A Probabilistic Quantitative Analysis of Probabilistic-Write/Copy-Select*

Christel Baier, Benjamin Engel, Sascha Klüppelholz, Steffen Märcker, Hendrik Tews, and Marcus Völp

Institute for Theoretical Computer Science and Institute for Systems Architecture
Technische Universität Dresden, Germany
{baier,klueppel,maercker}@tcs.inf.tu-dresden.de,
{engel,tews,voelp}@os.inf.tu-dresden.de

Abstract. Probabilistic-Write/Copy-Select (PWCS) is a novel synchronization scheme suggested by Nicholas Mc Guire which avoids expensive atomic operations for synchronizing access to shared objects. Instead, PWCS makes inconsistencies detectable and recoverable. It builds on the assumption that, for typical workloads, the probability for data races is very small. Mc Guire describes PWCS for multiple readers but only one writer of a shared data structure. In this paper, we report on the formal analysis of the PWCS protocol using a continuous-time Markov chain model and probabilistic model checking techniques. Besides the original PWCS protocol, we also considered a variant with multiple writers. The results were obtained by the model checker PRISM and served to identify scenarios in which the use of the PWCS protocol is justified by guarantees on the probability of data races. Moreover, the analysis showed several other quantitative properties of the PWCS protocol.

1 Introduction

Control mechanisms for shared data is a central task for the design of parallel systems. Various protocols to ensure exclusive access to shared data have been developed by the operating system community. Most prominent are sophisticated locking schemes. These, however, became more and more complex. Moreover, scalability turns out to be problematic because atomic operations and cache synchronization between an ever growing number of cores became more and more expensive [MCS91].

At a recent Real-Time Linux Workshop, Nicholas Mc Guire proposed a promising idea to exploit the increasing complexity of modern manycore systems for synchronizing shared objects [Gui11]. Rather than avoiding inconsistencies at all cost by locking objects or updating their state with increasingly

* This work was in part funded through the DFG project QuaOS, the CRC 912 Highly-Adaptive Energy-Efficient Computing (HAEC), the EU under FP-7 grant 295261 (MEALS), the DFG/NWO-project ROCKS, the cluster of excellence cfAED (center for Advancing Electronics Dresden) and by the EU and the state Saxony through the ESF young researcher group IMData 100098198.

more expensive atomic operations, Mc Guire proposed to not synchronize readers and writers at all. Instead, he proposed to explicitly allow data races on the shared object but to make inconsistencies from ongoing writes detectable. As such, Mc Guire's protocol is an instance of a new class of algorithms that make use of the randomness that is inherent in complex modern computer architectures. This randomness is caused by differences in the content of the core-local caches and by arbitration at the hardware level, which together induce small, but almost unpredictable differences in the execution time. In the approach of Mc Guire, writers are viewed as fault injectors that access the shared data items probabilistically. Instead of single data items that are protected by some locking scheme, Mc Guire's approach deals with a fixed number of replicas of the shared objects (called "copies" in [Gui11]) that are written and read in reversed order. The idea is that, thanks to the inherent randomness of the writers, the probability for a reader to find at least one consistent replica is sufficiently high. For this reason, Mc Guire used the notion *Probabilistic-Write/Copy-Select*, PWCS for short, for his approach. Mc Guire reports in [Gui11] on measure-based experiments illustrating that PWCS is indeed a promising approach that outperforms most locking schemes in high-end cache coherent systems. Moreover, PWCS makes no special assumptions on the memory consistency model except that modifications will eventually propagate to prospective readers.

In this paper, we report on a formal analysis of Mc Guire's protocol using probabilistic model checking techniques. We designed a continuous-time Markov chain (CTMC) to model the PWCS protocol, using exponential distributions as a formalization of the inherent randomness of complex systems observed by Mc Guire. While [Gui11] only addresses the case of a single writer and multiple readers, we analyzed the protocol for multiple writers. This requires the consideration that the replica of shared data items can be in three modes: consistent, currently modified (by precisely one writer) or damaged (concurrently modified by two or more writers). We identified a series of quantitative measures that serve to evaluate the adequacy of the PWCS protocol and that address different aspects. From the readers perspective, guarantees on the success rate and the required time to find at least one consistent replica are most relevant. The average repairing time provides a formal criterion for the usefulness of the implicit repairing mechanism of damaged replica, given by the possibility that eventually some writer modifies the damaged replica without being interfered by other writers. These and other quantitative measures have been formalized as quantitative queries using continuous stochastic logic (CSL) [ASSB00,BHHK03] and its extension for reasoning about rewards (CSRL) [BHHK00] and analyzed using the model checker PRISM [KNP04,KNP09]. The model checking results indeed confirm Mc Guire's observations.

At its current stage, it is too early to give affirmative answers to the applicability of PWCS. But both, Mc Guire's measure-based evaluation and our quantitative analysis, give evidence in the potential of PWCS-like protocols that make use of the inherent probabilism in complex systems to avoid the drawbacks of standard locking schemes or other coordination mechanisms relying on

a deterministic protocol or explicit probabilistic algorithms (e.g., using random number generators).

Related work. Probabilistic model checking was already used in various application areas, ranging from distributed randomized algorithms over energy management, communication, gossiping and cryptographic protocols to network on chip design and system biology. See e.g. the homepages of the model checkers PRISM [KNP04,KNP05,KNP09], MRMC [KZH+11] or the CADP tool set [CGH+10]. Most related to our approach are case studies that address the quantitative analysis of non-randomized mutual exclusion protocols where stochastic distributions were used to model the delay or duration of actions. Examples are the case studies performed by Mateescu and Serwe with the CADP toolbox [MS10] and a series of classical mutual exclusion algorithms and our recent work on a simple spinlock protocol using PRISM and MRMC [BDE+12b,BDE+12a]. The timing behavior of standard mutual exclusion protocols using mathematical reasoning with stochastic distributions was also investigated by the algorithm community, see e.g. [GM99].

2 Probabilistic-Write/Copy-Select

In [Gui11], Mc Guire presented two alternative implementations of PWCS [Gui11] projected objects, which differ in the type of token they use for making object inconsistencies detectable: tag-based consistency tokens complement the object with a pair of version numbers that have to match for the data to be read consistent; hash-based consistency tokens on the other hand store a hash of the data, which up to collision uniquely identifies the consistent states of the object.

Writes proceed by first marking the object inconsistent, either by incrementing the version of the end tag or, in case of hashes, by simply modifying some part of the object to cause a mismatch between the stored hash and data. After the modification completes, the begin tag is incremented to mark the data consistent again. In case of hashes, consistency is established automatically once the stored hash and all modifications become visible.

Readers match this operation by taking a copy of the object and its consistency token. More precisely, they first copy the begin tag and then the data into a private buffer. After that, they match the buffered begin tag with the end tag stored in the object to determine whether they have read a consistent version. If not they retry or follow some other back-off strategy. Replicas of the object are used to further reduce the chance of a reader finding an inconsistent object.

To enforce the order in which end tag, data and begin tag become visible, fences have to be used on modern processor architectures possibly in combination with some delay loop or packet ordering scheme to ensure that the respective updates become visible in the desired order. Hashes further relax these hardware dependencies because no matter in which order the data and tags are read or written, once all data plus the corresponding hash arrives at a reader, it will find its copy to be consistent.

Mc Guire does permits only one writer process. Here, we extend PWCS and consider multiple writers that may modify all replicas of the object concurrently. It may therefore happen that some replicas get damaged because different writers succeed in storing parts of their data. In this case, the stored data and the hash do not match, which allows a reader to recognize an inconsistency. Note that tags would not allow for a reliable damage detection. In contrast to our model, a reader cannot distinguish between a damaged replica and a temporarily inconsistent one that is currently updated by a single writer. We assume that a damaged replica becomes consistent again, and is thus implicitly repaired, when a single writer can modify it without others interfering.

3 Stochastic model of PWCS

To model the PWCS protocol we use exponential distributions for representing the inherent randomness in an explicit way. This leads to a continuous-time Markov chain (CTMC), which then serves as basis for the formal analysis using the CTMC engine of the PRISM model checker. The CTMC is obtained in a compositional way, using CTMCs with action labels for each writer and each reader. The replica are represented by a non-stochastic transition system that synchronize with the writers to change their states.

3.1 Preliminaries

We briefly summarize the relevant principles of continuous-time Markov chains. Further details can be found in textbooks on Markov chains, see e.g. [Kul95,KS60].

If S is a finite set then a distribution on S is a function $\nu : S \to [0,1]$ with $\sum_{s \in S} \nu(s) = 1$. For $U \subseteq S$, $\nu(U)$ is a shortform notation for $\sum_{s \in U} \nu(s)$.

The CTMCs we use here are tuples $\mathcal{M} = (S, Act, R, \mu)$ where S is a finite state space, Act a finite set of action names and R a function of the type $R : S \times Act \times S \to \mathbb{R}_{\geq 0}$, called the rate matrix of $\mathcal{M}$. The last component μ is a distribution on S specifying the probabilities for the starting states. If ν is a distribution on S then we write $\mathcal{M}_\nu$ for the CTMC (S, Act, R, ν) that results from $\mathcal{M}$ by replacing the initial distribution μ with ν.

We write $s \xrightarrow{\lambda:\alpha} s'$ if $R(s, \alpha, s') = \lambda > 0$ with the intuitive meaning that $\mathcal{M}$ has a transition from state s to state s' with action label α and rate λ. The value λ specifies the rate of an exponential distribution. That is, the probability for the transition $s \xrightarrow{\lambda:\alpha} s'$ to be ready for firing some time in the interval $[0,t]$ is $1-e^{-\lambda t}$. Thus, the average delay of this transition is $1/\lambda$. If $R(s, \alpha, s') = 0$ then $\mathcal{M}$ cannot move from s to s' via action α. The choice between several enabled transitions in state s relies on the race condition. Thus, the time-abstract probability to fire a particular transition $s \xrightarrow{\lambda:\alpha} s'$ in state s is $P(s, \alpha, s') = \lambda / E(s)$ where $E(s)$ denotes the exit rate of state s, i.e., the sum of the rates of all outgoing transitions of state s. The probability that $s \xrightarrow{\lambda:\alpha} s'$ will fire within t time units is then $P(s, \alpha, s') \cdot \left(1 - e^{-E(s) \cdot t}\right)$.

Paths in a CTMC are sequences of consecutive transitions augmented by the time points when they are taken. The quantitative analysis using the logics CSL [ASSB00,BHHK03] relies on the standard σ-algebra on infinite paths and probability measure (see e.g. [Kul95,KS60]). To specify measurable sets of infinite paths, we will use LTL-like notations, such as $\Diamond T$ ("eventually T") where $T \subseteq S$ denotes the set of all infinite paths that contain at least one T-state. Similarly, $\mathcal{U}$ denotes the until-operator and $\mathcal{U}^{\leq t}$ the time-bounded until with time bound t. To formalize measurable sets of paths in a state-based logical framework, we will also use state predicates and propositional formulas built upon them as a symbolic formalism for sets of states. The state predicates and their meanings will be obvious from the names of the states in our model.

For our analysis, we are chiefly interested in the long-run behavior, i.e., the system behavior when time tends to infinity and when the system is in equilibrium. For this purpose, we deal with the *steady-state distribution* $\theta : S \to [0,1]$ where $\theta(s)$ represents the average fraction of time to be in state s on the long-run. Formally, $\theta(s)$ is defined by

$$\theta(s) \quad = \quad \lim_{t \to \infty} \theta(s,t)$$

where $\theta(s,t)$ denotes the probability for $\mathcal{M}$ being in state s at time instant t. For finite CTMCs, function θ is well-defined and it is indeed a distribution on S. Long-run probabilities refer to the probability measure obtained for the CTMC $\mathcal{M}_\theta$ where the original initial distribution μ of $\mathcal{M}$ is replaced with the steady-state distribution θ.

Suppose now that U is a set of states such that $\theta(U) > 0$. Conditional long-run probabilities under condition U refer to the long-run behavior of $\mathcal{M}$ when starting in one of the U-states. These are obtained by using the probability measure of the CTMC $\mathcal{M}_\theta^U = \mathcal{M}_\nu$ where ν is the distribution on S given by $\nu(s) = 0$ if $s \in S \setminus U$ and $\nu(s) = \theta(s)/\theta(U)$ if $s \in U$. If Π is a measurable set of infinite paths, then the *conditional long-run probability* for Π under condition U is the probability measure of Π in the CTMC $\mathcal{M}_\theta^U$ and denoted by $\Pr\big(\Pi \,\big|\, U\big)$.

We will also study reward-based properties formalized using the logic CSRL [BHHK00]. These require an extension of $\mathcal{M}$ by a reward function $rew : S \to \mathbb{R}_{\geq 0}$ where $rew(s)$ specifies the reward to be earned per time unit when staying in state s. For finite paths one can then reason about the accumulated reward. Suppose π is a finite path where the underlying state sequence is $s_0\, s_1 \ldots s_n$ and let $t_0 = 0$ and t_i the time point where π takes the i-th transition. The accumulated reward of π is:

$$Rew(\pi) \quad = \quad \sum_{i=0}^{n-1} \big(t_{i+1} - t_i\big) \cdot rew(s_i)$$

Suppose U is a set of states with $\theta(U) > 0$ and $\Pr\big(\,\Diamond T \mid U\big) = 1$. The *conditional long-run accumulated reward* for eventually reaching T under condition U is defined as the expected value of the random variable that assigns to each infinite path in $\Diamond T$ the accumulated reward of the shortest prefix that ends in

a T-state under the probability measure in the CTMC $\mathcal{M}_\theta^U$. It is denoted by $\mathrm{AccRew}\big(\lozenge\, T \,\big|\, U\big)$. In the analysis of the PWCS-protocol, we will deal with the reward function that assigns value 1 to all states. In this case, $\mathrm{AccRew}\big(\lozenge\, T \,\big|\, U\big)$ can be interpreted as the average amount of time to reach T from U on the long-run.

For the quantitative analysis of the PWCS-protocol, we will consider several instances of $\Pr\big(\,\Pi \,\big|\, U\big)$ and $\mathrm{AccRew}\big(\lozenge T \,\big|\, U\big)$, including those where U is given a set of actions rather than a set of states. In those cases, U is identified with the set of states that can be entered via taking some transition with an action label in U.

3.2 Modeling the PWCS-protocol

The CTMC for the PWCS-protocol will be obtained by composing CTMCs for the writers and readers and ordinary (non-stochastic) transition systems for the replica. For the synchronization of the writers with the replica, we use CSP-like notations for actions: $!a$ for the sending of a signal by some writer and $?a$ for the matching receive action by the replica. Since all state changes of the replica are triggered by the writers, the action alphabet of the replica consists of actions of the form $?a$ where the corresponding send action $!a$ is in the action alphabet of some writer. The other actions of the writers and all actions of the readers are executed in an interleaved way. Since only the sending actions are augmented with rates, the rate for the synchronization of $?a$ and $!a$ in the composite CTMC is the rate of the sending action $!a$ in the CTMC of the writer. This corresponds to the following SOS-rules to combine the CTMCs for the writers and the readers with the transition system for the replica to obtain a CTMC for the PWCS-protocol:

$$\frac{s \xrightarrow{\lambda:\alpha} s'}{\langle s, \overline{x}\rangle \xrightarrow{\lambda:\alpha} \langle s', \overline{x}\rangle} \qquad \frac{w \xrightarrow{\lambda:!a} w', \quad r \xrightarrow{?a} r'}{\langle w, r, \overline{y}\rangle \xrightarrow{\lambda:a} \langle w', r', \overline{y}\rangle}$$

In the first rule, $s \xrightarrow{\lambda:\alpha} s'$ stands for a transition of some writer or reader, while $\overline{x}$ stands for the tuple consisting of the local states of all other components.

In the second rule, $w \xrightarrow{\lambda:!a} w'$ and $r \xrightarrow{?a} r'$ stand for a transition of some writer and replica, respectively. Here, $\overline{y}$ stands for the tuple consisting of the local states of all readers and all other writers and replica.

Replicas. The replicas themselves are interpreted as shared data objects among the readers and writers and behave according to the control-flow diagram shown in Fig. 1. We abstract away from the concrete values stored in the object, and only represent the three possible modes of a replica: it can be either consistent, currently modified, or damaged. Therefore, the model for the k^{th} replica consists of the three locations $\mathit{consistent}_k$, $\mathit{currently\_modified}_k$, or $\mathit{damaged}_k$. An integer variable w_k keeps track of the number of writers that are currently writing the k^{th} replica. The edges in the control-flow diagram partly refer to the counter variable w_k by means of a guard or an assignment. The usual unwinding of the control-flow diagram yields a transition system where each state consists of a

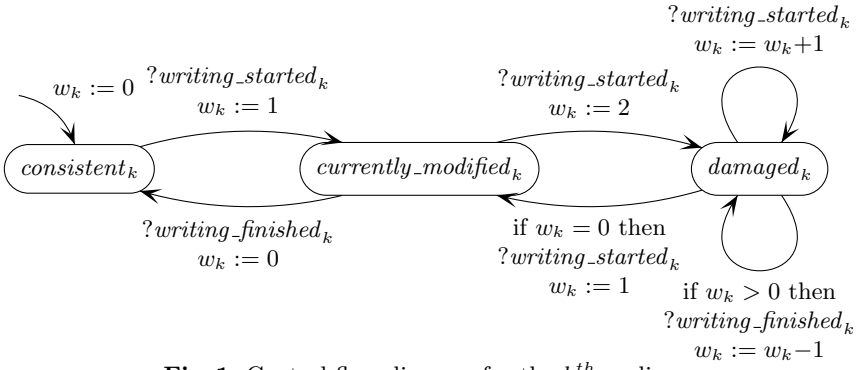


Fig. 1. Control-flow diagram for the k^{th} replica

location and a value for the counter variable w_k and where the transitions are labeled by some receive action $?a$ (without any reference to the counter variable).

Each replica is assumed to be initially consistent and no writer is currently actively writing. In Fig. 1, the transitions $writing\_started_k$ and $writing\_finished_k$ stand for sets of transitions $writing\_started_k^i$ and $writing\_finished_k^i$, $1 \leq i \leq I$ which synchronize with the respective actions triggered by any of the writers. That is, whenever some writer starts operating on the k^{th} replica, which is indicated by the synchronous action $writing\_started_k^i$, the replica changes its location, where it is now considered to be under modification and the counter variable w_k is increased accordingly. Similarly, the synchronous action $writing\_finished_k$ indicates the end of a write operation on the k^{th} replica. A replica is said to be damaged if more than one writer is operating on the replica at the same time. A replica can only become consistent if one writer can successfully write its data without interference from another writer. We mark a damaged replica as currently modified once a single writer starts modifying the replica exclusively. If it succeeds writing the replica without interference from another writer, we consider the replica to be repaired and hence consistent. We say a writer interferes with another reader or writer if it writes a replica that is concurrently read or written by this other process.

Readers. Fig. 2 shows the CTMC formalizing the operational behavior of the j^{th} reader. For each transition we assign a rate of an exponential distribution. The j^{th} reader is initially in the state $idle^j$ before it starts reading the replica in the order of decreasing indices. The delay of the transition from the idle state to the state $reading_K^j$ is exponentially distributed with rate κ. Intuitively, κ defines the "read rate" of an individual reader, which can be understood as the average number of reading requests per time unit in state $idle^j$. The transition from state $reading_k^j$ to state $check_k^j$ with action $reading\_finished_k^j$ fires with rate δ. In states $check_k^j$ the reader checks whether or not the read of the k^{th} replica was successful. The read operation is successful if the replica was found consistent at the beginning of the read and there was no interference from a concurrent write operation of some writer. For the latter, the j^{th} reader has access to the shared boolean variables $corrupted_k^j$. The variables $corrupted_k^j$ are set to *false*

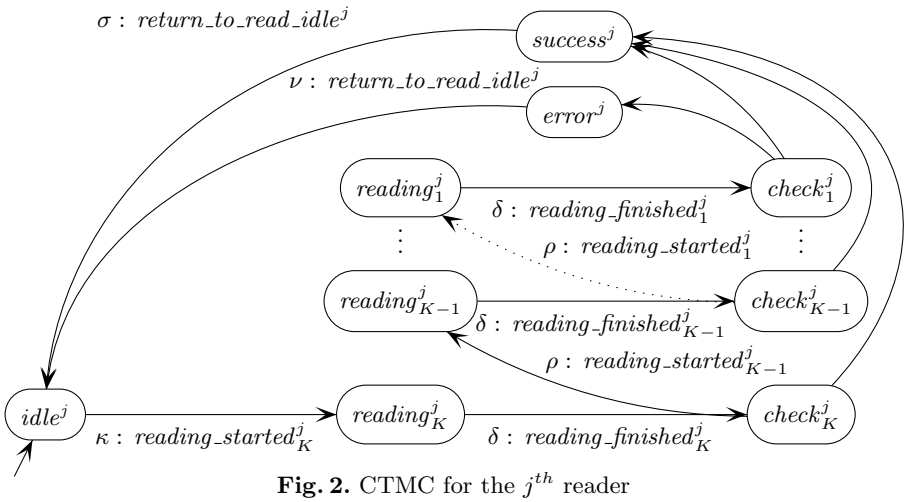


Fig. 2. CTMC for the j^{th} reader

whenever a transition with action $reading\_started^j_k$ is fired and set to *true* by the writers as specified in the next paragraph of this section. In case that the replica was found consistent and no writer was writing the replica concurrently, the j^{th} reader changes its state to $success^j$, from where it returns to state $idle^j$ with rate σ. Otherwise it proceeds reading the next replica with rate $\rho \in \mathbb{R}_{>0}$. If the reader could not find a consistent replica without interference from a writer, the reader moves to the state $error^j$. The transition from $error^j$ to $idle^j$ can be understood as a high-level representation of some error handling which is modeled here stochastically using an exponential distribution with rate ν. (Also the original PWCS protocol proposed by Mc Guire does not consider any concrete policy for the error handling.) In the following, we say a reader is in a read cycle if it is in some state other than $idle^j$ and define the term write cycle accordingly.

Writers. The writers are modeled by the CTMC shown in Fig. 3. The i^{th} writer starts in its initial state $idle^i$ and changes its state to $writing^i_1$ with rate $\gamma \in \mathbb{R}$, while firing the send action $!writing\_started^i_1$ synchronously with the matching receive action $?writing\_started^i_1$ by the first replica. Hence, the writer starts writing the first replica when it enters the location $writing^i_1$. Once the writer has started, it will write all replica in the order of increasing indices (i.e., in reversed order of the readers). When the write operation of the k^{th} replica ($k \in \mathbb{N}$) is finished, the writer changes its state to $ready^i_k$ with the synchronous action $writing\_finished^i_k$ and rate λ before continuing with the next replica. The time to access the next replica is exponentially distributed with rate $\mu \in \mathbb{R}$. After the writer has finished writing the last replica it changes back to state $idle^i$ via action $return\_to\_write\_idle^i$ with rate η. We assume that the effect of firing $writing\_started^i_k$ will be that the shared variables $corrupted^j_k$ are set to *true* for all readers (i.e., $1 \leq j \leq J$). This is to "inform" the readers which are currently reading the k^{th} replica about the interfering write operation. We have introduced this signaling mechanism because the interfering write may corrupt

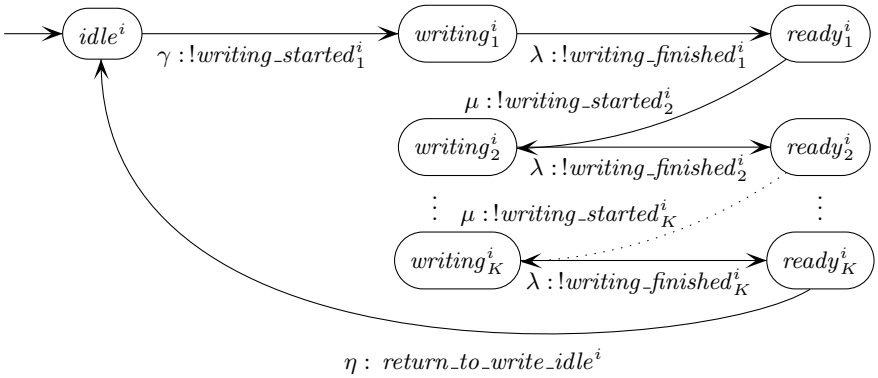


Fig. 3. CTMC for the i^{th} writer

the data the reader retrieves from the replica. Concurrent reads may retrieve corrupt data even though the replica may be consistent before and after the write.

To obtain the PRISM model, we did some technical modifications on the CTMC models presented above. Action labels have been encoded as variables. This enables references to actions in the state-labeled logics CSL and CSRL. Furthermore, we identify the last ready-state $ready_K^i$ with the idle-state $idle^i$.

4 Quantitative Analysis

We now proceed to the quantitative analysis of PWCS. There are several interesting questions to ask about PWCS when one seeks to use it in a specific scenario. For example, what is the likelihood to read a consistent object, how many replicas have to be read for that and how long does it take on average. The following queries, which we have model checked, give answers to these questions and insights into the balance between repair and damage that we found to be crucial for the performance of PWCS in the presence of multiple concurrent writers.

Queries. To obtain deeper insight in PWCS and have analyzed the following six queries. The writer index $1 \leq i \leq I$ and reader index $1 \leq j \leq J$ are arbitrarily chosen but fixed.

Q1: "*Probability to successfully read a replica (on the long run)*": The j^{th} reader successfully reads a replica, when it finds one consistent replica during its read cycle without interference. That is, the reader starts its read cycle when performing the first read action $reading\_started_K^j$ and reaches location $idle^j$ via $success^j$ (rather than via $error^j$). Formally, the task is to compute:

$$\Pr\big(\neg error^j \ \mathcal{U}\ idle^j \ \big|\ reading\_started_K^j\big)$$

Q2: "*The $p \in [0,1]$ time-quantile for a successful read (on the long run) within time bound t*":

$$\min\{t \;:\; p \leq \Pr(\neg \mathit{error}^j \; \mathcal{U}^{\leq t} \; \mathit{idle}^j \mid \mathit{reading\_started}^j_K)\}$$

In this query we are interested in the minimum time bound t such that the probability to successfully read a replica on the long run (cf. Query Q1) is above a certain threshold p.

Q3: "*Fraction of time in which all K replica are damaged*": While all replicas stay damaged, readers have no chance to successfully read a single replica.

$$\theta(\mathit{damaged}_1 \wedge \ldots \wedge \mathit{damaged}_K)$$

For this query we will investigate the effect on the model checking outcome when increasing the number of replicas K present in the model, as this should raise the probability of finding a consistent replica for the readers.

Q4: *"Average time (on the long run) for repairing a damaged replica"*: In this query we are interested in the average repair time once a replica becomes damaged. For the computation we annotate all states of the model with reward 1 and compute the following conditional long-run accumulated reward:

$$\mathrm{AccRew}(\Diamond \mathit{consistent}_k \mid \mathit{just\_damaged}_k)$$

Here, $\mathit{just\_damaged}_k$ is a shorthand notation for the transition in which a second writer starts operating on the k^{th} replica, i.e., the transition from location $\mathit{currently\_modified}_k$ to location $\mathit{damaged}_k$ with an action $\mathit{writing\_started}^i_k$ (cf. Fig. 1).

Q5: *"The $p \in [0,1]$ time-quantile for repairing a damaged replica (on the long run) within time bound t"*: In this query we are interested in the minimum time bound t such that the probability to successfully repair a damaged replica on the long run (cf. Query Q4) is above a certain threshold p.

$$\min\{t \;:\; p \leq \Pr(\Diamond^{\leq t} \mathit{consistent}_k \mid \mathit{just\_damaged}_k)\}$$

Q6: *"The probability to write at least c consistent replica within one write cycle where $c \leq K$."*: We say that the i^{th} writer successfully writes at least c replicas in one write cycle if on the path through the cycle there are at least c indices $\ell = \ell_1, \ldots, \ell_c$, where $w_\ell = 0$, $\mathit{writing\_started}^i_\ell$ is executed and followed by $\mathit{writing\_finished}^j_\ell$ without interfering writes on the ℓ^{th} replica by any of the other writers.

$$\Pr(\Pi_c \mid \mathit{writing\_started}^i_1)$$

Here, the set Π_c consists of all infinite paths that have a finite prefix that meets the constraints imposed above.

5 Evaluation

We have evaluated PWCS for the three different scenarios depicted in Table 1: Scenario 1 (*frequent reads and writes*) is a worst-case setup for PWCS where readers and writers access the shared object as fast as they can. Scenario 2

	Scenario 1		Scenario 2		Scenario 3	
	time	**rate**	**time**	**rate**	**time**	**rate**
idle time (writer)	1	$\gamma = 1$	20	$\gamma = 0.05$	200	$\gamma = 0.005$
idle time (reader)	1	$\kappa = 1$	2	$\kappa = 0.5$	20	$\kappa = 0.05$

parameters common to all scenarios

	time	**rate**
write duration	2	$\lambda = 0.5$
read duration	1	$\delta = 1$
other	0.01	$\mu = \rho = \sigma = \nu = 100$

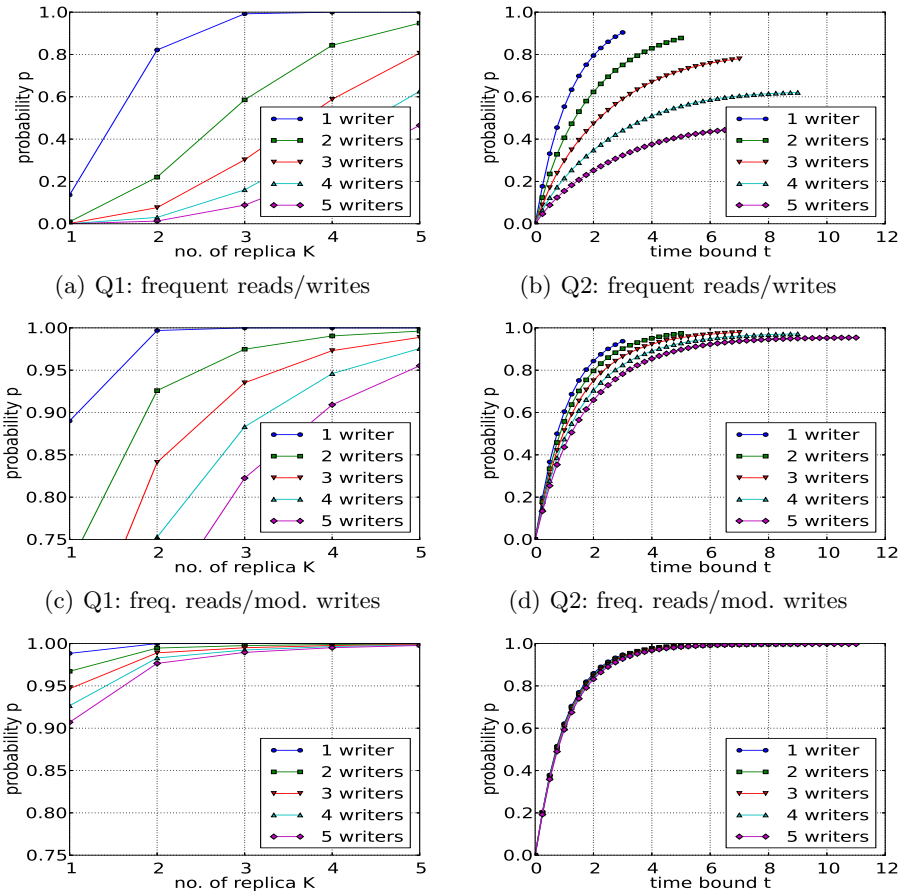
Table 1. Parameters for the three evaluated scenarios.

(*frequent reads/moderate writes*) characterizes a read-most data structure where writers access the object only every 10^{th} read access in average. Scenario 3 (*moderate reads/occasional writes*) is a setup where both readers and writers access different parts of fine-granular synchronized objects or where the times to access objects are significantly smaller than the computation phases between subsequent accesses. Due to the cache-agnostic nature of our CTMC model and because we are primarily interested in the synchronization behavior of one selected reader, we will instantiate our model with one reader (i.e., $J = 1$) and vary the number of writers I between 1 and 5. For the queries Q1, Q3 and Q5, we vary the number of replicas K between 1 and 5. For the remaining queries we fix K to 5. All times are average durations relative to the average read duration.

The computations were carried out on an Intel Core i7 2640M @ 2.8 GHz. For our parameter sets, the model sizes ranged from 13 states ($I = K = 1$) up to approximately 50 million states ($I = K = 5$). By applying PRISM's built-in symmetry reduction, we were able to reduce the state space significantly to about 0.65 million states. Using PRISM's sparse engine, we observed model checking times between a fraction of a second and 6 minutes (Q1, $I = K = 5$). To obtain the long-run probabilities and accumulated rewards, we applied our PRISM extension [BDE+12b] that calculates the weighted sums using the steady-state distribution θ. For Q2 we approximated the time-quantile by sampling with a period of 0.25 time units. In order to compute property Q6 efficiently, we translated it into a nested PCTL query that yields a compact Rabin automaton for the converse property: "The probability to write at most c damaged replicas where $c \leq K$".

Reader Performance (Queries Q1 – Q2). Figs. 4(a), 4(c) and 4(e) show the probability to read a consistent replica in the three analyzed scenarios. For Scenario 2 and 3, we clearly see that as few as four respectively two replicas suffice to reach success rates over 95 % even if replicas are damaged by interfering writers. In the worst case Scenario 1, reads are still successful in over 45 % of all cases once the number of replicas exceeds the number of writers.

Query Q2 projects the Q1 results into the time domain. Figs. 4(b), 4(d) and 4(f) show the probability of reading the shared object successfully within the time bound t. Recall, the average time to read a replica is one time unit.



(a) Q1: frequent reads/writes

(b) Q2: frequent reads/writes

(c) Q1: freq. reads/mod. writes

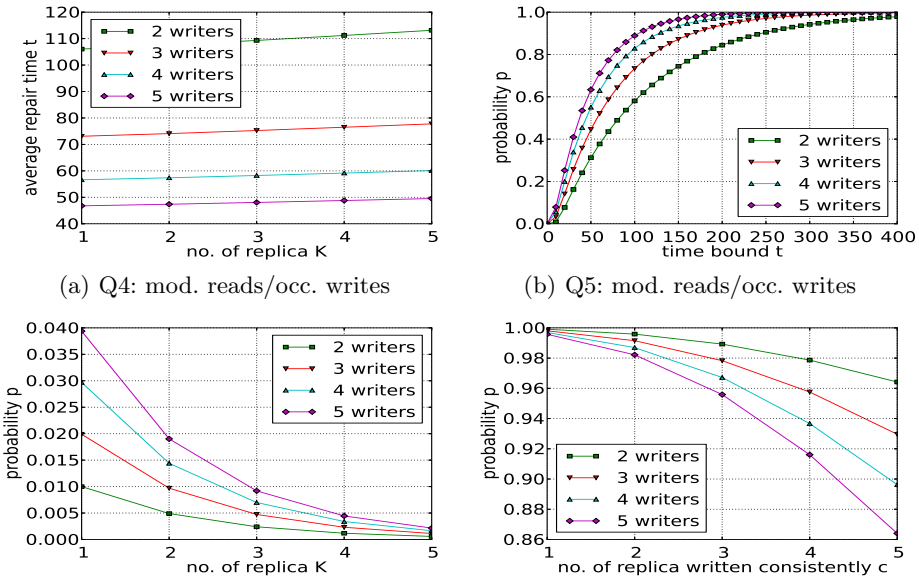
(d) Q2: freq. reads/mod. writes

(e) Q1: mod. reads/occ. writes

(f) Q2: mod. reads/occ. writes

Fig. 4. Results for the Queries Q1 and Q2

As expected, additional writers cause more replica to be inconsistent. The time to find a consistent replica increases with the number of writers. There are two important points to notice. First, if we take the average write duration multiplied by the number of writers plus the average time to read a replica (i.e., 3, 5, 7, 9 and 11 time units for $1, \ldots, 5$ writers, respectively) the probability to have read the object successfully is well over 90 % for Scenario 2 and well over 99 % for Scenario 3 and 2 or more writers. Another point to notice is the gap between the curves and probability 1. In particular for Scenario 1, the 4-writer curve approaches 62 % but never reaches 1. Part of this gap can be explained by writers currently modifying the replica, which renders the replica temporarily inconsistent. To better grasp the influence of damage on this gap, we have analyzed queries Q3 to Q6.



(a) Q4: mod. reads/occ. writes

(b) Q5: mod. reads/occ. writes

(c) Q3: mod. reads/occ. writes

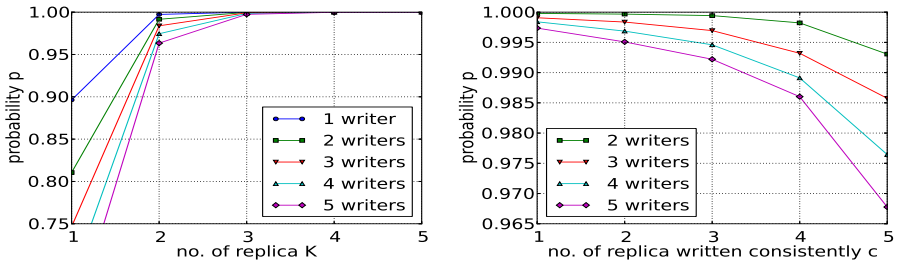
(d) Q6: mod. reads/occ. writes

Fig. 5. Results for the Queries Q3 – Q6

Replica Damage (Queries Q3 – Q6). Fig. 5 shows the results for queries Q3 to Q6. Due to the limited space, we present only the results for Scenario 3. However, we confirmed that the results for Scenario 2 follow the same trend. Q3 shows that in a system with only a single replica, the likelihood that it is damaged is below 4 %. The probability that all replicas are damaged further decreases when the number of replicas is increased. Queries Q4 (Fig. 5(a)) and Q5 (Fig. 5(b)) address the time a replica stays damaged. In both figures, we see a superposition of two effects: more writers damage a replica with higher probability but the higher write rate reduces the time before a replica gets repaired. Q6 confirms these observations by answering the question how likely it is to write c consistent replicas out of 5 replicas. Over 99 % of all writes manage to produce at least one replica, which explains the high success rate of readers.

From these observations, we can conclude: (1) PWCS preserves a high chance of finding the shared object consistent as long as the number of replica exceeds the number of writers. (2) Special precautions to avoid damage or to make damage distinguishable from temporary inconsistencies are not justified.

Experimental Confirmation. To confirm our findings about damaged replicas, we have implemented an element-wise PWCS-protected array. We vary the size of the array to adjust read and write rates. Fig. 6 shows the results for queries Q1 and Q6. All measurements were taken on an Intel Xeon X5650 @ 2.67 GHz (hyperthreading disabled). The array remains in the shared L3 cache throughout our benchmark. The measurements confirm the analytical results except in the one replica case in Q1. We attribute this deviation to interference from the cache



(a) Q1: mod. reads/occ. writes (b) Q6: mod. reads/occ. writes

Fig. 6. Measurement results for the Queries Q1 and Q6

coherence protocol, which prior to writing invalidates copies of the replica of all other readers and writers. Our model does not take cache effects into account.

Notice that we do not give graphs for Q4 and Q5 because both queries are very hard to measure, since they require global knowledge and very tight synchronized clocks. Moreover, the operations to gather and distribute this information from the damaging writers to the repairing writer would influence the measurement code such that they conceal the actual effect we would like to measure.

6 Conclusions

This work presented a quantitative analysis of Probabilistic-Write/Copy-Select (PWCS) using continuous-time Markov chains and probabilistic model-checking techniques implemented in the model checker PRISM. PWCS is a new synchronization protocol based on the implicit-randomness induced by the complexity of todays many-core systems. In our analysis, we were able to confirm Mc Guire's measure-based experiments: few replicas suffice to maintain a high probability ($> 95\,\%$) of finding a consistent replica. We established these results for the common situations where reads dominate the shared object accesses. In addition, we also confirmed these findings for more exceptional scenarios with frequent writes.

We extended PWCS and considered multiple, parallel writers without additional synchronization. Our analysis revealed a high probability of repairing damaged replicas within reasonable time bounds. This high repair rate translates into a low probability ($< 4\,\%$) to actually damage the object by damaging all its replicas.

A particularly interesting point of our formal analysis is that it revealed insights in the behavior of PWCS that evade measurement-based investigations. We consider this as a general advantage of probabilistic-model checking and plan to investigate further low-level algorithms that exhibit very short runtimes and where the instrumentation-induced interfere conceals the quantities to measure. In addition, we plan to look into further variants of PWCS and, more generally, into implicit-randomness based stochastic algorithms.

References

ASSB00. A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Model checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

BDE+12a. C. Baier, M. Daum, B. Engel, H. Härtig, J. Klein, S. Klüppelholz, S. Märcker, H. Tews, and M. Völp. Chiefly symmetric: Results on the scalability of probabilistic model checking for operating-system code. In *SSV 2012*, volume 102 of *EPTCS*, pages 156–166, 2012.

BDE+12b. C. Baier, M. Daum, B. Engel, H. Härtig, J. Klein, S. Klüppelholz, S. Märcker, H. Tews, and M. Völp. Waiting for locks: How long does it usually take? In *FMICS 2012*, volume 7437 of *LNCS*, pages 47–62. Springer, 2012.

BHHK00. C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *27th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.

BHHK03. C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

CGH+10. N. Coste, H. Garavel, H. Hermanns, F. Lang, R. Mateescu, and W. Serwe. Leveraging applications of formal methods, verification, and validation. In *4th International Symposium on Leveraging Applications (ISoLA (2))*, volume 6416 of *LNCS*, pages 128–142. Springer, 2010.

GM99. E. Gafni and M. Mitzenmacher. Analysis of timing-based mutual exclusion with random times. In *18th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 13–21. ACM, 1999.

Gui11. N. Mc Guire. Probabilistic write copy select. In *13th Real-Time Linux Workshop*, pages 195–206, Oct. 2011.

KNP04. M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *STTT*, 6(2):128–142, 2004.

KNP05. M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: case studies with prism. *SIGMETRICS Performance Evaluation Review*, 32(4):16–21, 2005.

KNP09. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: probabilistic model checking for performance and reliability analysis. *SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

KS60. J. Kemeny and J. Snell. *Finite Markov Chains*. D. Van Nostrand, 1960.

Kul95. V. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995.

KZH+11. J.-P. Katoen, I. S. Zapreev, E. Moritz Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.

MCS91. J. Mellor-Crummey and M. Scott. Scalable reader-writer synchronization for shared-memory multiprocessors. In *PPOPP'91*, pages 106–113. ACM, April 1991.

MS10. R. Mateescu and W. Serwe. A study of shared-memory mutual exclusion protocols using CADP. In *15th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, volume 6371 of *LNCS*, pages 180–197. Springer, 2010.