Analysis and Evaluation of Locks designed for NUMA system

Joohwan Hong¹, Seokyong Jung¹, Kihyun Yun¹, and Minsoo Ryu¹
¹Department of Computer Science and Engineering, Hanyang University, Seoul, Korea
{jhhong, syjung, khyun}@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

Abstract - As computer systems grow to multi-core and multi-node systems for higher performance, Non-Uniform Memory Access (NUMA) architecture is widely adopted in recent computer systems. In NUMA architectures, typical lock primitives such as spinlock are major performance bottleneck by causing frequent inter-node traffic. For improving performance in NUMA architectures, lock algorithm is cautiously considered to minimize cache coherence traffic and cache coherence misses.

In this paper, we examine two locks designed for NUMA architecture: Cohort Lock and NUMA-Aware Reader-Writer Lock. We analyze the performance with various read-write workload ratios using a microbenchmark. We also evaluate with various settings to improve throughput in two tunable parameters: handoff bound in Cohort Lock and maximum patience limit in NUMA-Aware Reader-Writer Lock. Evaluation shows that throughput is improved up to 3% when we apply appropriate settings in tunable parameters.

Keywords: NUMA, reader-writer lock

1 Introduction

Currently, most high performance systems use hardware platforms with two or more sockets and nodes utilizing several cores per socket and node. In this hardware platform, multi-sockets and multi-nodes system incorporates Non-Uniform Memory Access (NUMA) architecture. NUMA architecture is structured as two or more nodes which are consisted of several cores, a local cache and memory. Local cache and memory of each node can be accessed to other nodes by the medium of interconnect; however, the access time to inter-node is slower than access to intra-node.

As the number of cores grows up in a NUMA system, parallel programming is becoming a key issue for performance since cores operate at the same time. Several applications are designed for parallel programming, but performance is limited by shared resource that can be accessed simultaneously. Especially, typical lock algorithm for shared resource is restricted for performance caused by long access latencies in NUMA system. The reason for long access latencies is caused by inter-node traffic slower than intra-node traffic. Therefore, using intra-node cache, reducing cache coherence traffic, and minimizing cache coherence misses are key factors for improving performance of lock algorithm.

MCS lock [3] is widely known as scalable in shared resource. MCS lock maintains a queue based list to wait for acquiring the lock. The head of the queue-based list is a thread that holds the lock, and the thread which tries to acquire the lock is added to the tail of queue-based list, pointing itself to tail, and spinning on a local variable. When the head thread is about to release the lock, the thread notifies its successor thread in the queue-based list. When the successor thread receives the notification, it stops spinning and acquires the lock, thus the thread becomes the head of the queue-based list. MCS lock avoids bursty accesses to lock when a thread releases the lock by permitting only the successor thread to acquire the lock. However, the performance dropped significantly in NUMA system due to the inter-node traffic when updating and monitoring the tail of the list.

The hierarchical Backoff Lock (HBO) [4] is designed for reducing remote cache access. When a thread waits for accessing shared resource protected by lock, it waits for a backoff time to delay, waits until the backoff time is expired, and tries to acquire the lock for accessing shared resource again. The backoff time can be tuned adaptively, so a thread in a same node with a lock holding thread can be set a backoff time shortly to take a more chance to acquire the lock. Adaptive backoff time can reduce the cache coherence traffic; however, it needs a tunable backoff parameter to get scalable result for every hardware platform, so it cannot give a credible performance and fairness issue [2].

We discussed two locks for the shared resources in multi-threaded applications, but the performance is not scalable well in NUMA system. Therefore, we need to analyze the lock algorithm in NUMA system cautiously for higher performance.

In this paper, we examine the locks designed for the NUMA system, and evaluate the performance in various read-write workload ratios. We also attempt to find out appropriate settings to improve throughput over existing values in tunable parameters. We describe to analyze the scalable NUMA-aware lock in Section 2. We provide an experimental evaluation in Section 3.
2 Locks designed for NUMA system

2.1 Lock Cohorting

Cohort lock [2] is consisted of two levels of locks: global lock and local lock. Global lock is shared by all threads in NUMA system, and local lock is only shared with same node in NUMA system. The thread needs to acquire both locks by a sequence of local lock first and global lock next, and then executes the critical section. When a thread releases the lock, the thread checks a waiting thread in the same node. If there is a waiting thread in the same node, a lock holding thread releases the local lock first, and then passes the global lock to the waiting thread in the same node.

Cohort lock gives a priority to the thread in the same node utilizing the intra-node cache; consequently, inter-node traffic and cache misses can be reduced. However, cohort lock occurs starvation in the thread trying to acquire the lock existed in other nodes. It is because current lock holding thread does not release a global lock to the thread in other nodes. To solve this problem, cohort lock sets a threshold to make a global lock held in a specific node continually. After a thread meets the threshold, it must release a global lock to yield the lock to other nodes. This threshold is called handoff bound and it is set to 64.

2.2 NUMA-Aware Reader-Writer Locks

NUMA-aware reader-writer lock [1] is designed based on cohort lock to minimize lock migrations between writers by releasing global lock to other waiting threads in a same node. Until a writer meets the threshold in the same node, a writer transfers global lock to other waiting writers in the same node. Furthermore, to get a high performance, this lock maximizes the concurrency between readers by batching reader and writer operations. The readers actually do not need to acquire the lock, so they can access the shared resource concurrently when the writers do not execute the critical section protected by the locks. To implement this, it uses read indicators located in each node, arriving when a reader acquires a lock and departing when releases a lock. Writers for acquiring a lock only check the read indicators whether a reader is in the critical section.

NUMA-Aware reader-writer locks present three different preferences: writer-preference, reader-preference, and neutral preference. Writer-preference and reader-preference give a priority to execute critical section regardless of arrival order to critical section. However, this makes starvation to opposite operation to each other preference. To overcome this starvation, this lock algorithm adopts the patience limit blocking preference operation continuously. For example, with a writer preference, a reader waits over patience limit, a reader sets a barrier flag and the waiting writers are blocked by barrier until a reader finish to execute critical section.

Calciu et al. [1] sets patience limit value to 1000.

Figure 1. The graph shows the average throughput of the critical section and non-critical section executed alternately during 10 seconds varying read-writer workload ratios: 2%, 50%, 90% of write operation in critical section.
<table>
<thead>
<tr>
<th>Handoff bound</th>
<th>4 Threads</th>
<th>8 Threads</th>
<th>12 Threads</th>
<th>16 Threads</th>
<th>20 Threads</th>
<th>24 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>1.573E+07</td>
<td>2.196E+07</td>
<td>2.235E+07</td>
<td>2.199E+07</td>
<td>2.091E+07</td>
<td>2.023E+07</td>
</tr>
<tr>
<td>64</td>
<td>1.625E+07</td>
<td>2.187E+07</td>
<td>2.220E+07</td>
<td>2.214E+07</td>
<td>2.141E+07</td>
<td>2.070E+07</td>
</tr>
<tr>
<td>96</td>
<td>1.557E+07</td>
<td>2.201E+07</td>
<td>2.231E+07</td>
<td>2.222E+07</td>
<td>2.161E+07</td>
<td>2.081E+07</td>
</tr>
<tr>
<td>128</td>
<td>1.617E+07</td>
<td>2.203E+07</td>
<td>2.232E+07</td>
<td>2.227E+07</td>
<td>2.167E+07</td>
<td>2.106E+07</td>
</tr>
</tbody>
</table>

(a) C-PTL-TKT with Read 50%, Write 50%

<table>
<thead>
<tr>
<th>Patience Limit</th>
<th>4 Threads</th>
<th>8 Threads</th>
<th>12 Threads</th>
<th>16 Threads</th>
<th>20 Threads</th>
<th>24 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.713E+07</td>
<td>2.129E+07</td>
<td>2.267E+07</td>
<td>2.298E+07</td>
<td>2.303E+07</td>
<td>2.296E+07</td>
</tr>
<tr>
<td>500</td>
<td>1.652E+07</td>
<td>2.116E+07</td>
<td>2.308E+07</td>
<td>2.398E+07</td>
<td>2.450E+07</td>
<td>2.450E+07</td>
</tr>
<tr>
<td>1000</td>
<td>1.624E+07</td>
<td>2.110E+07</td>
<td>2.257E+07</td>
<td>2.344E+07</td>
<td>2.408E+07</td>
<td>2.445E+07</td>
</tr>
<tr>
<td>5000</td>
<td>1.621E+07</td>
<td>2.110E+07</td>
<td>2.255E+07</td>
<td>2.326E+07</td>
<td>2.360E+07</td>
<td>2.374E+07</td>
</tr>
<tr>
<td>10000</td>
<td>1.656E+07</td>
<td>2.112E+07</td>
<td>2.258E+07</td>
<td>2.322E+07</td>
<td>2.368E+07</td>
<td>2.376E+07</td>
</tr>
</tbody>
</table>

(b) C-RW-WP with Read 50%, Write 50%

Table 1. Result of throughput with different values in tunable parameters: (a) handoff bound in C-PTL-TKT, and (b) patience limit in C-RW-WP for mixed (read 50%, write 50%) during 10 seconds.

3 Evaluation

In this section, we evaluated the locks designed for NUMA system, and analyze the performance by varying read-write workload ratios. Also, we experiment with various values in two tunable parameters: handoff bound in Cohort Lock and maximum patience limit in NUMA-Aware Reader-Writer Lock. We implemented all locks in C compiled with GCC 4.6.3 at optimization level -O3. All these experiments were conducted on the machine with two Intel Xeon(R) E5-2620 CPUs, six cores for each NUMA nodes, and each core has two hardware threads, which means a total 24 hardware threads context.

Cohort Lock can be implemented with state-of-the-art locks: ticket locks [3], a partitioned ticket lock [5], a MCS queue lock, and a simple test-and-test-and-set backoff lock [6]. We implemented two types of cohort locks: C-PTL-TKT and C-BO-MCS. C-PTL-TKT is composed of a ticket lock for global lock and a partitioned ticket lock for a local lock. C-BO-MCS is consisted of a simple test-and-test-and-set backoff lock for global lock and a MCS queue lock for local lock. These two cohort locks outperform other combination of cohort locks [2]. NUMA-Aware Reader-Writer lock is based on Cohort lock and represents this lock with writer preference lock as C-RW-WP and reader preference as C-RW-RP. We implement C-RW-WP and C-RW-RP locks based on C-PTL-TKT.

3.1 Microbenchmark

We evaluate the various locks and compare each other with microbenchmark designed by Calciu et al. [1]. The microbenchmark can vary a number of executing threads concurrently, iterations of critical section and non-critical section, total executing seconds, probability for reading or writing. Each critical section of reading and writing executes read or write two variables in shared array with 64 integers. We evaluated 4 times of reading and writing critical section, and 64 times of non-critical section alternately during 10 seconds.

Figure 1 reports the throughput of various locks during 10 seconds in varying read-write workload ratios. The locks designed for NUMA system give high performance than the-state-of-the art locks: mcs, backoff, and posix pthread rwlock. Especially, C-RW-WP outperforms the throughput than other locks in read-dominant case (Figure 1 (a)). In write-dominant case (Figure 1 (c)), C-RW-RP shows higher throughput than other NUMA aware locks, however, the difference of throughput is not noticeable than read-heavy case.

3.2 Tunable parameters to improve the throughput

There are parameters to improve to throughput in Cohort Lock and NUMA-aware Reader-Writer lock. In Cohort Lock, handoff bound is a parameter to prevent a starvation by releasing a global lock to other nodes if a specific node sequentially performed critical section until encountered this bound. We experiment with four different handoff bounds in C-PTL-TKT lock during 10 seconds as a same way to Figure 1(b). Table 1(a) shows the throughput of C-PTL-TKT lock with different handoff bound, and we find out higher throughput when we set handoff bound to 128 than any other values. Calciu et al. [1] describe that patience limit in C-RW-WP is a tunable parameter for improving performance and set default value to 1000. We evaluate C-RW-WP lock by with five different patience limits: 100, 500, 1000, 5000, and 10000. Table 1(b) shows the result of throughput with different patience limit, and throughput is improved with...
patience limit value 500 than other patience limits. Consequently, we find out appropriate settings in tunable parameters: 128 in handoff bound C-PTL-TKT and 500 in patience limit C-RW-WP.

We compare the throughput in C-RW-WP lock default values (64 in handoff bound, 1000 in patience limit) with appropriate settings in tunable parameters that we derived with experiments in Table 1. Figure 2 shows the throughput during 10 seconds as a same way to Figure 1(b). The throughput is improved up to 3% with appropriate settings in tunable parameters than default settings.

![Figure 2](image)

**Figure 2.** The graph shows the average throughput for mixed (read 50%, write 50%) during 10 seconds. C-RW-WP-ori stands for C-RW-WP with default parameters: 64 in handoff bound and 1000 in patience limit. C-RW-WP-tun stands for C-RW-WP with tunable parameters: 128 in handoff bound and 500 in patience limit.

4 Conclusions

We reviewed Cohort Lock and NUMA-Aware Reader-Writer Lock, and we analyzed the performance with various read-write workload ratios. These locks gave higher performance than the-state-of-the-art locks. In NUMA system, reducing inter-node traffic and cache coherence misses allows high performance in shared-resources that can be accessed concurrently. We also evaluated with different values in two tunable parameters in Cohort Lock and NUMA-Aware Reader-Writer Lock, and we found out the appropriate settings to improve throughput up to 3% than default values.

In the future, we will review the locks designed for deep NUMA hierarchy more than two-level cohort lock as a NUMA system grows more complex with deep levels. In addition, we will examine delegation lock that provides high throughput by allowing lock holding thread to execute other threads’ critical section. This lock allows shared resource to be fetched in private cache and reduces data transfer between inter-node caches.

5 Acknowledgement

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2017-2012-0-00628) supervised by the IITP(Institute for Information & communications Technology Promotion) and partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (R0114-16-0046, Software Black Box for Highly Dependable Computing) and partly supported by the MISP(Ministry of Science, ICT & Future Planning), Korea, under the National Program for Excellence in SW(R71161610270001002) supervised by the IITP(Institute for Information & communications Technology Promotion).

6 References


