Complexity-Effective Contention Management with Dynamic Backoff for Transactional Memory Systems

Seung Hun Kim, *Student Member, IEEE*, Dongmin Choi, Won Woo Ro, *Member, IEEE Computer Society*, and Jean-Luc Gaudiot, *Fellow, IEEE*

Abstract—Reducing memory access conflicts is a crucial part of the design of Transactional Memory (TM) systems since the number of running threads increases and long latency transactions gradually appear: without an efficient contention management, there will be repeated aborts and wasteful rollback operations. In this paper, we present a dynamic backoff control algorithm developed for complexity-effective and distributed contention management in Hardware Transactional Memory (HTM) systems. Our approach aims at controlling the restarting intervals of aborted transactions, and can be easily applied to the various TM systems. To this end, we have profiled the applications of the STAMP benchmark suite and have identified those "problem" transactions which repeatedly cause aborts in the applications with the attendant high contention rate. The proposed algorithm alleviates the impact of these repeated aborts by dynamically adjusting the initial exponent value of the traditional backoff approach. In addition, the proposed scheme decreases the number of wasted cycles down to 82% on average compared to the baseline TM system. Our design has been integrated in LogTM-SE where we observed an average performance improvement of 18%.

Index Terms—Transactional Memory, Contention Management, High Contention, Exponential Backoff

1 Introduction

The Transactional Memory (TM) [1] paradigm has been developed to provide efficient synchronization in parallel environments by lowering the programming complexity required for synchronization. The main effect of a TM system is that it enables the concurrent execution of threads without using fine-grained locks. Therefore, any problems caused by the traditional lock concept for critical sections are eliminated. This allows TM to provide a stable and convenient interface for parallel programming as well as to offer higher performance.

In TM systems, if two or more transactions access the same data element and at least one of them is a write operation, there is a conflict. When a conflict occurs, only one transaction should be allowed to commit, while all the others must be made to rollback to their starting point [1]. This means that transactions can be either completely executed on a *commit* or can be flushed and retried on an *abort* in order to guarantee atomicity. In order to achieve correct rollback, a properly designed version management policy is needed.

Previous projects have proposed mechanisms for conflict detection and version management in hardware [2], [3], [4], [5], [6], software [7], [8], [9], [10], or hybrid hardware-software [11], [12], [13], [14]. Although these projects have addressed several issues to improve TM, there still exist additional significant drawbacks caused by high contention. Contention is a state of crowded transactions in a shared memory space [15]. High contention consequently introduces transaction conflicts which should be detected and resolved. Applying an appropriate conflict detection method [16] and a proper conflict resolution policy [17] is important since it eventually contributes to reducing the number of transaction conflicts.

In order to lower contention and avoid performance crippling conflicts, transaction scheduling has also been studied [18], [19], [20], [21], [22], [23], [24]. Most of the proposed methods strongly rely on a prediction which is based on the history of the transaction. With Adaptive Transactional Scheduling (ATS), one stores and holds aborted transactions in a centralized queue in order to prevent conflicts [22]. Proactive Transactional Scheduling (PTS) enforces thread swapping at conflict time and executes another thread which contains non-conflicting transactions [23]. In addition, Bloom Filter Guided Transaction Scheduling (BFGTS) has been introduced to enhance the performance of PTS [24]. Both aim at distributing conflicting transactions according to an analysis of the previous conflicts. However, these approaches usually impose additional software/hardware overhead and consequently cause additional power consumption.

The research in this paper has been motivated by the observation that a backoff algorithm can still be a complexity-effective contention management method since it does not require additional hardware such as registers, caches, and interconnection networks that are assumed in PTS [23] and

- *S.H. Kim and W.W. Ro are with the School of Electrical and Electronic Engineering, Yonsei University, Sinchon-dong, Seodaemun-gu, Seoul, Korea.*
  E-mail: kseunghun@gmail.com, wro@yonsei.ac.kr
- *D. Choi is with Samsung Electronics, #94-1, Imsoo-dong, Gumi-City, Gyeongbuk, Korea.*
  E-mail: dm.choi@samsung.com
- *J-L. Gaudiot is with the Department of Electrical Engineering and Computer Science, University of California, Irvine.*
  E-mail: gaudiot@uci.edu

BFGTS [24]. On the other hand, we have also found that not every application suffers from a large amount of conflicting transactions. Therefore, incurring an excessively high overhead only for contention management might not be an optimal system-wide solution since the overhead is incurred for including the non-conflicting ones.

However, as pointed out in the literature, the original randomized backoff algorithm might suffer from repeated contention under similar circumstances [23]. To overcome this drawback, we propose here a dynamic backoff control mechanism. Applying a proper exponent value to the traditional exponential backoff algorithm and dynamic control on the initial value provides a very effective contention management mechanism and brings an important performance improvement over the traditional backoff algorithm. We evaluate its performance on benchmarks in the STAMP [25] suite and the SPLASH-2 [26] suite, as well as microbenchmarks of GEMS [27] and observed an 18% performance improvement on average when using our modified backoff algorithm. With the proposed scheme, the number of retries is reduced and the number of wasted cycles in transactions is also decreased. The contributions of the paper can be summarized as follows:

- A proposal and demonstration of complexity effective contention management,
- A detailed analysis of the contention rate of transactions, and
- The development of a metric that demonstrates the effectiveness of contention management.

The rest of the paper is organized as follows: Section 2 describes previous research on contention management with exponential backoff. Section 3 presents our proposed dynamic backoff control, which is our new contention management algorithm. The experimental setup for performance evaluation and verification is described in Section 4. Results and analysis are shown in Section 5. Finally, Section 6 concludes the paper.

2 Contention Management - Prior Work

In this section, we provide background studies and related work for our proposed idea.

2.1 Problem Definition

One simple policy to control transaction contention consists in inserting a time delay (backoff) between *abort* and *restart*. The basic concept behind this approach is abort recovery, backoff, and restart, as shown in Fig. 1: if a transaction is restarted immediately after an abort, there is a high possibility that repeated aborts will occur, due to a conflict with the transaction which has survived in the previous conflict. Consequently, program execution cycles are wasted by the repeated aborts. A safe way to avoid repeated aborts is to wait a sufficient amount of time until the surviving transaction has committed; however, this waiting also could reduce the degree of available parallelism.

The optimal backoff delay is tightly coupled with the transaction size. For example, Transaction 1 and Transaction 2 in Fig. 2a experience a conflict and Transaction 2 is forced to abort. At that point, Transaction 2 undergoes a recovery procedure and has backed off before being restarted. Although there is some time delay due to the backoff, if Transaction 1 is too large to commit in that time, Transaction 2 will be aborted again. Then, the same procedure would be repeated so as to handle the second abort. The problem associated with this phenomenon is clearly observed considering Fig. 2b: the backoff delay in Fig. 2a is smaller than that in Fig. 2b. However, the execution time in Fig. 2b is shorter due to the two consecutive aborts in Fig. 2a.

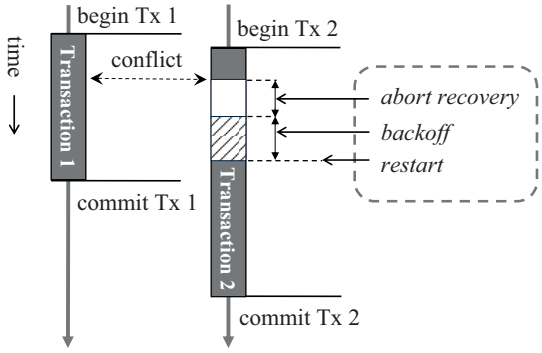
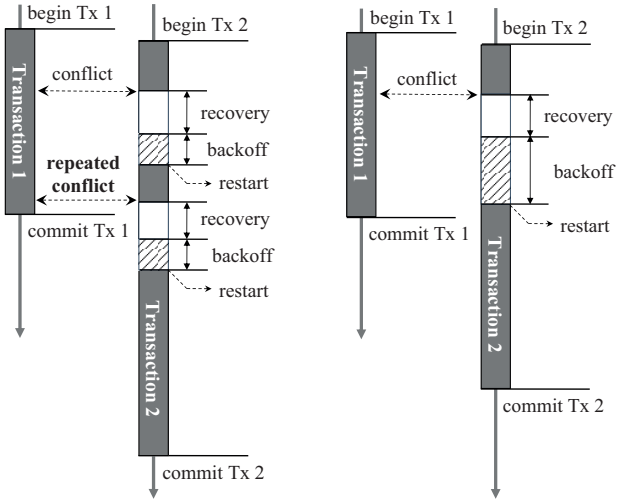


Fig. 1. Concept of abort recovery, backoff, and restart

In fact, large transactions cause high contention and require an advanced contention management policy. Traditional benchmark suites such as SPLASH-2 [26] and SPEComp [28] have small transactions and small amounts of synchronization. However, more recent benchmark suites such as STAMP [25] include high contention rates with large transactions; the benefit of the proposed idea with large-sized transactions will be shown.



(a) Repeated conflict with a short backoff

(b) Single abort with a long backoff

Fig. 2. Effect of backoff in large transactions

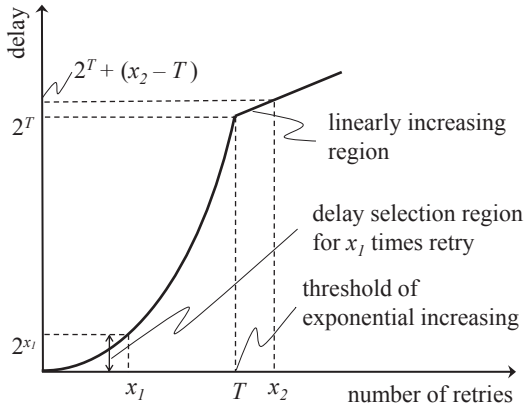


Fig. 3. Concept of the truncated binary exponential backoff algorithm

2.2 Exponential Backoff

Various backoff algorithms have been investigated in the area of computer networks [29]. The main idea is to delay each of the conflicting nodes which intend to use a shared network resource. Using a backoff algorithm makes possible an effective congestion control approach which can improve the network throughput. When applying to TM the traditional backoff algorithm used in the computer networks, the transactions on various threads can be regarded as nodes and the shared variables inside the transaction can be considered restricted resources.

In fact, both exponential backoff and linear backoff have been widely used for contention management in TM systems [23]. Several TM approaches adopted the truncated binary exponential backoff [30] shown in Fig. 3, where the amount of delay is randomly selected between 0 and the maximum value that increases exponentially with the number of aborts. If the exponent value exceeds a certain threshold, the upper limit to select a delay is increased linearly to prevent any excessive waiting. In other words, if the number of retries is x which is smaller than the threshold T, the delay is randomly selected between *0* and 2^x. Otherwise, the maximum value of the range is computed as $2^T + (x - T)$.

The performance of the binary exponential backoff approach has been analyzed by Kwak *et al.* [29]. They have assumed a fixed number of nodes which always have packets to transmit and have derived a mathematical model of exponential backoff. Their results show that exponential backoff provides fair service in terms of throughput. However, there are some fundamental differences when we consider TM. In fact, the contention rate in a program is not constant over the whole execution due to the characteristics of transactional workload. Therefore, one cannot suppose that contention in TM systems is identical to the network environment where all nodes are assumed to always have packets to transmit with a given probability.

Therefore, traditional randomized exponential backoff might not provide an optimal backoff solution for TM systems. In addition, having zero as the initial exponent value may make the backoff delay too short. Despite all these disadvantages, a backoff-based approach is still a simple and cost-effective solution [23]. As mentioned earlier, the main reason behind the effectiveness is the simplicity of the underlying hardware. For example, additional registers are needed in PTS to record currently running transactions and the size of read and write set. Bloom filters are also necessary to predict the conflict [23]. Furthermore, most contention management methods are centralized, which causes unavoidable modifications or additions to the interconnection network. On the contrary, backoff based contention management does not require any change since the mechanism operates in a distributed manner. In fact, hardware complexity and power consumption will become important issues as far as implementation is concerned [31]. This means that our proposed method can provide a lightweight solution by reducing the implementation overhead [32]. In addition to the simplicity in hardware, there is no software overhead which a complicated conflict prediction algorithm would bring.

2.3 Related Work

Scherer III and Scott presented nine ad-hoc contention managers to efficiently select transactions to be aborted in Dynamic Software Transactional Memory [33]. Also, they provided a detailed analysis regarding five of the previously proposed managers and proposed a new approach [34]. The new method takes into consideration not only the starting point of each transaction but also the amount of computation which has been processed.

In addition to the contention managers, *transaction scheduling* algorithms have been proposed to control the starting points of transactions. Ansari *et al.* proposed a method which controls the number of concurrently executable transactions by measuring the rate of transactions committed in a sample period [18]. Dolev *et al.* introduced a serialized approach with a per-core transaction queue: transactions that are likely to experience conflicts are assigned to the same core [19]. Ansari *et al.* introduced a similar idea in [20]; however, they added load balancing considerations. Sönmez *et al.* developed a different concurrency control mechanism based on the *hot* variables that caused frequent aborts [21]. In their method, optimistic or pessimistic control is selected at run time. Dragojevic *et al.* performed a theoretical analysis of transaction scheduling [35]. In addition, Maldonado *et al.* proposed kernel-level support for transactional scheduling [36]. The scheduling methods developed in STM require software overhead and impose additional design complexity [37].

Transaction scheduling has been studied in Hardware Transactional Memory (HTM) as well. Yoo and Lee proposed ATS [22] to prevent transaction conflicts. In ATS, aborted transactions are delayed in a centralized queue according to *contention intensity* which represents the degree of contention. When the contention intensity is larger than a threshold, transactions are stored in the queue. The *contention intensity* is maintained in each thread and is updated when a commit or an abort occurs. One of the main disadvantages of ATS is that it serializes execution of all transactions in the centralized queue [24].

PTS [23] by Blake *et al.* provides more concurrency than ATS. In PTS, the scheduler arranges non-conflicting threads by

use of the conflict predictor and swaps thread when a conflict occurs. In other words, independent transactions that are not in conflict with each other should be executed in parallel. The PTS algorithm identified hot spots which cause performance degradation in applications and showed that these hot spots can be predicted. The scheduler selects transactions around these hot spots and schedules those selected transactions based on the PTS algorithm. The basic concept is similar to the approach of Zilles and Baugh [38] in that both proposed to suspend conflicting transactions. However, PTS causes more software overhead than ATS since it requires frequent accesses to the conflict history to find independent transactions [24].

Blake *et al.* further proposed Bloom Filter Guided Transaction Scheduling (BFGTS) [24] which used *Similarity* as determined by the access patterns. To efficiently analyze the behavior of transactions, they proposed to use the Bloom filter [39]. In spite of a remarkable performance improvement, there still exists significant implementation overhead. Contrary to those prior works, Hasenfratz *et al.* proposed cost effective methods such as *QuickAdapter* and *AbortBackoff* [31] and introduced *RememberingBackoff* which moderated the amount of backoff by utilizing the number of aborts of the previous transaction.

3 Dynamic Backoff Control Algorithm

We have shown that having a delay before restarting aborted transactions helps to reduce the abort penalty. In fact, exponential backoff is widely used due to its simplicity for determining the amount of the delay. However, the traditional and simple backoff algorithm may not successfully manage contention. Our dynamic backoff control has been developed to mitigate these problems and we now present in this section its detailed mechanism.

3.1 Optimal Initial Value

Generally, the initial exponent value starts from 0 in order to find the minimal delay for contention management. This can be an effective approach, particularly for applications which have small transactions with a low contention rate. When the size of the transactions is small, the transaction which has made the other transactions abort can commit within a short delay. Therefore, the possibility for the aborted transactions to experience the conflict again becomes quite low. However, this observation now becomes different when the transactions become larger: having a short interval between abort and restart would cause repeated aborts.

We have profiled the STAMP benchmark suite in order to show the problems described in the previous paragraph with the traditional exponential backoff algorithm. Table 1 shows the distribution of the number of aborts required for each transaction to commit.

From the results, we observe that the low-contention applications such as *genome*, *kmeans*, and *ssca2* have extremely low percentage of repeated aborts as expected. More than 99% of transactions are committed without aborts. It implies that reducing the number of repeated aborts cannot efficiently improve the overall performance in those applications. On the other hand, *labyrinth* and *intruder* have experienced many repeated aborts. The percentage of transactions which experience more than two recovery operations is 48% in *labyrinth* and 32% in *intruder*. In *vacation* and *yada*, most transactions are committed without abort. However, the relative percentages of repeatedly aborted transactions are still higher compared to the low-contention applications.

TABLE 1
Percentage of Aborts until Commit (%)

Application	Number of aborts until commit					
	0	1	2	3	4	5 or more
genome	99.32	0.45	0.10	0.07	0.02	0.05
kmeans	99.56	0.38	0.05	0.01	0.00	0.00
ssca2	99.89	0.09	0.01	0.00	0.00	0.01
vacation	97.73	1.07	0.24	1.15	0.00	0.81
yada	92.75	2.03	1.19	0.73	0.40	2.90
labyrinth	51.27	0.63	4.43	4.43	2.53	36.71
intruder	63.86	5.36	3.02	1.63	1.65	24.47

Indeed, we have been motivated to develop a new backoff algorithm which can benefit applications with large transactions. First of all, we believe that an initial exponent value largely affects the overall performance of the backoff algorithm. Our first proposed idea is to set the initial exponent value as a positive number.

In fact, applying a positive integer number instead of zero for the initial exponent value helps to decrease the number of repeated aborts; this prevents having too little backoff delay. We have first increased the initial exponent value to a specific positive integer number k. The effect of this modification is depicted in Fig. 4. In the figure, the range of delay values is 2^k times larger than the original backoff. However, this method may cause unnecessarily long delays as well as have a large exponent value. To address this, we regulate the exponential increment threshold as *T-k*. Thus the actual value of the exponential increment is 2^T. In the proposed algorithm the upper value of exponential growth remains the same as in the original approach.

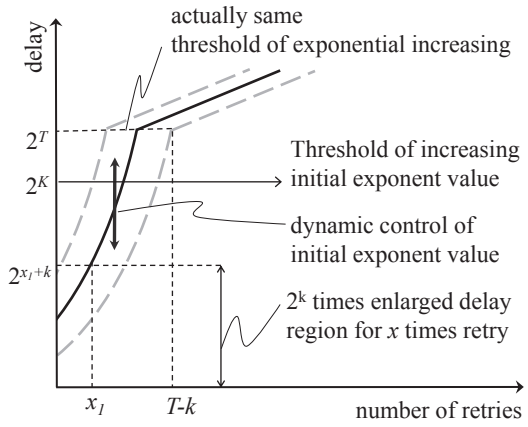


Fig. 4. Concept of the dynamic backoff control

3.2 Dynamic Control on Initial Value

While profiling the benchmark suite, we have also found that some transactions of high-contention applications are aborted more than 20 times before they commit. These repeated aborts obviously cause severe performance degradation. In this case, setting the initial exponent value large is helpful to reduce the number of repeated aborts. However, having a fixed initial value may not be an optimal solution for all applications. To this end, we propose a dynamic control on the initial value to provide more flexibility.

The proposed method dynamically regulates the initial exponent value monitoring the run-time behavior. With this approach, we intend to control the initial exponent value according to the amount of contention in each application. Table 2 shows the number of transactions that have repeated aborts for the application. We can recognize that there are no more than 11 successive aborts in the low-contention applications such as *genome* and *kmeans*. The basic idea of the dynamic control is to have a higher backoff delay for the high-contention applications. To this end, we define *Repeated Aborts Threshold* (RAT) to dynamically control the initial exponent value.

In the proposed design, when the number of repeated aborts exceeds the predefined RAT value, the initial exponent value is incremented by 1 from k_0; k_0 is the first initial exponent value which is decided by our heuristic approach. However, a large initial exponent value may cause excessive backoff delay in some cases. To prevent any excessive delay, we have developed two additional conditions. First, there is an upper bound to limit the excessive increase of the initial exponent value. By defining the threshold, our algorithm effectively lessens the contention without any side effect. Second, the initial exponent value is decreased by 1 when there are no transactions which experience more than 3 repeated aborts within a window of recently committed 10 transactions; the values 3 and 10 are heuristically determined and the lower bound of the initial exponent value is limited as zero. With this approach, each application eventually can adjust a proper initial value for the exponent. We have obtained those parameters using the profiling results and have found that the optimal RAT value is 10 and k_0 is 3.

TABLE 2
Number of Transactions for Large Number of Repeated Aborts

Application	Number of aborts until commit					
	9	10	11	12	13	14 or more
genome	5	1	0	0	0	0
kmeans	0	0	0	0	0	0
ssca2	0	0	0	0	0	0
vacation	0	1	1	1	6	18
yada	43	35	36	15	13	67
labyrinth	5	12	18	11	1	6
intruder	399	451	388	255	203	139

3.3 Necessary Wasted Cycles

In fact, we have defined four different categories of cycles that can be used to demonstrate efficiency of the proposed dynamic backoff algorithms. Those include: *stall cycles*, *backoff cycles*, *flush cycles*, and *abort cycles*. The stall cycles represent the wasted cycles in the stalling state of a transaction between conflict detection and conflict resolution. The backoff cycles are the cycles wasted during the backoff delay before the restart operation. Thus, the backoff cycles are determined directly by the backoff algorithm. The flush cycles represent the cycles corresponding to the work which is flushed due to the rollback operation. The abort cycles are the cycles spent for the abort operations such as restoring the old data and rolling back to the starting point of the transaction.

The amount of these four kinds of cycles are tightly coupled; for example, having a long delay on restart can decrease the other three cycle counts by reducing the number of aborts, but the increased delay is also a cause of performance degradation. Similarly, reducing the backoff delay may cause more aborts which might increase both flush and abort cycles. However, the exact effect of those cycles on the overall performance cannot be predicted before run time since every application has its own, different run time behavior.

From the above observation, one must consider the four kinds of wasted cycles together in the evaluation of the dynamic backoff control algorithm. We have named the summation of the four as *Necessary Wasted Cycles (NWC)*; those cycles represent necessary features in TM systems although the factors introduce overhead and some damage to the overall performance. NWC is expressed as (1).

$$\begin{aligned} NWC = {} & Stall\ cycles + Backoff\ cycles \\ & + Flush\ cycles + Abort\ cycles \end{aligned} \tag{1}$$

The first step in the development of the proposed dynamic backoff control is finding an appropriate initial exponent value. The evaluation is based not only on execution time but also on NWC. We have profiled the benchmark programs with various values for parameters; the results are presented in Section 5.

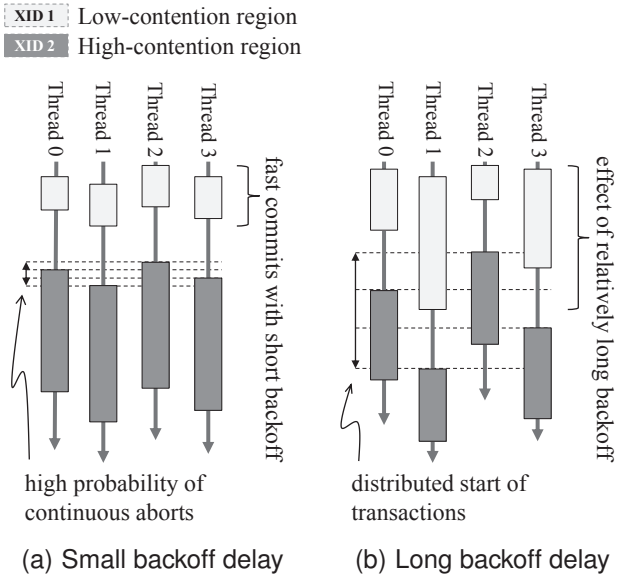
3.4 Effect of Different Transactional Regions

In general, there are several different transactional regions in an application. The regions mean the blocks of code in the application that are marked by the transaction begin and end instructions. In fact, it is possible that each transactional region shows a distinct characteristic with regard to the contention rate. Consequently, the number of repeated aborts might be different in each region. As an example, we have shown those values for six transactional regions of the *yada* application in Table 3. In the table, each of the transactional regions is represented with a distinct transactional id (XID). The third one experiences a large number of repeated aborts before commit. On the other hand, there is no abort on the second, the fourth, and the sixth transactional regions and there are relatively few repeated aborts on the first and the fifth regions.

From these properties, a modified dynamic control mechanism might provide better performance. This mechanism would apply a different initial exponent value and count

TABLE 3
Number of Repeated Aborts in Each Transactional Region

Application	Tx region	Number of aborts until commit															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15 or more
yada	XID 1	2908	98	21	12	12	18	11	21	23	19	15	8	1	0	0	0
	XID 2	3151	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	XID 3	1926	162	141	88	39	43	26	15	12	18	10	24	13	13	13	54
	XID 4	2597	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	XID 5	2500	26	6	3	6	7	7	9	12	6	10	4	1	0	0	0
	XID 6	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
intruder	XID 1	2808	208	93	30	28	24	45	48	70	152	160	84	13	5	0	0
	XID 2	2292	179	137	108	95	76	72	92	104	99	102	110	111	92	56	28
	XID 3	2100	217	111	46	63	78	85	115	115	148	189	194	131	106	48	7



(a) Small backoff delay (b) Long backoff delay

Fig. 5. Effect of backoff delay on different transactional regions

the number of repeated aborts in each transactional region separately; we call this method *private*. The private method probably provides a short delay on the region which shows a small number of aborts to reduce overall execution time. However, we have found that the global approach gives better results than the private method in some of the benchmark programs we have used. The reason can be found to be as follows:

Assume that there are two different transactional regions in an application, as depicted in Fig. 5. The first transactional region (XID 1) shows a small number of aborts whereas the second transactional region (XID 2) results in more aborts due to contention. It might sound reasonable to have a short backoff delay in the first transactional region. However, as shown in Fig. 5a, having a short backoff may cause those different threads to enter the second region (XID 2) nearby in time and to experience conflicts. On the other hand, having a long backoff delay in XID 1 as shown in Fig. 5b can make the entering points to XID 2 of multiple threads dispersed in time. This serves as a simple example to show the advantage of having a globally consistent initial exponent value. In fact, the main purpose of the backoff algorithm is to distribute transaction starting and our proposed method maximizes the effectiveness of the algorithm.

It might seem that if there is a barrier between two transactional regions in Fig. 5, the global approach might not work properly. However, the upper limit of the initial exponent value and the decreasing policy which is described in Section 3.2 help to prevent any performance degradation due to the global approach even in the programs with barriers. This is also demonstrated by the simulation results in Section 5.

The most important advantage of the global approach is implementation simplicity. In other words, the global approach does not require additional hardware or software to support an unbounded number of transactional regions. For the private approach to be applied in TM systems, there should be as many registers as the number of transactional regions to record the exponent value for each region. As a matter of fact, the number of the regions is not fixed or predictable. On the contrary, the global approach needs only one register.

4 Evaluation Methodology

This section describes the simulation environment and the benchmark programs that have been used to evaluate the proposed method.

TABLE 4
System Configuration

Feature	Description
Processor	In-order, 16 cores CMP, 2GHz, IPC = 1
L1 Cache	32 KB, 4-way, split, 64 byte block, 1-cycle latency
L2 Cache	8 MB, 8-way, shared, 64 byte block, 34-cycle latency
Directory	Full-bit vector sharer list, 6-cycle latency
Memory	8 GB, 450-cycle latency
Interconnect	64 byte links, 3-cycle link latency

TABLE 5
Transactional Characteristics and Input Parameters [25]

Application	Input Parameter	Tx Length	R/W Set	Tx Time	Contention
genome	-t15 -g256 -s16 -n16384	Medium	Medium	High	Low
kmeans	-p15 -m15 -n15 -t0.00001 -i random-n2048-d16-c16	Short	Small	Low	Low
ssca2	-t15 -s13 -i1.0 -u1.0 -l3 -p3	Short	Small	Low	Low
vacation	-c15 -n64 -q10 -u80 -r65536 -t4096	Medium	Medium	High	Medium
yada	-t15 -a20 -i 633.2	Long	Large	High	Medium
labyrinth	-t15 -i random-x16-y16-z3-n16	Long	Large	High	High
intruder	-t15 -a10 -l16 -n4096 -s1	Short	Medium	Medium	High

4.1 Simulation Environment

LogTM-SE is a baseline TM model since it is used by many HTM researchers [40]. We have implemented the proposed idea using GEMS 2.1.1 [27], which is driven by the Virtutech Simics 3.0.31 full-system simulator [41]. Simulation parameters are in Table 4.

The processor is a 16-core SPARC processor; each core has L1 instruction and L1 data caches. Each L1 cache is 32 KB with 64 byte cache blocks and 4-way associativity. The L2 cache is shared by the 16 cores and implemented with 8-way associativity. The L2 cache has 8 MB with 64 byte cache blocks. The cache coherence protocol is based on MESI and the *perfect* filter is applied for the signature [42]. The on-chip directory holds a bit-vector of sharers. The total size of the main memory is 8 GB and has a 450 cycle access latency. The operating system is Solaris 10.

The baseline system has eager conflict detection with signature and eager version management using *per-thread logs*. In this eager system, abort recovery causes much overhead to replace the new data in the memory with the old data kept in the log. Therefore, the abort penalty is proportional to the size of the transaction. Conflict resolution, which decides on abort or commit for each conflicting thread, is performed after conflict detection. After conflict resolution and recovery of an aborted transaction, a backoff is used to avoid repeated conflicts [43].

4.2 Benchmark Programs

We have used the STAMP [25] benchmark suite, two applications from SPLASH-2 [26], and two microbenchmarks of GEMS [27] to evaluate the proposed idea. The source code of each application in STAMP and SPLASH-2 has been modified to be executed on the baseline TM system. The *bayes* benchmark from STAMP has not been used since it has non-deterministic termination conditions and the execution time is not deterministic [23]. *Cholesky* and *raytrace* are chosen from SPLASH-2 and *btree* and *deque* are chosen from microbenchmarks; they are the most frequently used applications in prior TM research [4], [6], [22], [44], [45].

Table 5 shows the transactional characteristics of the STAMP applications and input parameters which have been used in the simulation. We have used different parameters in profiling and evaluation to guarantee the validity of the profiling. The information on characteristics of each program is provided by the STAMP project. The third column is transaction length, which is determined based on the number of instructions. The fourth column presents the size of the read and write sets during the transaction. The transaction time that is shown in the fifth column is time spent in transactions. The last column describes the contention level of each application. More details about these transactional characteristics are well explained by Minh *et al.* [25]. The input parameters of additional benchmark programs are shown in Table 6.

TABLE 6
SPLASH-2 and Microbenchmarks Setting

Application	Input	Contention
cholesky	tk14	Medium
raytrace	teapot	Medium
btree	15 32768 32768 32768	High
deque	15 1500 1024	High

In fact, we consider the implications of these characteristics when analyzing the effect of the dynamic backoff control algorithm. For instance, *yada* has relatively higher transaction time than *kmeans*. Therefore, reducing the number of aborts may be more effective in *yada* than in *kmeans*.

5 Results and Analysis

In this section, we present our performance evaluation with the simulation results to demonstrate the advantage of using the proposed dynamic backoff algorithm. Every benchmark program was simulated using 10 different versions of the operating system image to show the effect of different seed values for the random number generator. In addition, we give a thorough analysis of the results with consideration given to the various application characteristics.

5.1 Effect of Non-Zero Initial Value

In Fig. 6 we present four criteria to evaluate performance of each application. Each result is the average of the 10 runs explained in the above paragraph.

In the figure, the relative number of aborts represents the total number of aborts per transaction that is normalized to

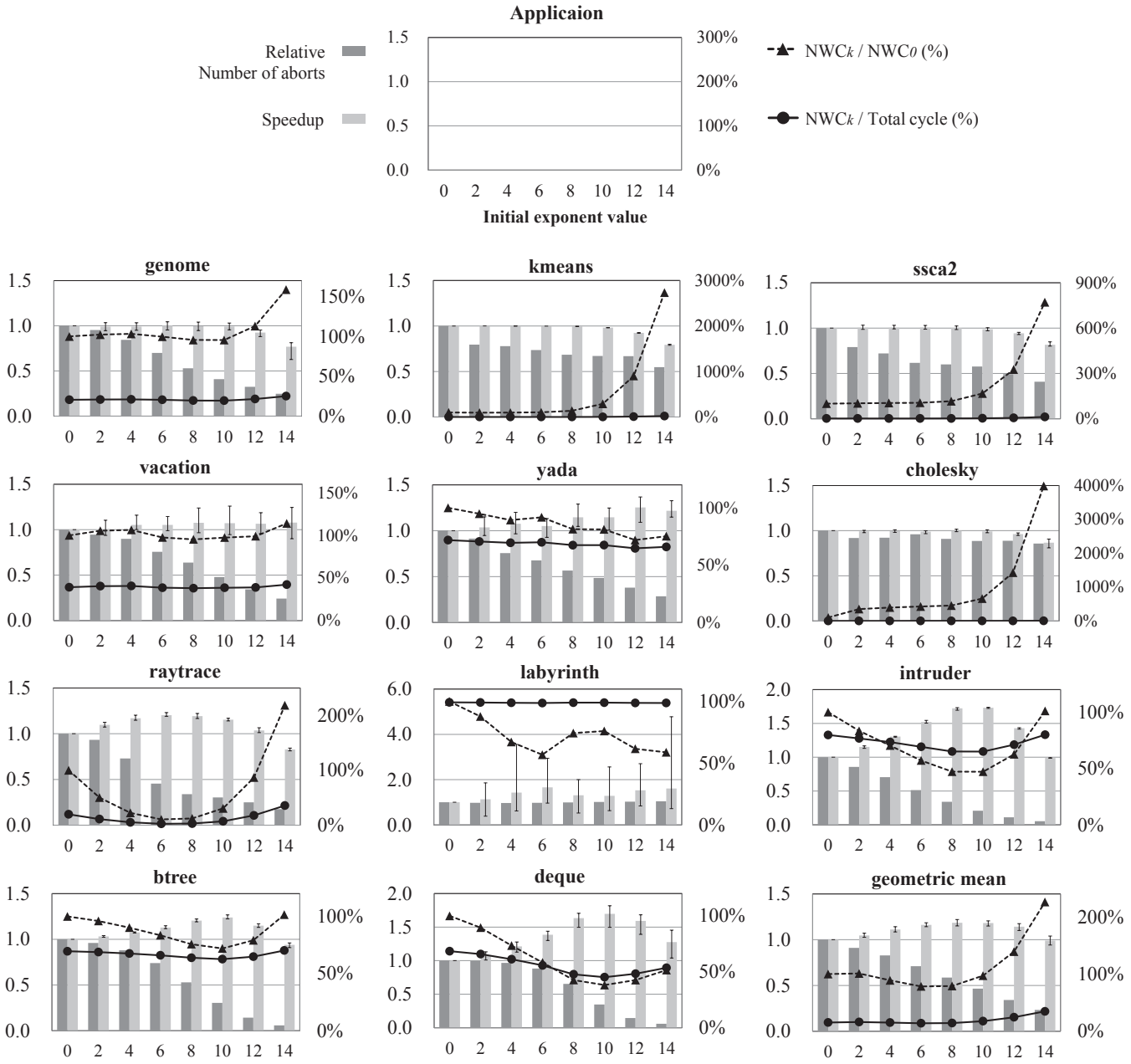


Fig. 6. Benchmark results with various initial exponent values

the number in the baseline system. The speedup shows the performance improvement of each program with the various initial exponent values. In addition, the dotted lines represent the normalized NWC over NWC_0; NWC_0 represents the NWC value when the original exponential backoff algorithm is used (in other words, when k is equal to zero). Lastly, the solid lines indicate the percentage of NWC over the total number of cycles.

Considering the results in Fig. 6, two distinct observations can be made. First, applying various initial values helps reduce NWC. However, as we have stated in Section 2.1, having an excessively large value of k causes an increase in NWC. As shown in the figure, the average amount of NWC becomes 2.3 (geometric mean) times larger when k = 14.

Second, the relative number of aborts is decreased as the value of k is increased. This is due to the increased delay in the backoff algorithm: transactions tend to wait for the restart rather than being conflicted and aborted. In other words, the abort penalty can be minimized by increasing the value of k. However, as pointed out earlier, an unlimited increase of k cannot guarantee performance improvement due to the negative effect of a large value of k. For this reason, we have set the threshold to 12 considering the performance results shown in Fig. 6.

5.2 Performance Analysis with Varying Initial Values

In the simulation results shown in Fig. 6, every benchmark program shows a reduction in NWC and the number of aborts. Among them, *yada*, *intruder*, and *deque* have achieved

remarkable performance improvements with varying initial exponent values.

When we simulated the *intruder* and *deque* applications with our proposed idea using 10 for k, NWC reduced by more than half compared to the baseline system. Also, remarkable speedups of 1.7 have been achieved in both applications. Especially, the observed variation in *intruder* is quite small. In the case of *yada*, a speedup of 1.3 has been achieved when k = 12 and NWC has been decreased by one-third over the baseline system.

As shown in Fig. 6, about 70% of the total number of execution cycles are NWC in the above three applications when k is 0, which means the baseline system. At this point, a non-zero initial value decreases NWC by reducing the number of aborts. The amount of NWC is affected by the characteristics that are shown in Table 5. The *yada* application has long transactions, large read and write sets, and high transaction time that causes large abort penalties. In the *intruder* application, in spite of the short length of transactions, frequent re-balancing operations on the tree data structure cause high levels of contention [25]. *Deque* has characteristics similar to *intruder*. In the program, each thread performs insert and delete operations on the shared double-ended queue which raises the contention. This consequently results in a large amount of NWC compared to low-contention applications.

The performance of *raytrace* and *btree* also has been improved and the variation is quite small using a non-zero initial exponent value. These results imply that the proposed mechanism is effective for diverse applications. In addition, the results show very similar trends across all the programs we tested. That is, the performance improves as k_0 increases; however excessively large values of k_0 could cause increased NWC and performance degradation.

In *labyrinth*, the geometric mean over 10 simulations shows that the proposed method improves performance. However, the variation is quite large and performance degradation would be observed in some cases. For example, an average speedup of 1.3 is achieved when k = 8 whereas a 0.5 speedup occurs in one case among the 10 simulations. This is due to a property of the program; *labyrinth* uses a variant of Lee's path routing algorithm [46] which requires long transactions with a large number of read and write operations. Consequently, contention is extremely high. Thus a more elaborate control of the backoff is needed.

The method has not been effective at improving performance in the case of *genome*, *kmeans*, and *ssca2* since they have low contention as shown in Table 5. Although *genome* and *kmeans* have shown reduced NWC values, the performance improvement is negligible due to the small ratio of NWC to the total cycles. In other words, decreasing NWC is not necessarily an important factor for those applications. Consequently, there is no significant improvement in low-contention applications. However, we also have found that there is no serious performance degradation in those applications if k_0 is less than 12. For example, the measured speedup is 0.97 although NWC increased more than 14 times compared to the baseline system in *cholesky* with k_0 = 12; this is due to the extremely low NWC ratio to the total cycles. In conclusion, having a non-zero initial value can contribute to the performance of high-contention applications while giving a marginal benefit or no harm in low-contention applications.

This observation gives another insight for contention management including transaction scheduling: considering that chances to improve performance in low-contention applications are quite low, the overhead spent in scheduling transactions should be minimized for system-wide tradeoffs. The main purpose of the approach is to prevent conflicts and to reduce the number of wasteful operations represented by NWC. However, the NWC in the low-contention applications is already small, and therefore the potential performance improvement by minimizing NWC in low-contention applications is quite limited. For example, the percentage of NWC over the total number of cycles is less than 1% in *kmeans*.

As shown in the speedup results in Fig. 6, the best performance for each application has been achieved with a respective k value. For example, the best selection of k for *intruder* is 10 while the value for *raytrace* is 6. In addition, if we use a fixed value of k as 8, *labyrinth* may experience severe performance degradation whereas *intruder* and *yada* benefit from the backoff algorithm. This implies that using a static k value cannot provide an effective backoff algorithm for all applications. Therefore, we believe that the proposed dynamic control can be a better solution with a more flexible utilization of the backoff algorithm.

5.3 Effectiveness of Dynamic Control

To demonstrate the performance of the dynamic control mechanism, we have performed extensive simulations with various RAT values and k_0. The geometric means for speedup using the dynamic control method are presented in Fig. 7 and the variations over 10 different systems are shown in Fig. 8. The positive value represents the difference between the maximum

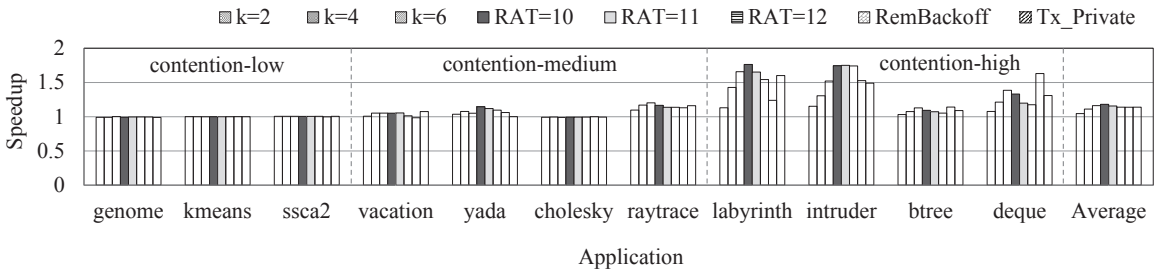


Fig. 7. Speedup results without dynamic control and with dynamic control

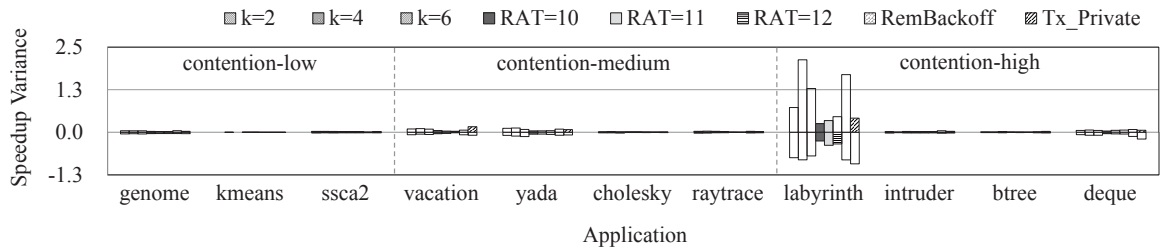


Fig. 8. Speedup variance in various systems

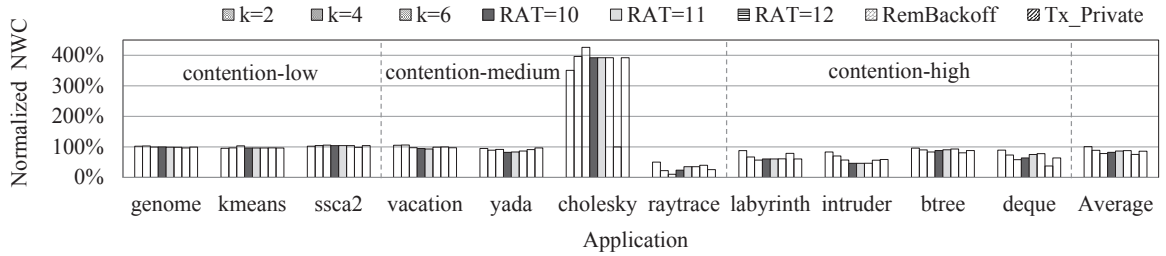


Fig. 9. Normalized NWC in various systems

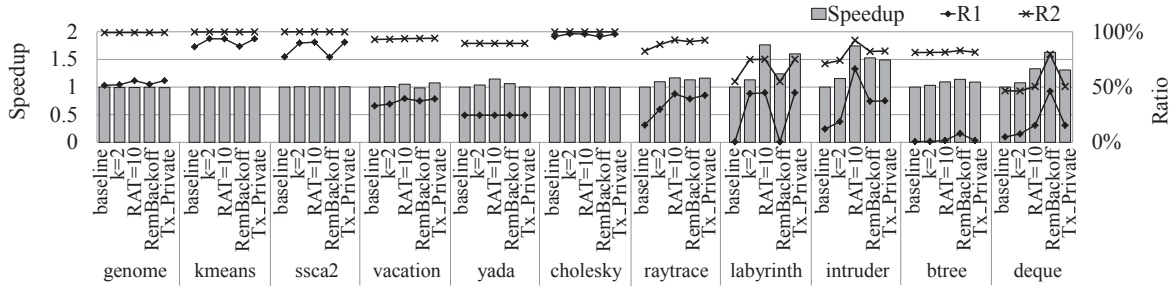


Fig. 10. Speedup results with the ratio of one abort, commits without abort or with one abort

and the mean whereas the negative value shows the difference between the minimum and the mean. In the figure, results with the static approach with three different k values are shown as k_0 = 2, k_0 = 4, and k_0 = 6. Performance results with the dynamic approach are also shown with RAT = 10, RAT = 11, and RAT = 12. In addition, there are two additional configurations. First, RemBackoff represents one of the previously proposed algorithms, *RememberingBackoff* [31]. The last one, Tx_Private shows the performance results with the private approach introduced in Section 3.4. Since the proposed method aims at reducing complexity by employing a backoff based contention manager, PTS and BFGTS would still perform better in terms of execution time; however, both approaches have relative higher hardware/software complexity as described in Section 2.2.

As shown in Fig. 7 and Fig. 8, relatively stable performance has been achieved in every application using the proposed dynamic control method; there is no critical performance degradation in any application. This is particularly the case in *labyrinth*: this is the problem where severe performance fluctuations have been clearly reduced using our proposed method. Also, it should be noted that the *intruder* application has achieved the best performance with the dynamic control method. In fact, values around 11 were selected for the possible RAT. This is based on the profile given in Table 2. The validity of the heuristic approach is verified by the speedup results that are loosely coupled with the value of RAT or k_0. In fact, the dynamic control approach automatically adjusts and finds the optimal initial exponent value according to application behavior at run time. It has also been found that each application results in a different initial exponent value when it completes.

On average, nearly 20% improvement in performance has been achieved throughout the eleven applications (when RAT = 10 and k_0 = 3). Excluding low-contention applications, the performance improvement reaches nearly 30% over eight applications. We have also measured the NWC value for each configuration in Fig. 9. With proper control on the initial exponent value, an excessive increase of NWC has been prevented with the dynamic control method.

To demonstrate how the reduction of repeated aborts affects the overall performance, Fig. 10 shows two important ratios. R1 represents the ratio of the number of commits with only one abort over the number of commits with one or more aborts. R2 stands for the ratio of the number of commits without abort or with one abort over the total number of commits. In fact, the ratio of commits with one abort (R1) has been increased due to the reduction of repeated aborts. This trend is clearly shown in the high-contention and medium-contention applications.

Interestingly, R1 also increases in the low-contention appli-

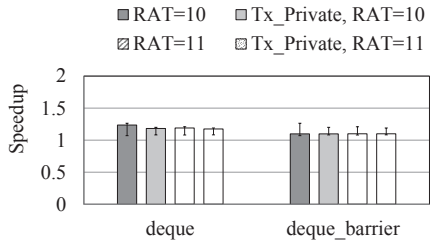


Fig. 11. Effect of synchronization barrier in the proposed method

cations. To clarify this, we also investigated the effect of R2; as seen in the graph, the number of commits without abort or with one abort is almost 100% in the low-contention applications even in the baseline model. This clearly demonstrates that those applications contain low-contention transactions and repeated aborts barely happen. Therefore, eliminating of repeated aborts in these applications is not a critical factor in performance and consequently barely affects the overall performance.

The private method which has been presented in Section 3.4 shows an improved performance compared to the baseline system. However, as we can see in *yada* and *intruder* in Fig. 7 and Fig. 8, the best improvements in performance and its variance factors have been achieved with the global approach. That is due to the reasons explained in Section 3.4. As we have explained, regionally different exponent values cannot achieve efficient contention management. In addition, a sufficiently large amount of backoff cannot be obtained fast because each transactional region privately handles its own exponent value.

Furthermore, we have evaluated the effect of a synchronization barrier with the private method. The programs had been newly synthesized to execute high-contention and low-contention transactional regions. These are noted as *deque* and *deque_barrier* in Fig. 11. *Deque_barrier* has a synchronization barrier between two different transactional regions. As shown in the figure, the amount of performance improvement in *deque_barrier* is not different between the global and private approaches as we have described in Section 3.4. In addition, it has been noted that a lower speedup is achieved in *deque_barrier* since the barrier weakens the effect of the backoff algorithm; distribution of the moment of transaction starting is less dispersed than in cases which do not use barriers.

RememberingBackoff shows stable performance without performance degradation in low-contention applications. However, there is relatively little performance improvement with severe performance fluctuation in the *labyrinth* program. Also, the performance improvement in *intruder* and *yada* is smaller than that of our proposed dynamic backoff control. The *RememberingBackoff* algorithm cannot provide flexible contention management for each program to reduce contention.

6 Conclusion and Future Work

Contention management is an important issue in TM systems to improve performance under high conflict rates caused by the increasing number of cores and large transactions. A backoff-based method for contention management has been proposed. With this approach, repeated aborts can be avoided and the abort penalty can be reduced. The exponential backoff algorithm is the most widely used to determine the amount of delay. In this paper, we pointed out limitations of the algorithms which were presented earlier and proposed our dynamic backoff control algorithm. The purpose of the idea is to reduce the abort penalty by eliminating repeated aborts with a complexity effective approach.

The algorithm introduced the use of a non-zero initial exponent value and proposed to dynamically control the initial exponent value at run time. The idea was implemented on LogTM-SE and various programs including the STAMP benchmark suite were used for evaluation. We proved that our proposed idea was effective with applications which have high contention, and produces marginal benefits or no harm with low-contention applications. Finally, it can be observed that, on average, 18% performance improvement could be achieved compared to the baseline system.

Our main motivation is to provide a complexity-effective contention manager employing a backoff-based approach. We demonstrated that the execution time of the proposed design was lower compared to traditional backoff algorithms. In addition, the previously proposed scheduling approaches, which aggressively use hardware and software resources, would introduce additional implementation issues and power consumption problems.

As future research, we will explore the detailed hardware design for implementing the dynamic backoff control algorithm for various HTM systems. We will further analyze the hardware and software overhead and the possible power consumption of the proposed design considering implementation issues. In addition, we will investigate the trade-offs of various contention management methods based on the overhead analysis.

Acknowledgments

We would like to thank the reviewers whose efforts have tremendously helped us improve this paper. We would also like to thank Dr. Woojin Choi for providing STAMP binary for GEMS. This work is partly supported by the National Science Foundation (Grant No. CCF-1065147) and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0013202). Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of the National Research Foundation of Korea.

References

[1] M. Herlihy and J. E. B. Moss, "Transactional Memory: Architectural Support For Lock-free Data Structures," in *Proc. 20th Int'l Symp. on Computer Architecture (ISCA '93)*, May 1993, pp. 289–300.

[2] L. Ceze, J. Tuck, C. Cascaval, and J. Torrellas, "Bulk Disambiguation of Speculative Threads in Multiprocessors," in *Proc. 33rd Int'l Symp. on Computer Architecture (ISCA '06)*, 2006, pp. 227–238.

[3] L. Hammond, V. Wong, M. Chen, B. Carlstrom, J. Davis, B. Hertzberg, M. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, "Transactional Memory Coherence and Consistency," in *Proc. 31st Int'l Symp. on Computer Architecture (ISCA '04)*, June 2004, pp. 102–113.

[4] K. Moore, J. Bobba, M. Moravan, M. Hill, and D. Wood, "LogTM: Log-Based Transactional Memory," in *Proc. 12th Int'l Symp. on High-Performance Computer Architecture (HPCA '06)*, Feb. 2006, pp. 254–265.

[5] C. Ananian, K. Asanovic, B. Kuszmaul, C. Leiserson, and S. Lie, "Unbounded transactional memory," in *Proc. 11th Int'l Symp. on High-Performance Computer Architecture (HPCA '05)*, Feb. 2005, pp. 316–327.

[6] L. Yen, J. Bobba, M. Marty, K. Moore, H. Volos, M. Hill, M. Swift, and D. Wood, "LogTM-SE: Decoupling Hardware Transactional Memory from Caches," in *Proc. 13th Int'l Symp. on High-Performance Computer Architecture (HPCA '07)*, Feb. 2007, pp. 261–272.

[7] M. Herlihy, V. Luchangco, M. Moir, and W. N. Scherer III, "Software Transactional Memory for Dynamic-Sized Data Structures," in *Proc. 22nd Symp. on Principles of Distributed Computing (PODC '03)*, 2003, pp. 92–101.

[8] W. N. V. J. Marathe, Scherer III and M. L. Scott, "Adaptive Software Transactional Memory," *Technical Report 868, Computer Science Department, University of Rochester*, Feb. 2005.

[9] M. Olszewski, J. Cutler, and J. Steffan, "JudoSTM: A Dynamic Binary-Rewriting Approach to Software Transactional Memory," in *Proc. 16th Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT '07)*, Sept. 2007, pp. 365–375.

[10] B. Saha, A.-R. Adl-Tabatabai, R. L. Hudson, C. C. Minh, and B. Hertzberg, "McRT-STM: A High Performance Software Transactional Memory System for a Multi-Core Runtime," in *Proc. 11th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP '06)*, 2006, pp. 187–197.

[11] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum, "Hybrid Transactional Memory," in *Proc. 12th Int'l Conf. on Architectural Support for Programming Languages and Operating systems (ASPLOS '06)*, 2006, pp. 336–346.

[12] S. Kumar, M. Chu, C. J. Hughes, P. Kundu, and A. Nguyen, "Hybrid Transactional Memory," in *Proc. 11th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP '06)*, 2006, pp. 209–220.

[13] A. Shriraman, M. F. Spear, H. Hossain, V. J. Marathe, S. Dwarkadas, and M. L. Scott, "An Integrated Hardware-Software Approach to Flexible Transactional Memory," in *Proc. 34th Int'l Symp. on Computer Architecture (ISCA '07)*, 2007, pp. 104–115.

[14] R. Titos, M. Acacio, and J. Garcia, "Speculation-Based Conflict Resolution in Hardware Transactional Memory," in *Proc. IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS '09)*, May 2009, pp. 1–12.

[15] M. F. Spear, L. Dalessandro, V. J. Marathe, and M. L. Scott, "A Comprehensive Strategy for Contention Management in Software Transactional Memory," *SIGPLAN Not.*, vol. 44, pp. 141–150, Feb. 2009.

[16] W. Choi and J. Draper, "Locality-aware Adaptive Grain Signatures for Transactional Memories," in *Proc. IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS '10)*, Apr. 2010, pp. 1–10.

[17] S. Tomic, C. Perfumo, C. Kulkarni, A. Armejach, A. Cristal, O. Unsal, T. Harris, and M. Valero, "EazyHTM: EAger-LaZY Hardware Transactional Memory," in *Proc. 42nd IEEE/ACM Int'l Symp. on Microarchitecture (Micro '09)*, Dec. 2009, pp. 145–155.

[18] M. Ansari, C. Kotselidis, K. Jarvis, M. Luján, C. Kirkham, and I. Watson, "Advanced Concurrency Control for Transactional Memory Using Transaction Commit Rate," in *Proc. 14th Int'l Euro-Par Conf. on Parallel Processing (Euro-Par '08)*, 2008, pp. 719–728.

[19] S. Dolev, D. Hendler, and A. Suissa, "CAR-STM: Scheduling-Based Collision Avoidance and Resolution for Software Transactional Memory," in *Proc. 27th Symp. on Principles of Distributed Computing (PODC '08)*, 2008, pp. 125–134.

[20] M. Ansari, M. Luján, C. Kotselidis, K. Jarvis, C. Kirkham, and I. Watson, "Steal-on-Abort: Improving Transactional Memory Performance through Dynamic Transaction Reordering," in *Proc. 4th Int'l Conf. on High Performance Embedded Architectures and Compilers (HiPEAC '09)*, 2009, pp. 4–18.

[21] N. Sonmez, T. Harris, A. Cristal, O. Unsal, and M. Valero, "Taking the Heat off Transactions: Dynamic Selection of Pessimistic Concurrency Control," in *Proc. IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS '09)*, May 2009, pp. 1–10.

[22] R. M. Yoo and H.-H. S. Lee, "Adaptive Transaction Scheduling for Transactional Memory Systems," in *Proc. 20th Symp. on Parallelism in Algorithms and Architectures (SPAA '08)*, 2008, pp. 169–178.

[23] G. Blake, R. Dreslinski, and T. Mudge, "Proactive Transaction Scheduling for Contention Management," in *Proc. 42nd IEEE/ACM Int'l Symp. on Microarchitecture (Micro '09)*, Dec. 2009, pp. 156–167.

[24] G. Blake, R. G. Dreslinski, and T. Mudge, "Bloom Filter Guided Transaction Scheduling," in *Proc. 17th Int'l Symp. on High-Performance Computer Architecture (HPCA '11)*, Feb. 2011, pp. 75–86.

[25] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford Transactional Applications for Multi-Processing," in *Proc. IEEE Int'l Symp. on Workload Characterization (IISWC '08)*, Sept. 2008, pp. 35–46.

[26] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and Methodological Considerations," in *Proc. 22nd Int'l Symp. on Computer Architecture (ISCA '95)*, June 1995, pp. 24–36.

[27] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *SIGARCH Comput. Archit. News*, vol. 33, pp. 92–99, Nov. 2005.

[28] Standard Performance Evaluation Corporation, SPEC OpenMP Benchmark Suite. [Online]. Available: http://www.spec.org/omp

[29] B.-J. Kwak, N.-O. Song, and L. E. Miller, "Performance Analysis of Exponential Backoff," *IEEE/ACM Trans. Netw.*, vol. 13, pp. 343–355, Apr. 2005.

[30] Y.-f. Ling, D.-y. Meng, and F. Gao, "Study on Improved Truncated Binary Exponential Back-off Collision Resolution Algorithm," *Int'l Journal of Computer Science and Network Security*, vol. 6, pp. 97–101, Nov. 2006.

[31] D. Hasenfratz, J. Schneider, and R. Wattenhofer, "Transactional Memory: How to Perform Load Adaption in a Simple and Distributed Manner," in *Proc. Int'l Conf. on High Performance Computing and Simulation (HPCS '10)*, July 2010, pp. 163–170.

[32] C. Ferri, S. Wood, T. Moreshet, R. Iris Bahar, and M. Herlihy, "Embedded-TM: Energy and Complexity-effective Hardware Transactional Memory for Embedded Multicore Systems," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 1042–1052, 2010.

[33] W. N. Scherer III and M. L. Scott, "Contention Management for Dynamic Software Transactional Memory," in *Proc. ACM PODC Workshop on Concurrency and Synchronization in Java Programs*, July 2004.

[34] W. N. Scherer III and M. L. Scott, "Advanced Contention Management for Dynamic Software Transactional Memory," in *Proc. 24th Symp. on Principles of Distributed Computing (PODC '05)*, 2005, pp. 240–248.

[35] A. Dragojević, R. Guerraoui, A. V. Singh, and V. Singh, "Preventing Versus Curing: Avoiding Conflicts in Transactional Memories," in *Proc. 28th Symp. on Principles of Distributed Computing (PODC '09)*, 2009, pp. 7–16.

[36] W. Maldonado, P. Marlier, P. Felber, A. Suissa, D. Hendler, A. Fedorova, J. L. Lawall, and G. Muller, "Scheduling Support for Transactional Memory Contention Management," in *Proc. 15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP '10)*, 2010, pp. 79–90.

[37] C. Cascaval, C. Blundell, M. Michael, H. W. Cain, P. Wu, S. Chiras, and S. Chatterjee, "Software Transactional Memory: Why Is It Only a Research Toy?" *Commun. ACM*, vol. 51, pp. 40–46, Nov. 2008.

[38] C. Zilles and L. Baugh, "Extending Hardware Transactional Memory to Support Non-busy Waiting and Non-transactional Actions," in *Proc. 1st ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing*, 2006, pp. 63–72.

[39] L. Michael, W. Nejdl, O. Papapetrou, and W. Siberski, "Improving Distributed Join Efficiency with Extended Bloom Filter Operations," in *Proc. 21st Int'l Conf on Advanced Information Networking and Applications (AINA '07)*, 2007, pp. 187–194.

[40] M. Lupon, G. Magklis, and A. González, "Version Management Alternatives for Hardware Transactional Memory," in *Proc. 9th Workshop on MEmory performance: DEaling with Applications, Systems and Architecture (MEDEA '08)*, 2008, pp. 69–76.

[41] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *IEEE Tran. Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

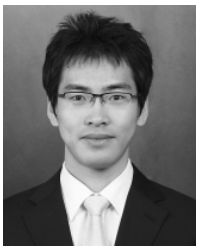
[42] D. Sanchez, L. Yen, M. Hill, and K. Sankaralingam, "Implementing Signatures for Transactional Memory," in *Proc. 40th IEEE/ACM Int'l Symp. on Microarchitecture (Micro '07)*, Dec. 2007, pp. 123–133.

[43] M. Lupon, G. Magklis, and A. González, "FASTM: A Log-based Hardware Transactional Memory with Fast Abort Recovery," in *Proc. 18th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT '09)*, Sept. 2009, pp. 293–302.

[44] J. Bobba, N. Goyal, M. D. Hill, M. M. Swift, and D. A. Wood, "TokenTM: Efficient Execution of Large Transactions with Hardware Transactional Memory," in *Proc. of the 35th Annual International Symp. on Computer Architecture (ISCA '08)*, 2008, pp. 127–138.

[45] J. R. T. Gil, M. E. A. Sanchez, and J. M. G. Carrasco, "Characterization of Conflicts in Log-Based Transactional Memory (LogTM)," in *Proc. of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, ser. PDP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 30–37.

[46] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *Electronic Computers, IRE Tran. on*, vol. EC-10, no. 3, pp. 346–365, Sept. 1961.



Seung Hun Kim received the B.S. and M.S. degrees in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2009 and 2011, respectively. He is currently Ph.D. student in Embedded Systems and Computer Architecture Laboratory, School of Electrical and Electronic Engineering, Yonsei University. His research interests include the Transactional Memory systems and multi-core architecture.

Dongmin Choi received the B.S. degree in biomedical engineering from Yonsei University, Wonju, Korea, in 2004 and the M.S. degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2012. He joined Samsung Electronics Inc. in 2004 and currently works in Global GSM Software Group in Samsung Electronics. His research area is Hardware Transactional Memory systems.

Won Woo Ro received the B.S. degree in electrical engineering from Yonsei University, Seoul, Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California in 1999 and 2004, respectively. He worked as a research scientist in the Electrical Engineering and Computer Science Department, University of California, Irvine. He currently works as an Associate Professor in the School of Electrical and Electronic Engineering at Yonsei University. Prior to joining Yonsei University, he has worked as an Assistant Professor in the Department of Electrical and Computer Engineering at California State University, Northridge. His industry experience also includes a college internship at Apple Computer, Inc., and a contract software engineer in ARM, Inc. His current research interests are high-performance microprocessor design, compiler optimization, and embedded system designs. He is a member of the IEEE.

Jean-Luc Gaudiot received the Diplôme d'Ingénieur from the École Supérieure d'Ingénieurs en Electrotechnique et Electronique, Paris, France in 1976 and the M.S. and Ph.D. degrees in Computer Science from the University of California, Los Angeles in 1977 and 1982, respectively. He is currently a Professor and Chair of the Electrical and Computer Engineering Department at the University of California, Irvine. Prior to joining UCI in January 2002, he was a Professor of Electrical Engineering at the University of Southern California since 1982, where he served as and Director of the Computer Engineering Division for three years. He has also done microprocessor systems design at Teledyne Controls, Santa Monica, California (1979-1980) and research in innovative architectures at the TRW Technology Research Center, El Segundo, California (1980-1982). He consults for a number of companies involved in the design of high-performance computer architectures. His research interests include multithreaded architectures, fault-tolerant multi-processors, and implementation of reconfigurable architectures. He has published over 200 journal and conference papers. His research has been sponsored by NSF, DoE, and DARPA, as well as a number of industrial organizations. In January 2006, he became the first Editor-in-Chief of IEEE Computer Architecture Letters, a new publication of the IEEE Computer Society, which he helped found to the end of facilitating short, fast turnaround of fundamental ideas in the Computer Architecture domain. From 1999 to 2002, he was the Editor-in-Chief of the IEEE Transactions on Computers. In June 2001, he was elected chair of the IEEE Technical Committee on Computer Architecture, and re-elected in June 2003 for a second two-year term.He is a member of the ACM, of the ACM SIGARCH, and of the IEEE. He has also chaired the IFIP Working Group 10.3 (Concurrent Systems). He is one of three founders of PACT, the ACM/IEEE/IFIP Conference on Parallel Architectures and Compilation Techniques, and served as its first Program Chair in 1993, and again in 1995. He has also served as Program Chair of the 1993 Symposium on Parallel and Distributed Processing, HPCA-5 (1999 High Performance Computer Architecture), the 16th Symposium on Computer Architecture and High Performance Computing (Foz do Iguacu, Brazil), the 2004 ACM International Conference on Computing Frontiers, and the 2005 International Parallel and Distributed Processing Symposium. In 1999, he became a Fellow of the IEEE. He was elevated to the rank of AAAS Fellow in 2007.