 10,000 transactions and sizes are shown on a logarithmic scale. The application repeatedly executes a sequence of 3 transactions. Two of them, denoted as T_2 and T_3 in the graph, do not vary much over time. The third one, T_1, exhibits an interesting trend that explains why our approach is so effective: transactions read more and more data, with a sharp spike in the end, while their number of writes first decreases before stabilizing and increasing steeply in the end. Therefore, the last transactions to execute are very long and, hence, are expected to encounter much contention. Limiting concurrency increases the likelihood of commit and, in turn, improves overall performance.

Note that we modified and experimented with other applications of the STAMP benchmark suite. We found out that, while `intruder` benefits most from dynamic adaptation of the concurrency level because of the wide variations in its workload, our exploration-based algorithm is also effective with other applications and can quickly find the optimal number of threads.

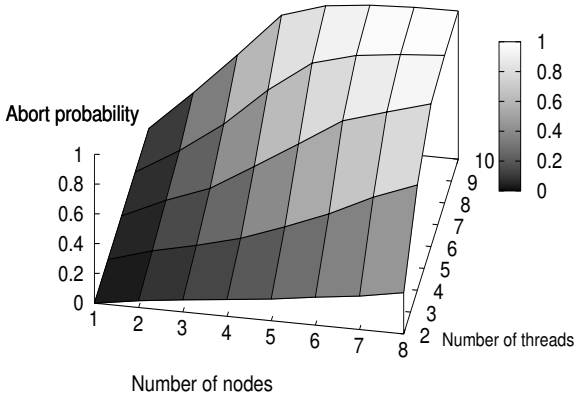
3 Model-driven Scaling

When considering DTM platforms [**?**], the problem of identifying the natural degree of parallelism for a given workload grows in complexity, turning into a bi-variate optimization problem. In distributed settings, in fact, the degree of concurrency is affected not only by the number of threads deployed on each node, but also by the number of nodes composing the distributed TM platform. The plots in Figure **??** clearly show the complex, non-linear interdependencies that can arise between the maximum throughput achievable by a DTM platform and both the number of nodes over which it is deployed and the number of threads running on every node. The DTM platform used in this study is Infinispan [**?**], a popular in-memory distributed transactional key-value store implemented in Java. Infinispan is the reference NoSQL data platform and clustering solution for JBoss AS, one of the market leading open source J2EE application servers, where it is used for replicating HTTP and EJB sessions states, as well as a second-level cache for Hibernate [**?**]. Infinispan supports different data replication/distribution strategies [**?**], and in this experiment we configured it to use a two-phase commit (2PC) based full-replication protocol. The workload was generated using the well known TPC-C benchmark [**?**], and we used, as experimental testbed, a cluster of 8 servers, each equipped with 8 cores and interconnected via a private Gigabit Ethernet.

As shown in Figure **??**, there is a non-trivial correlation between throughput, number of nodes in the system and number of threads per node. The plots highlight that, when fixing the number of nodes, performance varies significantly depending on the amount of active threads on each node. This depends on the intertwining



(a) Throughput (Committed Transactions per second)



(b) Transaction Abort Probability

Figure 5: Analyzing the performance of different configurations of the TPC-C benchmark when deployed on distributed TMs of variable sizes and local concurrency levels.

of effects associated with contention at both the data and physical level. As the number of nodes in the system increases, the latency of the 2PC-based replication protocol grows accordingly, which, in turn, leads to an increase of the time threads spend idle waiting for the completion of replica synchronization activities. Clearly, a way to increase the utilization level of the cores available at each node is to activate additional threads on each node. However, this can lead to a raise of the level of data contention that can outweigh the performance gains achievable by pursuing higher levels of utilization of the available computational resources. As depicted by Figure **??**, this is precisely what happens in this case, as increasing the number of active threads per nodes in configurations with more than 4 nodes leads to thrashing due to the rapid growth of the transaction abort rate.

Note that the problem of identifying the natural degree of parallelism in a DTM is not only more complex due to the inherent growth in the dimensionality of the

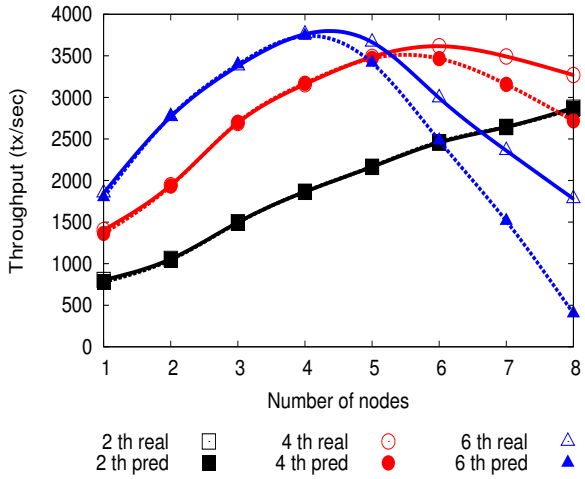


Figure 6: Accuracy of TAS performance predictions.

solution space. An additional factor that contributes to exacerbate the complexity of this problem is that, unlike in non-distributed TMs, exploratory steps that are at the basis of on-line learning techniques (such as the one presented in Section **??**) can be very costly. The addition of nodes to a DTM platform, in fact, requires expensive state-transfer phases aimed to ensure that the joining nodes install an updated snapshot of the TM before starting processing transactions. The actual duration of the state transfer phase depends on the extent of the topology change and on the amount of data maintained by the TM but, as shown in [**?**], it is not uncommon for it to take up to several minutes in complex applications deployed on large clusters. Throughout these phases, the DTM platform can be subject to a significant additional load [**?**, **?**] and suffer from severe performance degradation [**?**].

In the light of these considerations, we argue that mechanisms for self-tuning the degree of concurrency in distributed TM platforms should employ model-driven techniques capable of predicting the performance of the TM platform when deployed on a different number of nodes. To the best of our knowledge, there is no solution which tackles the problem of determining simultaneously the optimal degree of local and global concurrency level in a DTM.

TAS (Transactional Auto Scaler) [**?**] represents a first step towards filling this gap. TAS relies on a performance prediction methodology based on the joint usage of analytical and off-line trained machine learning models. The analytical models employed by TAS exploit the knowledge of the dynamics of the concurrency control/replication algorithm in order to forecast the effects of data contention via a white-box approach. Black-box machine-learning models, on the other hand, are employed to forecast the impact on performance due to shifts in the utilization of system level resources (e.g., CPU and network) caused by variations of the scale of

the platform. This avoids modeling the interactions with system resources, which is a time-consuming task given the complexity of current hardware architectures, and also makes this methodology viable for virtualized environments where little or no knowledge is available on the underlying physical infrastructure. In order to build an initial knowledge base for training the machine learners, TAS relies on a suite of off-line synthetic benchmarks that generate a breadth of heterogeneous transactional workloads (e.g., in terms of size of the generated messages, memory footprint at each node, throughput and execution duration of the transactional business logic), and which are executed under varying local and global degree of concurrency.

The plots in Figure **??** compare the actual performance of the system with the performance predictions generated by TAS. To this end, TAS was provided with information concerning utilization of logical and physical resources while running in a configuration with 2 nodes and 2 threads, and was queried in order to obtain performance forecasts while varying the number of nodes and threads in the system. The plots highlight the ability of TAS to forecast correctly not only the performance trends for the considered workload as a function of the degree of concurrency in the system, but also the configuration that maximizes system's throughput. On the other hand, the plots also highlight the existence of system configurations in which the accuracy of the performance predictions output by TAS can decrease significantly. We argue that this can depend on two main factors.

The first one is that the analytical model employed in TAS for capturing the effect of data contention relies on assumptions whose validity can be challenged at extremely high levels of contention. This claim is confirmed by correlating the plots of Figure **??** with the abort probability reported in Figure **??**, and noting that the accuracy of TAS prediction degrades in regions where the system's throughput starts dropping in settings exhibiting extreme levels of data contention (higher than 80%); note that this loss of accuracy in such scenarios is natural for analytical models, as shown in[**?**].

The second factor that may contribute to degrade the quality of TAS' predictions is imputable to the loss of accuracy of its machine-learning based models. Off-line learners' predictive power strongly depends on the representativeness of the samples observed during their training phase; however, the parameters' space associated with all possible workloads and concurrency levels in a DTM is extremely vast, thus exploring it in an exhaustive fashion is prohibitive. Thus, in TAS we use a uniform sampling of the features' space, and limit the duration of the off-line benchmarking to one hour. The downside of this approach is that it can lead to a degradation of the predictions' accuracy especially in those regions in which small fluctuations of the input variables lead to strong variations of the output variables (e.g., in case new threads are added in high CPU utilization scenarios).

4 Towards a Combined Approach

The self-tuning approaches presented in the two previous sections rely on opposite methodologies for identifying the optimal degree of concurrency of (D)TM applications; these two adaptation methodologies have complementary strengths and weaknesses.

As shown in Section **??**, exploration-based techniques shine in determining the optimal degree of local concurrency, avoiding the usage of time-consuming off-line training phases, or the design and validation of analytical models, which are instead necessary for model-driven techniques, such as the one presented in Section **??**. The other side of the coin is that, approaches based exclusively on exploration-based techniques result cumbersome and unattractive to determine the optimum number of nodes to use in a DTM, due to the high overheads associated with state transfer.

Model-based performance forecasting techniques, on the other hand, lend themselves better to tackle this problem. However, their accuracy is ultimately dependent on the quality of the models they employ, which can be affected by several factors, as pointed out in Section **??**.

A possible way to combine the two approaches might be to use model-based techniques to initialize the knowledge base of on-line learning mechanisms. This would allow to guide their exploration in order to avoid dwelling in regions that are clearly identified as unfavorable by model-based oracles, or to adjust dynamically the granularity adopted for on-line exploration. A similar approach has been proposed in different application domains, such as tuning the level of message packing in group communication systems [**?**], and has shown to boost considerably the convergence speed of pure on-line learning schemes. We are not aware, however, of applications adopting such techniques to identify the optimal degree of concurrency for (D)TM platforms.

Another way of composing exploration-based and model-based self-tuning approaches would be to use them according to a *divide-and-conquer* fashion. With this approach, the problem of identifying the optimal degree of concurrency in a DTM platform would be decoupled into two simpler sub-problems (namely determining the optimal number of distributed nodes vs. the optimal number of threads active per node), which could then be solved using the most appropriate methodology. For instance, model-driven techniques, such as TAS, could trigger a global reconfiguration of the system, which

could then be followed by exploration-based approaches aimed at adjusting the local concurrency level at each node in order to compensate possible errors of the model. Clearly, the key challenge here is devising methodologies capable of assessing, in an effective way, the quality of the solution identified via the divide-and-conquer approach.

Another research direction that we are currently exploring concerns how to incorporate the feedbacks gathered using on-line exploration techniques into performance forecasting models, with the ultimate goal of continuously enhancing their predictive power and overall robustness. Conventional white-box models of transactional platforms [**?**, **?**, **?**, **?**], for instance, are built on rigid assumptions (e.g., exponential arrival rate of lock-requests) and are not conceived to adapt themselves to cope with scenarios in which such assumptions are not (or are only partially) met.

In conclusion, we argue that further research is required in order to design hybrid methodologies, combining exploration-based and model-driven approaches with the goal of letting the two cooperate in synergy and having the strengths of one compensate the weaknesses of the other, and vice versa.

References

[1] ANSARI, M., LUJÁN, M., KOTSELIDIS, C., JARVIS, K., KIRKHAM, C., AND WATSON, I. Robust adaptation to available parallelism in transactional memory applications. *Transactions on High Performance and Embedded Architectures and Compilers 3*, 4 (2008).

[2] CAO MINH, C., CHUNG, J., KOZYRAKIS, C., AND OLUKOTUN, K. STAMP: Stanford transactional applications for multi-processing. In *IISWC* (2008), pp. 35–46.

[3] CICIANI, B., DIAS, D. M., AND YU, P. S. Analysis of replication in distributed database systems. *IEEE Trans. Knowl. Data Eng. 2*, 2 (1990).

[4] COUCEIRO, M., ROMANO, P., CARVALHO, N., AND RODRIGUES, L. D2stm: Dependable distributed software transactional memory. In *PRDC'09: Proceedings of the 15th Pacific Rim International Symposium on Dependable Computing* (Shanghai, China, Nov. 2009).

[5] DI SANZO, P., CICIANI, B., QUAGLIA, F., AND ROMANO, P. A performance model of multi-version concurrency control. In *MASCOTS* (2008).

[6] DIEGO DIDONA, PAOLO ROMANO, SEBASTIANO PELUSO, FRANCESCO QUAGLIA. Transactional Auto Scaler: Elastic scaling of NoSQL transactional data grids. Tech. Rep. 50/2011, INESC-ID, December 2011.

[7] FELBER, P., FETZER, C., AND RIEGEL, T. Dynamic performance tuning of word-based software transactional memory. In *PPoPP 08: 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Feb. 2008), pp. 237–246.

[8] HARRIS, T., LARUS, J. R., AND RAJWAR, R. *Transactional Memory*, 2nd edition ed. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publisher, 2010.

[9] HERLIHY, M., AND MOSS, J. E. B. Transactional memory: Architectural support for lock-free data structures. In *ISCA* (1993), pp. 289–300.

[10] JBOSS. JBoss Hibernate. http://www.hibernate.org/, 2011.

[11] NARASIMHA RAGHAVAN, ROMAN VITENBERG. Balancing the communication load of state transfer in replicated systems. In *SRDS* (2011), pp. 41–50.

[12] PAOLO ROMANO, MATTEO LEONETTI. Self-tuning Batching in Total Order Broadcast Protocols via Analytical Modelling and Reinforcement Learning. In *IEEE International Conference on Computing, Networking and Communications, Network Algorithm & Performance Evaluation Symposium (ICNC)* (January 2012).

[13] QI ZHANG, LUDMILA CHERKASOVA, EVGENIA SMIRNI. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of the Fourth International Conference on Autonomic Computing* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 27–.

[14] RED HAT/JBOSS. JBoss Infinispan. http://www.jboss.org/infinispan, 2011.

[15] RICARDO JIMÉNEZ-PERIS, MARTA PATIÑO-MARTÍNEZ, GUSTAVO ALONSO. Non-intrusive, parallel recovery of replicated data. In *SRDS* (2002), pp. 150–159.

[16] SAEED GHANBARI, GOKUL SOUNDARARAJAN, JIN CHEN, CRISTIANA AMZA. Adaptive learning of metric correlations for temperature-aware database provisioning. In *Proceedings of the Fourth International Conference on Autonomic Computing* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 26–.

[17] TPC COUNCIL. TPC-C Benchmark. http://www.tpc.org/tpcc, 2011.

[18] WANG, Q., KULKARNI, S., CAVAZOS, J., AND SPEAR, M. Towards applying machine learning to adaptive transactional memory. In *6th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT)* (June 2011).

[19] YU, P. S., DIAS, D. M., AND LAVENBERG, S. S. On the analytical modeling of database concurrency control. *J. ACM 40* (1993).