

Solving Operating-Systems Problems with Probabilistic Model Checking

Hendrik Tews

Institute for Theoretical Computer Science
Office at Operating systems group

Resilience talk
Mai 3, 2013

Outline

Introduction

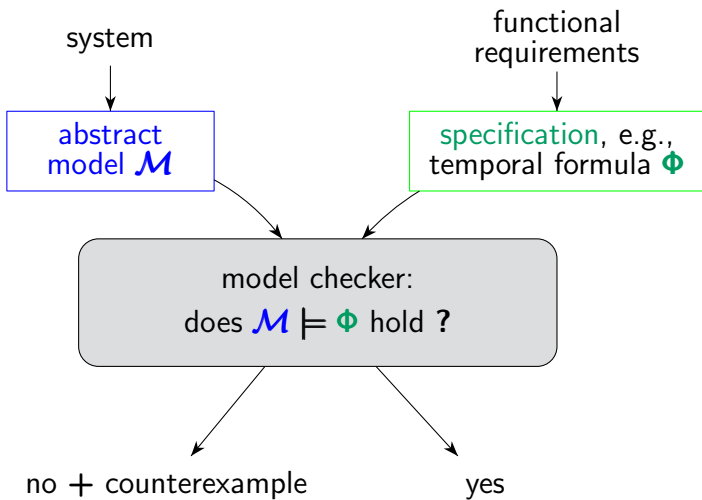
Spinlock Case Study

PWCS — probabilistic write-copy-select

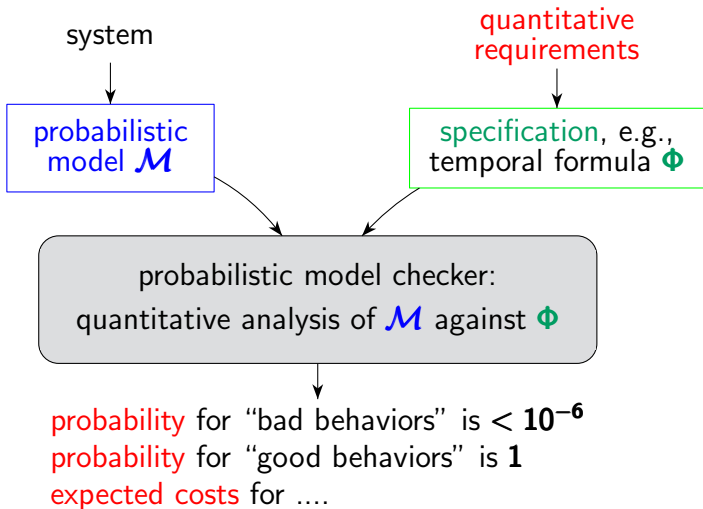
Romain Case Study

Conclusion

Model checking



Probabilistic model checking



Outline

Introduction

Spinlock Case Study

PWCS — probabilistic write-copy-select

Romain Case Study

Conclusion

Spinlocks

Problem

- ▶ n Processes on n CPU cores
- ▶ cooperate to protect a shared resource (OS-kernel ready-queue)
- ▶ Contention is rare, the lock is almost always free
- ▶ Inter-processor interrupts (IPI's) are far too slow in this case

Solution

- ▶ Synchronise over a shared lock variable
- ▶ change lock variable with atomic operations (CAS — compare and swap)
- ▶ expensive in the contention case

Questions

- ▶ Does it scale to 100 cores?
- ▶ For which workloads?

Spinlocks

Joint work with Christel Baier, Marcus Daum, Benjamin Engel, Hermann Härtig, Joachim Klein, Sascha Klüppelholz, Steffen Märcker and Marcus Völz

FMICS 2012 **Waiting for locks: How long does it usually take?**, in: M. Stoelinga, R. Pinger (Eds.), 17th International Workshop on Formal Methods for Industrial Critical Systems, Vol. 7437 of Lecture Notes in Computer Science, Springer, 2012, pp. 47–62.

SSV 2012 **Chiefly symmetric: Results on the scalability of probabilistic model checking for operating-system code**, in: F. Cassez, R. Huuck, G. Klein, B. Schlich (Eds.), Proceedings Seventh Conference on Systems Software Verification, Vol. 102 of EPTCS, 2012, pp. 156–166.

Test-And-Test-And-Set Lock

```
1 volatile bool occupied = false;

2 volatile void lock(){
3     while (atomic_swap(occupied, true)){
4         while (occupied){/* spin loop */}
5     }
6 }

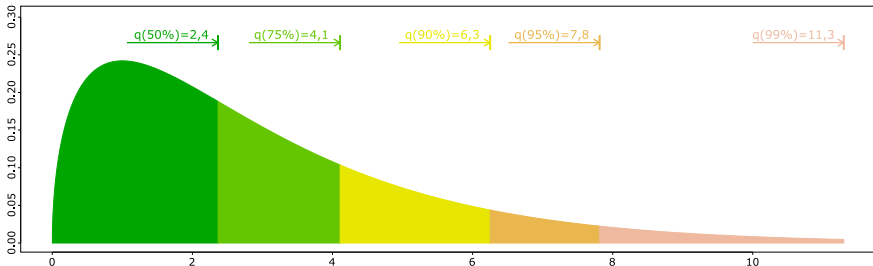
7 void unlock(){
8     occupied = false
9 }
```

- ▶ model n processes that compete for the lock
- ▶ model lock as separate process
- ▶ compare results with measurements

Interesting properties

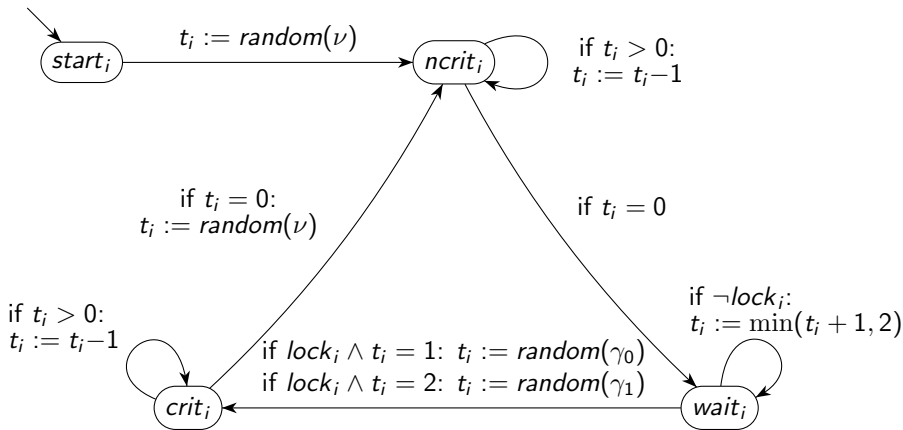
In the long run...

- ▶ Probability for finding the lock free
- ▶ Probability for getting the lock twice in a row without waiting
- ▶ Average waiting time for the lock (under the condition that the lock busy)
- ▶ the 95% quantile of the waiting time



quantile picture by Rene Schwarz from Wikimedia Commons

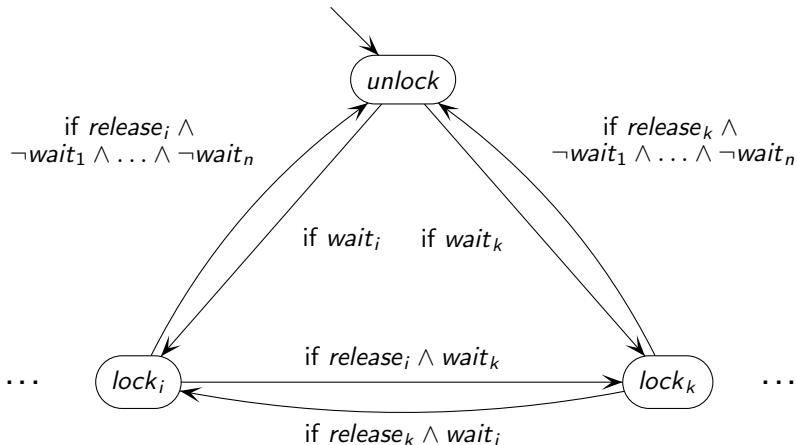
Process i : DTMC Model



Distributions:

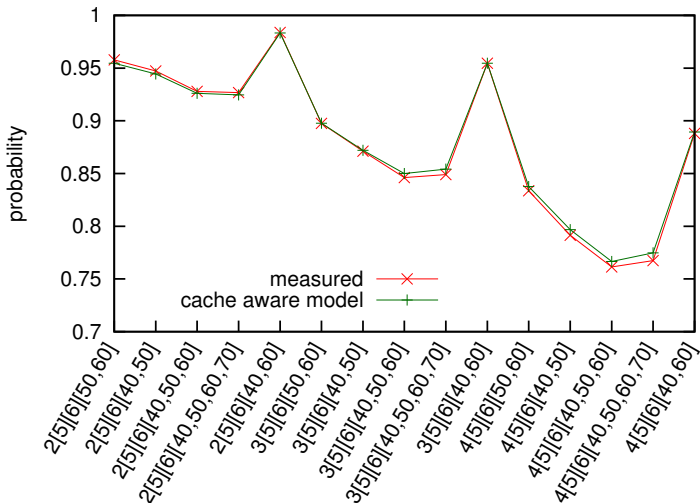
- ν non-critical region
- γ_0 critical region (without spinning)
- γ_1 critical region (with spinning)

The lock: DTMC Model

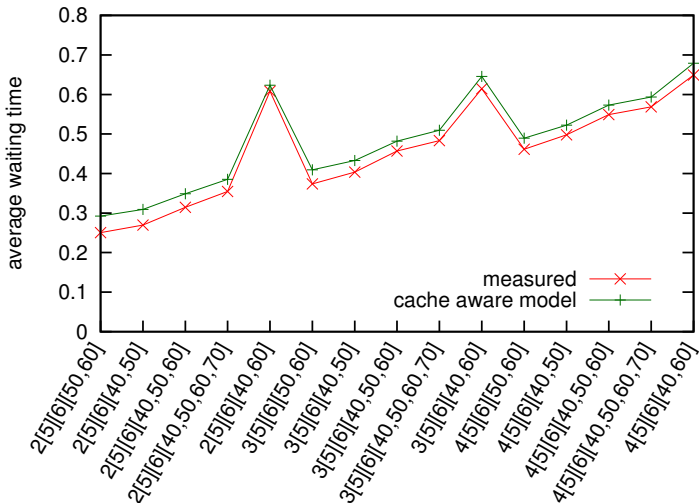


perform uniform probabilistic choice for selecting next lock owner

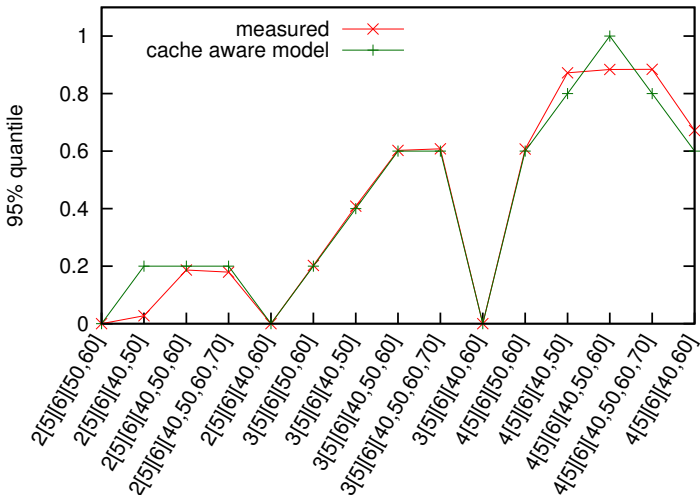
Results: Probability to find the lock free



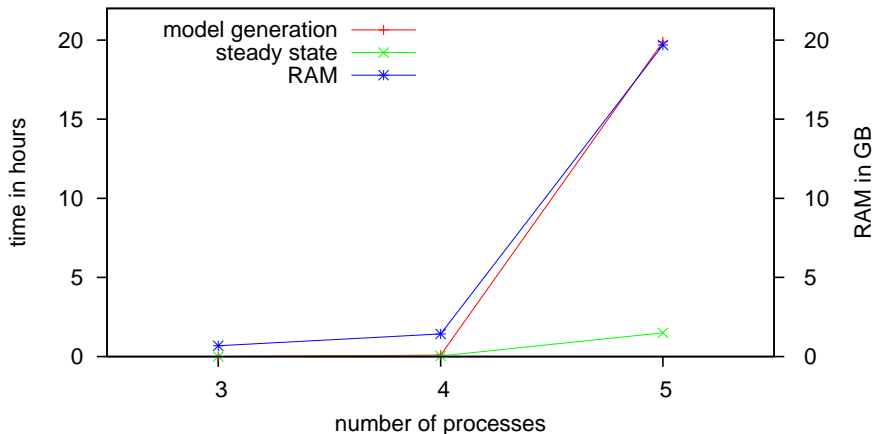
Results: Average waiting time for spinning processes



Results: 95% quantile of the waiting time

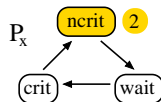
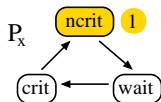
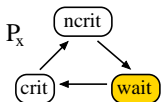
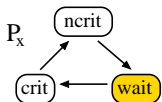
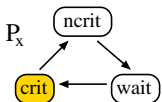
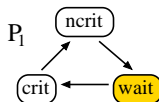
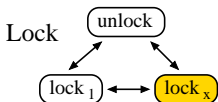


Scalability for PRISM, Distribution [40,50]

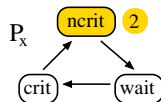
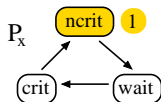
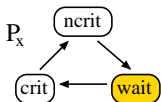
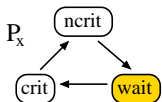
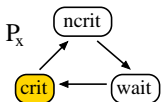
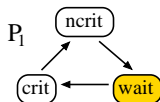
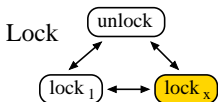


number of states: 3 proc. 4,082,808
 4 proc. 198,808,720

Symmetry reduction: Using a generic representative



Symmetry reduction: Using a generic representative

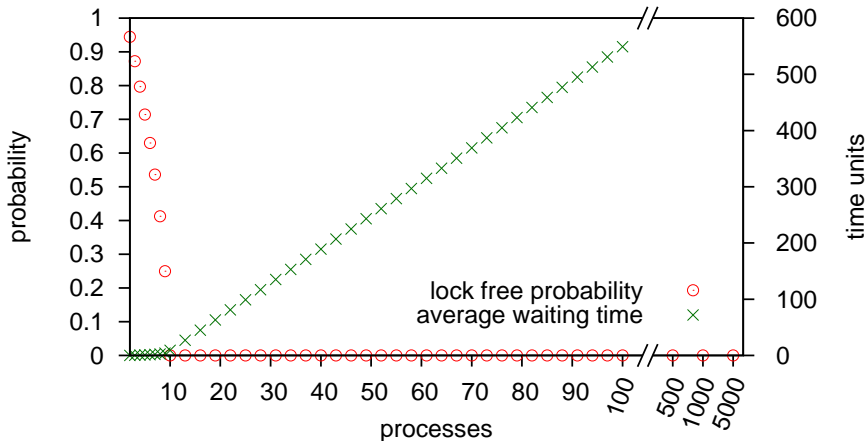


state counters:

crit : 1	ncrit 1 : 1
wait : 2	ncrit 2 : 1

Results for symmetry-reduced model

Non-critical Distribution [40, 50]



Why does it scale so extremely well?

Lock is oversaturated

- ▶ 1 process is in the critical section
- ▶ 6–9 processes are in their non-critical section
- ▶ the remaining processes spin
- ▶ adding another process only increases the spinning-counter by 1

Processes in non-critical region show a regular pattern

- ▶ 1 process releases the lock (circa) every 6 time units
- ▶ chooses a non-critical waiting time of 40 or 50 time units
- ▶ distance of waiting time is regular

Spinlock Conclusion

Lessons learned

- ▶ model checking can be used to predict properties of real systems
- ▶ abstract complicated cache behaviour
- ▶ impressive scalability with custom symmetry reduction

A spin lock for 10,000 processes?

- ▶ certainly nonsense if the lock is saturated for 10 processes already, but
- ▶ overbooked services exist
- ▶ symmetry reduction will yield similar improvements there

Outline

Introduction

Spinlock Case Study

PWCS — probabilistic write-copy-select

Romain Case Study

Conclusion

Scalability with shared resources

Traditional locking does not scale any more

- ▶ atomic operations are slow
 - CAS L1 hit ~ 30 cycles
 - CAS cache miss ~ 250 cycles
- ▶ the speed of light is limited

Read-copy-update (RCU)

Reader-writer Lock

- ▶ permit multiple readers

- ▶ readers can always proceed
- ▶ writers modify private copy and switch atomically
- ▶ readers may see outdated version

PWCS

- ▶ no locks, no atomic operations
- ▶ make inconsistencies detectable

PWCS

Joint work with Christel Baier, Sascha Klüppelholz, Steffen Märcker, Marcus Völp, Benjamin Engel

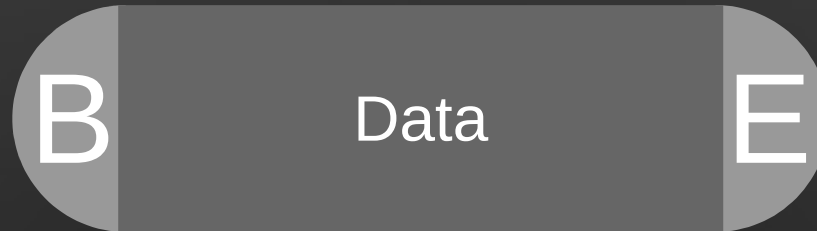
Nasa FM 2013

A Probabilistic Quantitative Analysis of Probabilistic-Write/Copy-Select, in: Brat, G.; Rungta, N.; Venet, A. (Eds.), 5th International Symposium, NFM 2013, Vol. 7871 of Lecture Notes in Computer Science

PWCS Protocol

Writer

```
tag_end++;  
write_data();  
tag_begin++;
```



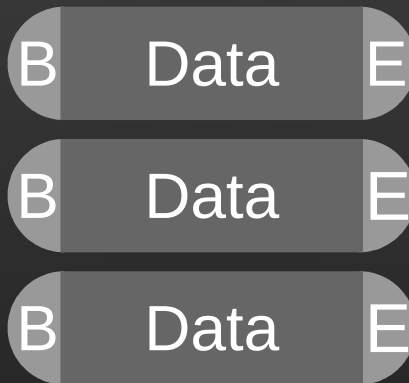
Reader

```
ta = tag_begin;  
copy_data();  
tb = tag_end;  
if (ta == tb)  
    return data;
```

Use Replication

Writer

```
for (i=0; i<3; i++) {  
    r = replica[i];  
    r.tag_end++;  
    r.write_data();  
    r.tag_begin++;  
}
```



Reader

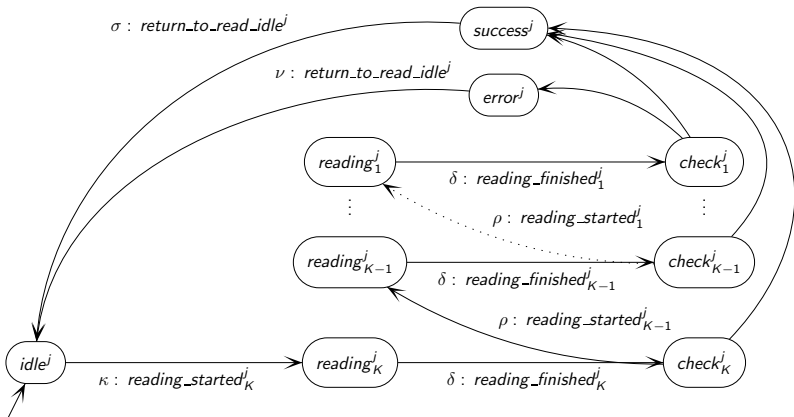
```
for (i=0; i<3; i++) {  
    r = replica[i];  
    ta = r.tag_begin;  
    r.copy_data(&data);  
    tb = r.tag_end;  
    if (ta == tb)  
        return data;  
}  
  
// retry/error handling
```

Interesting Properties

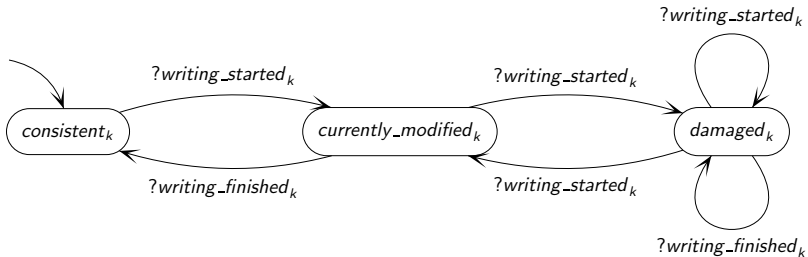
In the long run ...

- ▶ Probability to successfully read the data
- ▶ the 99% time-quantile for successful reading
- ▶ time fraction in which all replicas are damaged
- ▶ average time for repairing a damaged replica

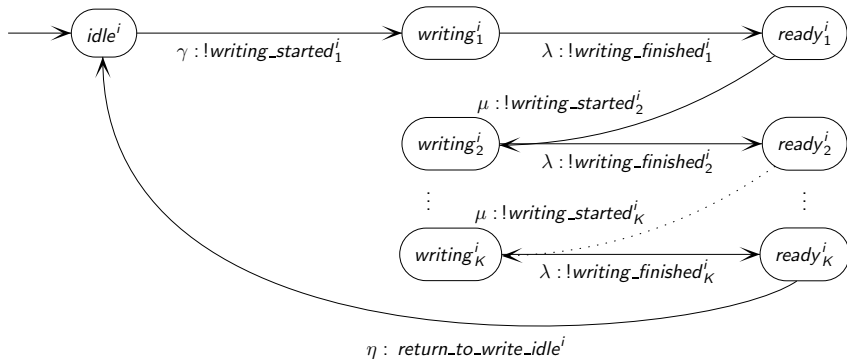
Reader Model



Copies



Writer Model



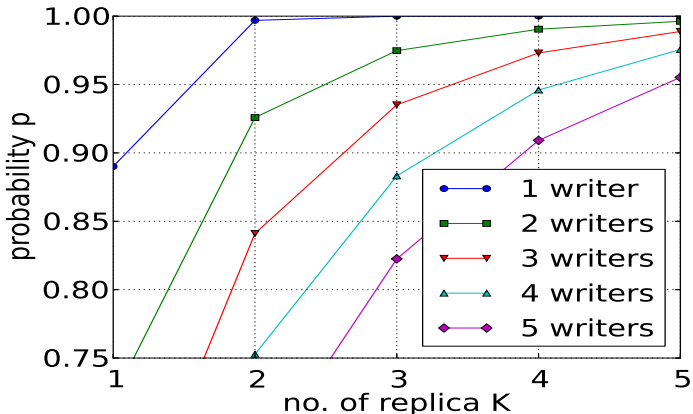
Scenarios

	frequent reads moderate writes		moderate reads moderate writes	
	time	rate	time	rate
idle time (writer)	20	$\gamma = 0.05$	200	$\gamma = 0.005$
idle time (reader)	2	$\kappa = 0.5$	20	$\kappa = 0.05$

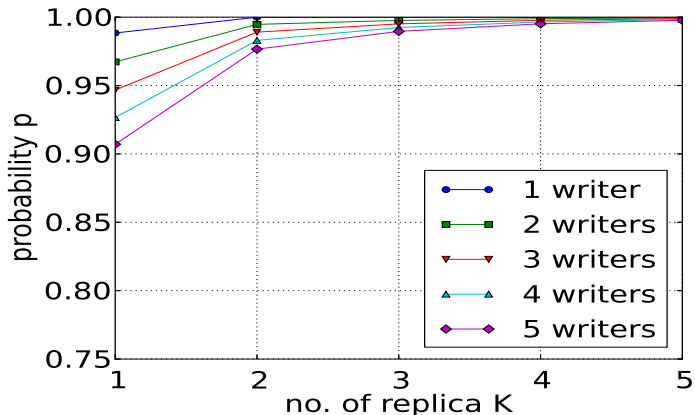
Common parameters

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

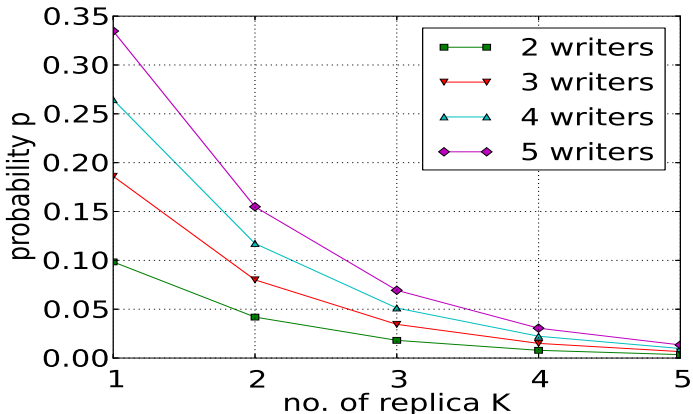
Results: Successfully read data, frequent reads, moderate writes



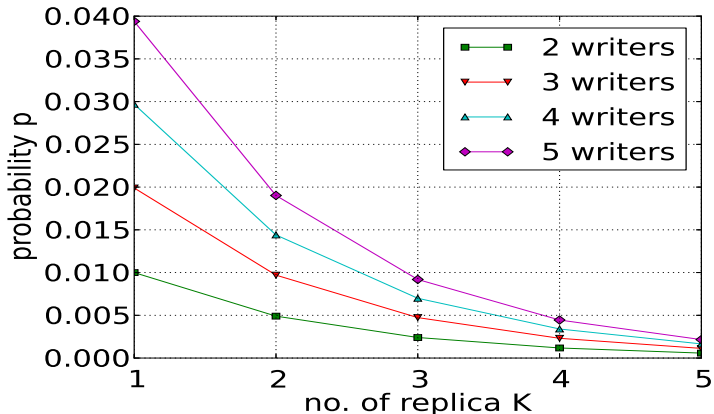
Results: Successfully read data, moderate reads, moderate writes



Results: All replicas damaged, frequent reads, moderate writes



Results: All replicas damaged, moderate reads, moderate writes



PWCS Conclusion

- ▶ investigated different workloads
- ▶ PWCS performs surprisingly well
- ▶ model contains no error handling
(read finally failed, replica damaged)
- ▶ analysis provides hints for error handling

Outline

Introduction

Spinlock Case Study

PWCS — probabilistic write-copy-select

Romain Case Study

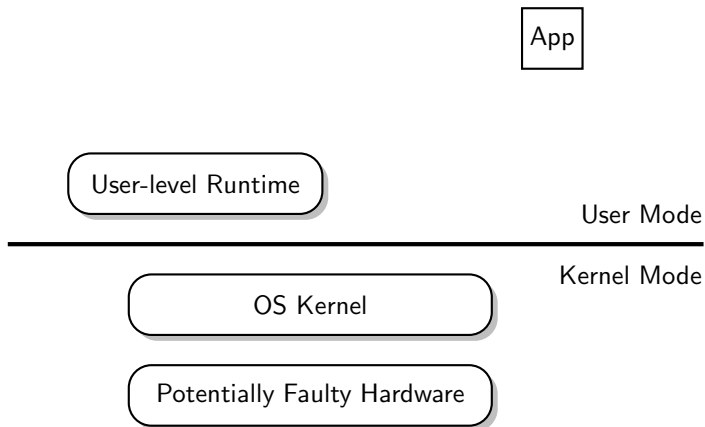
Conclusion

Romain Case Study

Sketch ideas for model-checking resilience properties

(joint work with Björn Döbel)

Resilient OS Structure (by Björn Döbel)



Resilient OS Structure (by Björn Döbel)



User-level Runtime

Romain

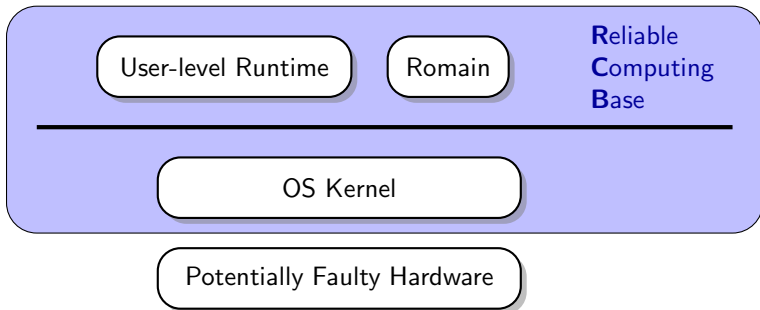
User Mode

OS Kernel

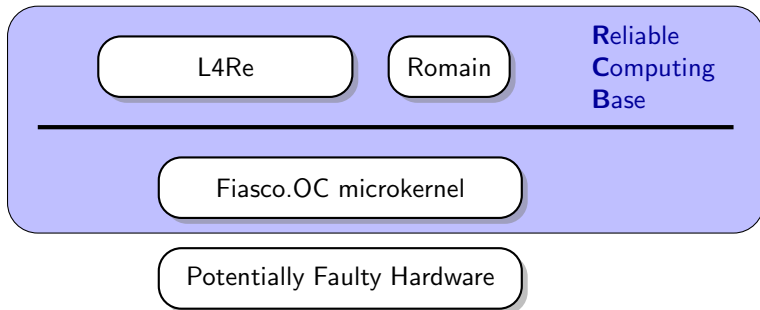
Kernel Mode

Potentially Faulty Hardware

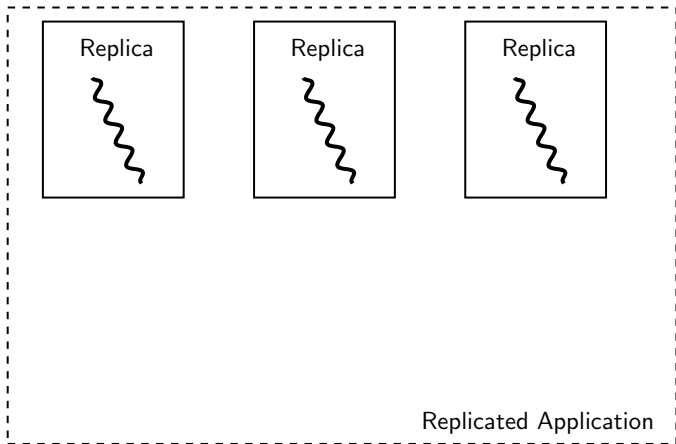
Resilient OS Structure (by Björn Döbel)



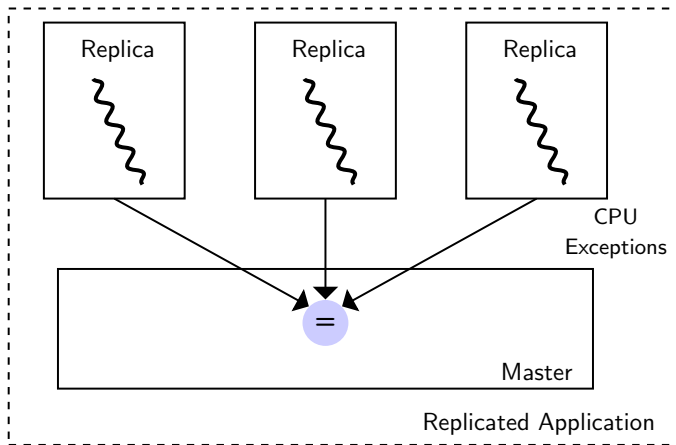
Resilient OS Structure (by Björn Döbel)



Romain: Redundant Replication

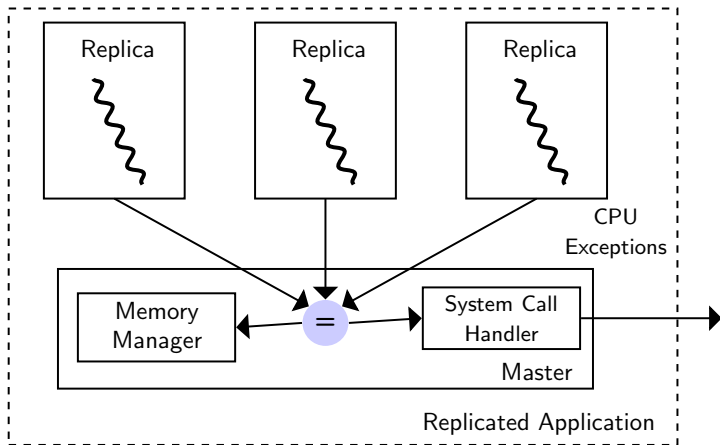


Romain: Redundant Replication



- ▶ majority voting
- ▶ forward recovery

Romain: Redundant Replication



- ▶ majority voting
- ▶ forward recovery

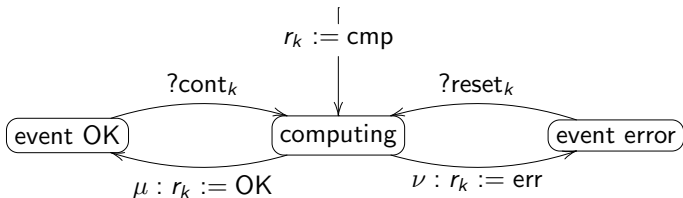
Interesting Properties

In the long run ...

- ▶ reliability depending on the number of replicas
probability of propagating an error because of biased majority
- ▶ resource consumption (trivial: $\# \text{ replicas} \times c$)
- ▶ energy consumption depending on reliability
- ▶ performance decrease
- ▶ probability of non-recoverable errors

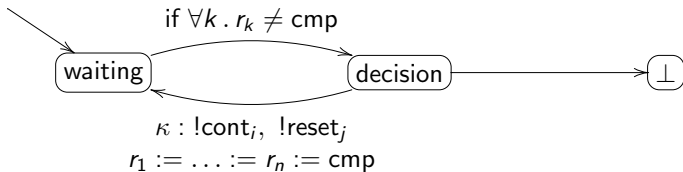
Model for replica k

Replica state $r_k = \text{cmp} \mid \text{OK} \mid \text{err}$



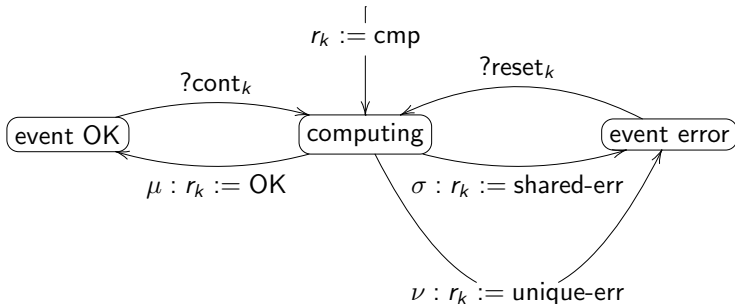
Model of master

Master



Model identical errors in different replicas

Replica state $r_k = \text{cmp} \mid \text{OK} \mid \text{unique-err} \mid \text{shared-error}$



No results yet for Romain

Outline

Introduction

Spinlock Case Study

PWCS — probabilistic write-copy-select

Romain Case Study

Conclusion

Conclusion

- ▶ probabilistic model checking can compute performance measures
 - ▶ energy
 - ▶ time
 - ▶ quantiles
- ▶ model size is always an issue
- ▶ model checking is a push-button technology

Conclusion

- ▶ probabilistic model checking can compute performance measures
 - ▶ energy
 - ▶ time
 - ▶ quantiles
- ▶ model size is always an issue
- ▶ model checking is a push-button technology
 - ▶ you may need an expert to push the button
 - ▶ pushing the button for the first time may take some time