Performance Prediction for Coarse-Grained Locking:
MCS Case

Vitaly Aksenov
ITMO University

Daniil Bolotov
ITMO University

Petr Kuznetsov
Télécom Paris

1 Introduction
Analytical prediction of the performance of a concurrent
data structure is a challenging task. Earlier works proposed
prediction frameworks for lock-free data structures [2–4] and
lock-based ones [1] in which the coarse-grained critical
section is protected with CLH lock [5]. In this paper, we make
the next step and describe an analytical prediction frame-
work for concurrent data structure based on MCS lock [8].
The MCS framework turned out to be more elaborated than
the CLH one.

2 Model
Consider a concurrent system with $N$ processes that obey
the following simple uniform scheduler: in every time tick,
each process performs a step of computation. This scheduler,
resembling the well-known PRAM model [7], appears to be
a reasonable approximation of real-life concurrent systems.
Suppose that the processes share a data structure exporting
a single abstract operation(). Assume that one operation
induces work of size $P$ and incurs no synchronization, and
one process performs work $\alpha$ in a time unit. Then the re-
sulting throughput is $N \cdot \alpha / P$ operations per time unit. One
way to evaluate the constant $\alpha$ is to experimentally count
the total number $F$ of operations, each of work $P$, completed
by $N$ processes in time $T$. Then we get $\alpha = PF / (TN)$. The
longer is $T$, the more accurate is the estimation of $\alpha$.

Consider a simple lock-based concurrent data structure,
where operation() is implemented using the pseudocode
in Figure 1.

The abstract machine used in our analytical throughput
prediction obey the following conditions.

First, we assume that the coherence of caches is main-
tained by a variant of MESI protocol [9]. Each cache line
can be in one of four states: Modified (M), Exclusive (E),
Shared (S) and Invalid (I). MESI regulates transitions be-
tween states of a cache line depending on the request (read
or write) to the cache line by a process or on the request
to the memory bus. The important transitions for us are:
(1) upon reading, the state of the cache line changes from
any state to $S$, and, if the state was $I$, then a read request
is sent to the bus; (2) upon writing, the state of the cache
line becomes $M$, and, if the state was $S$ or $I$, an invalidation
request is sent to the bus.

| 1 | operation() |
| 2 | lock.lock() |
| 3 | for i in 1..C: |
| 4 | nop |
| 5 | lock.unlock() |
| 6 | for i in 1..P: |
| 7 | nop |

Figure 1. The coarse-grained operation

We assume that the caches are symmetric: for each MESI
state, there exist two constants $R_i$ and $W_i$ such that any
read from any cache line with status $st$ takes $R_i$ work and
any write to a cache line with status $st$ takes $W_i$ work. David
et al. [6] showed that for an Intel Xeon machine (similar
to the one we use in our experimental validation below),
given the relative location of a cache line with respect to the
process (whether they are located on the same socket or not),
the following hypotheses hold: (1) writes induce the same
work, regardless of the state of the cache line; (2) swaps, not
county with other swaps, induce the same work as writes;
(3) reads from the invalid cache line costs more than a write.
Therefore, we assume that (1) $W = W_M = W_S = W_I$;
(2) any contention-free swap induces a work of size $W$; and
(3) $R_i \geq W$.

3 MCS lock
As the first step, we replace lock.lock() and lock.unlock() in
Figure 1 with MCS implementation: the resulting code is
presented in Figure 4.

We can then calculate the cost of each line: Lines 9 and 10
cost $W$ each; Line 11 costs $W$ in the uncontended case and
$X$ in the contended one; if the lock is taken, i.e., a condition
in Line 12 returns true, Line 13 costs $W$ and Line 14 costs $R_f$
the critical section costs $C$; if the lock is free, then the read
in Line 18 costs $R_M$, otherwise, it costs $R_f$; Line 19 costs $W$
in the uncontended case and $X$ in the contended one; Line 24
costs $W$; and, finally, the parallel section costs $P$.

In our theoretical analysis, we distinguish two cases: (1) when
the lock is always free, i.e., swap in Line 11 returns null, the
conditional statement in Line 12 does not succeed, and the
conditional statement in Line 18 and CAS in Line 19 always
succeeds; and (2) when the lock is always taken, i.e., the con-
ditional statement in Line 12 succeeds and the conditional
statement in Line 18 does not succeed.

We consider the schedules that correspond to these two
cases: the first case is shown in Figure 2a and the second is
shown in Figure 2b. The complete analysis is presented in
Appendix A.
Figure 2. Examples of executions of the coarse-grained algorithm. Blue intervals depict critical sections and red intervals depict parallel sections.

Figure 3. Throughput on 15 processes for $C \in \{500, 1000, 10000\}$

Now the throughput can be expressed as: $\frac{a}{C + 2R_f + W}$ if $P + 4 \cdot W \leq (N - 1) \cdot (C + 2R_f + W)$, and $\frac{a \cdot N}{P \cdot C + R_M + 4W}$ if $P + 2 \cdot W \geq (N - 1) \cdot (C + 2R_f + W) + R_f - R_M$.

To describe what is happening in between these two extremes, we can use linear approximation.

4 Experiments

The code of our benchmark is written in C++. We use an Intel Xeon Gold 6230 machine with 16 cores. Based on a single test run, we evaluate the constants as follows: $\alpha \approx 4.04 \cdot 10^5$, $W \approx 15$, $R_f \approx 30$, and $R_M \approx 15$. In Figure 3, we depict our measurement results (in blue) and our analytical predictions (in red) for three values of $C \in \{500, 1000, 10000\}$ on 15 processes. Here x axis specifies the size of the parallel section and at the y axis— the throughput. As one can see, our prediction matches the experimental results almost perfectly. The results of our experiments on an AMD machine are given in Appendix C.

5 Conclusion

In this short paper, we showed that in the model presented in [1] one can accurately predict the throughput of lock-based data structures not only when the lock is CLH, but also when the lock is MCS. Further, we want to close the gap by predicting the performance of data structures with the rest lock-types, such as spin locks, etc. Finally, we want to check whether in this model we can predict the throughput of lock-free data structures, such as Treiber’s stack.
A Analysis

A.1 The first case. Lock is unlocked.

We start with the first case. The corresponding schedule is presented on Figure 2a. The first process performs two writes each with cost $W$ in Lines 9 and 10, a contended swap with cost $X$ while other processes operate on cost $W$, a contended swap with cost $X$, and a write with cost $W$ in Line 13. Then, the first process performs: 1) the critical section with cost $C$; 2) a read from an invalid cache line with cost $R_l$ in Line 18 (on the first try, there is the second process waiting for the lock); 3) unlock the next process with cost $W$ in Line 24; 4) the parallel section of cost $P$; 5) two writes of cost $W$ in Lines 10 and 9; 6) a swap with null1 of cost $W$ in Line 11 since it is already uncontended; 7) the critical section with cost $C$; 8) a read from the exclusive cache line with cost $R_M$ in Line 18 since it has not been changed from Line 9; 9) a CAS of cost $W$ in Line 19; 10) the parallel section of cost $P$; 11) continue from the point 5) again.

The other processes do the similar thing. All other processes perform: 1) the invalid read of cost $R_l$ at Line 14 since there was the previous process waiting for the lock; 2) the critical section with cost $C$; 3) an invalid read of cost $R_l$ at Line 18 (please, note that process $n$ spends there only $R_M$, since for him the queue is empty); 4) unlock with cost $W$ in

B Experiments. Intel.

Results of more experiments on different critical section sizes from $\{500, 1000, 5000, 10000, 50000\}$ and the number of processes from $\{5, 10, 15\}$ can be seen on Figure 5.

C Experiments. AMD.

We use a machine with one AMD Opteron 6378 with 16 cores. We estimate the constants once during the one chosen run as $\alpha \approx 1.24 \cdot 10^3$, $W \approx 20$, $R_l \approx 35$, and $R_M \approx 15$. The plot 6 contains the real-life (blue) line and our expectation (red) line for five different values of critical section $C \in \{500, 1000, 5000, 10000, 50000\}$ on $\{5, 10, 15\}$ processes with the size of the parallel section at OX and throughput at OY axis. As one can see our prediction almost matches the experiment.
**Figure 5.** Throughput on \{5, 10, 15\} processes for \(C \in \{500, 1000, 5000, 10000, 50000\}\). Intel.
Figure 6. Throughput on \{5, 10, 15\} processes for \(C \in \{500, 1000, 5000, 10000, 50000\}\). AMD.