A Markov Reward Model Checker & How Fast and Fat is Your Probabilistic Model Checker?

Presented by Robert Jørgensgaard Engdahl

Joost-Pieter Katoen, Maneesh Khattri and Ivan S. Zapreev &
David N. Jansen, Joost-Pieter Katoen, Marcel Oldenkamp and and Mariëlle Stoelinga and Ivan Zapreev

november 16, 2007

The Two Papers

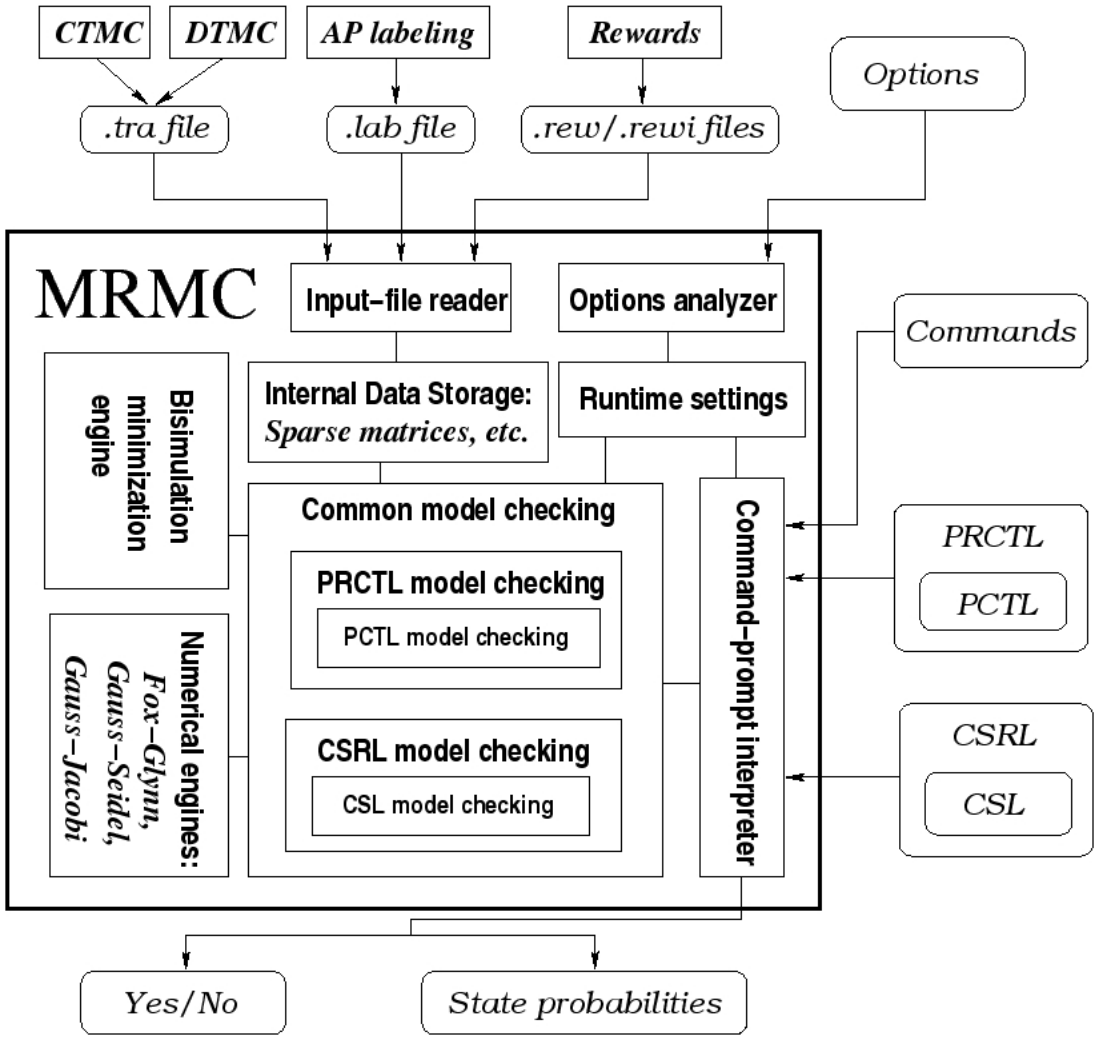
A Markov Reward Model Checker A tool (back-end) for performing model checking on Markov reward models.

How Fast and Fat is Your Probabilistic Model Checker A benchmark of mrmc and other model checking tools on Markov chains, discrete as continuous.

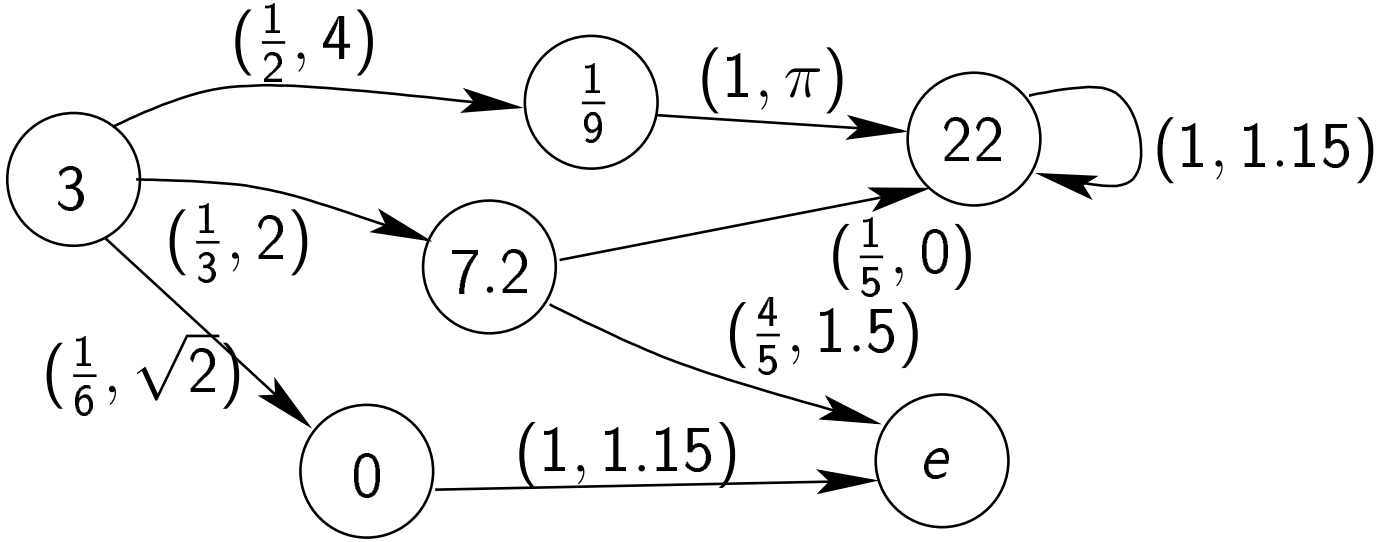
Programme

Next Section

Overview



Discrete Markov Reward Models



Definition

A discrete Markov reward model $\mathcal{M}$ is a tuple (S, P, ρ, ι), where S is a finite set of states; $P : S \times S \to [0; 1]$ is a probability transition function; $\rho : S \to \mathbf{R}$ is a reward rate function and $\iota : S \times S \to \mathbf{R}$ is an impulse reward function.

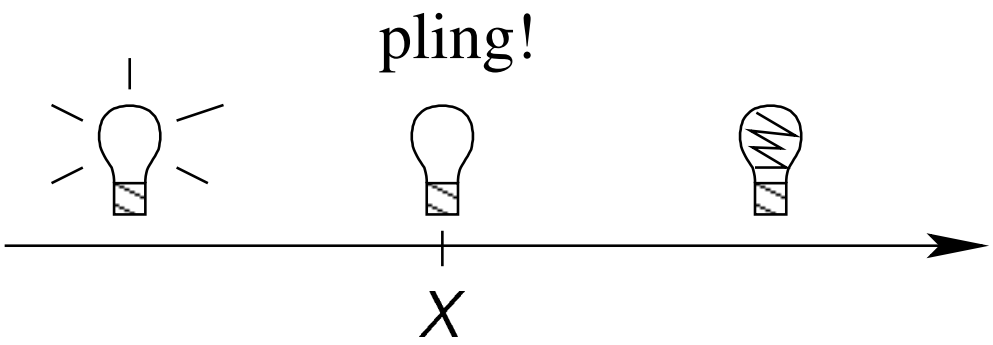
Probabilistic Reward Computation Tree Logic

```
CONST =   ff | tt
R     =   'real number'
N     =   'natural number'
OP    =   > | < | <= | >=

PFL   =   X SFL
        | SFL U SFL
        | SFL U[ N, N ][ R, R ] SFL
        | SFL U[ R, R ] SFL
```

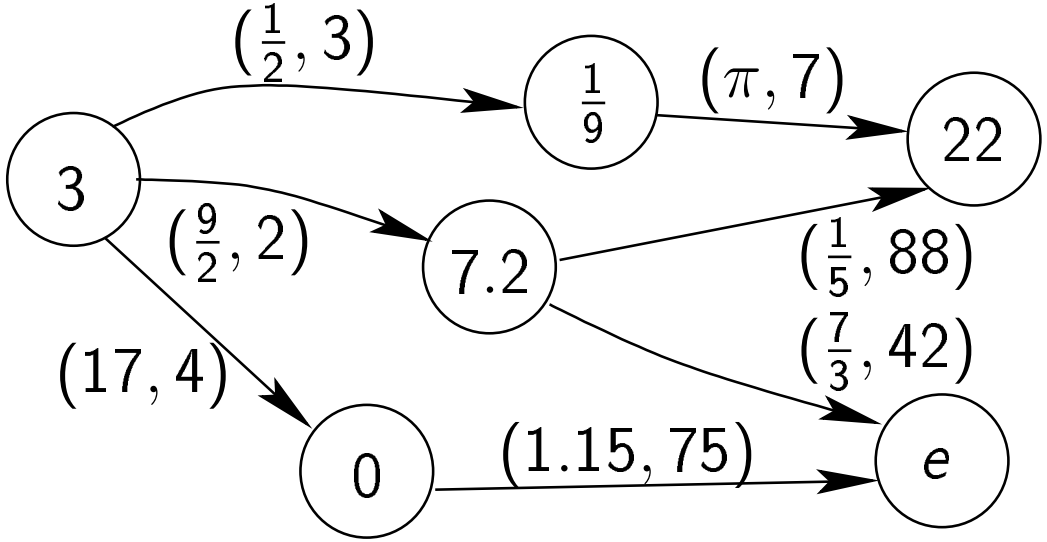
```
SFL   =   CONST
        | LABEL
        | ! SFL
        | SFL && SFL
        | SFL || SFL
        | ( SFL )
        | E [ R, R] [ SFL ]
        | E [N][ R, R] [ SFL ]
        | C [N][ R, R] [ SFL ]
        | Y [N][ R, R] [ SFL ]
        | L{ OP R }[ SFL ]
        | P{ OP R }[ PFL ]
```

The exponential distribution



- Failure probability is time-homogeneous.
- Cumulative distribution function
  $F(x) = Pr(X \leq x) = 1 - e^{-\lambda x}$
- Probability density function $f(x) = \lambda e^{-\lambda x}$.
- Expectation value (mean) $E[X] = \frac{1}{\lambda}$.

Continuous Markov Reward Model



Definition

A continuous Markov reward model $\mathcal{M}$ is a tuple (S, R, ρ, ι), where S is a finite set of states; $R : S \times S \to \mathbf{R}$ is a transition rate function; $\rho : S \to \mathbf{R}$ is a reward rate function and $\iota : S \times S \to \mathbf{R}$ is an impulse reward function.

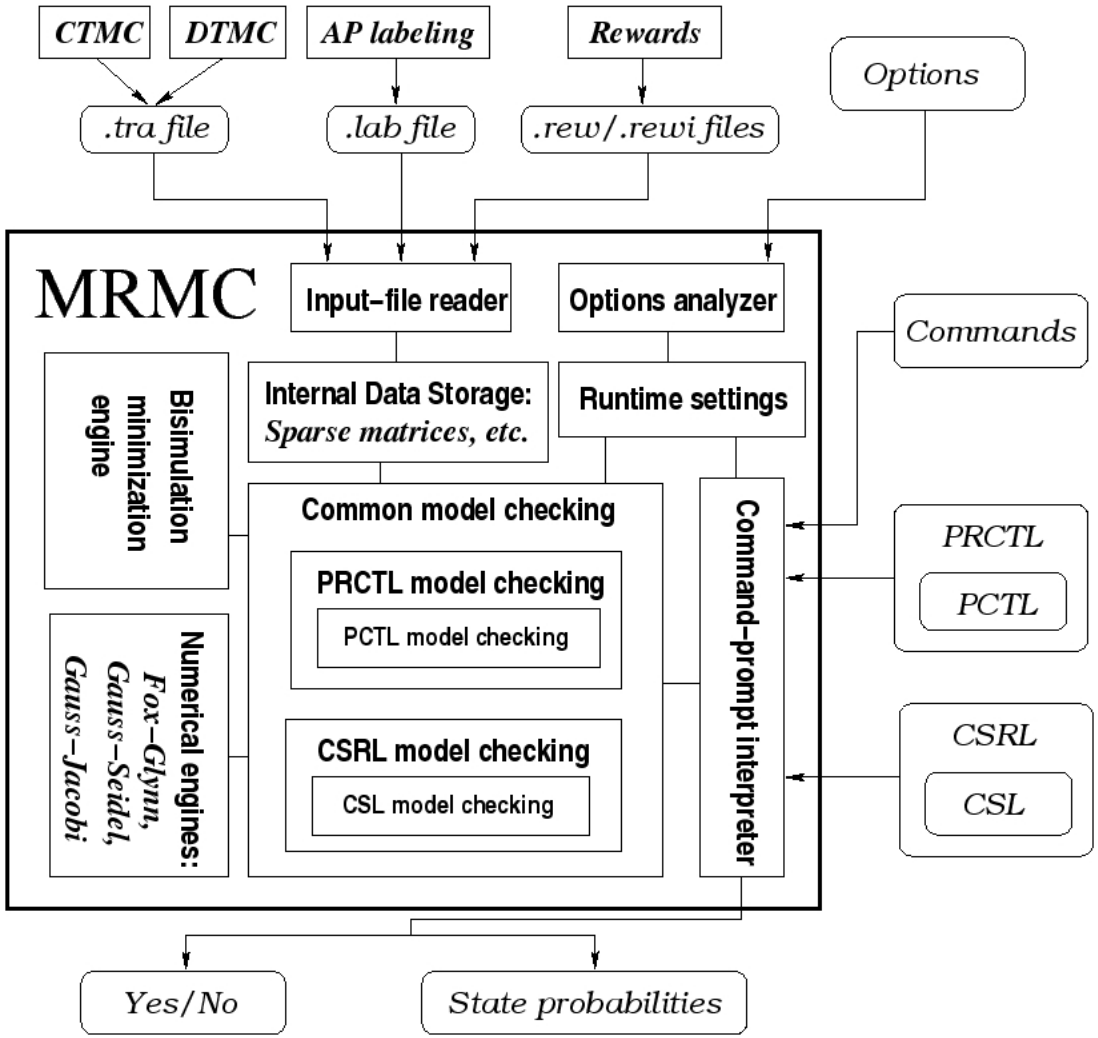
Continuous Stochastic Reward Logic

```
CONST =   ff | tt
R     =   'real number'
OP    =   > | < | <= | >=

PFL   =   X SFL
        | X [R, R] SFL
        | X [R, R][R, R] SFL
        | SFL U SFL
        | SFL U[ R, R ] SFL
        | SFL U[ R, R ] [ R, R ] SFL

SFL   =   CONST
        | LABEL
        | ! SFL
        | SFL && SFL
        | SFL || SFL
        | ( SFL )
        | S{ OP R }[ SFL ]
        | P{ OP R }[ PFL ]
```

Summary of MRMC



Next Section

Method

Tools: ETMCC (Java), MRMC (C), PRISM (Java and C++), VESTA (Java / statistical) and YMER (C++ / CTMC)

Formalisms: PCTL on DTMCs and CSL on CTMCs.

Models: DTMCs: Synchronous Leader Election, Randomised Dining Philosophers and Birth-Death Process.
CTMCs: Tandem Queueing Network and Cyclic Server Polling System.

Measures: Maximum memory consumption and CPU-time.

Scientific Basis

- Repeatability and Verifiability
- Statistical Significance
- Encapsulation
- Representative Samples?

Synchronous Leader Election

n processes are connected in a unidirectional network, forming a ring. In a round, each process selects a random id for itself in $1 \ldots k$. These are passed around s.t. all ids are known to everyone. If there is a unique id, the largest is elected leader. If not, a new round begins.

Randomised Dining Philosophers

n philosophers are sitting around a table with a bowl of infinite noodles. There is a single chopstick between each philosopher. Philosophers get hungry at random, and then they try to pick up chopsticks. if either one of them is missing, the philosopher gives up (lays down chop stick) and may become hungry again later.

Birth-Death Process

In a population one member is born ($n++$) or one member dies ($n--$) for each discrete time unit. When $n = 0$ or $n = m$ the process stops.

Tandem Queueing Network

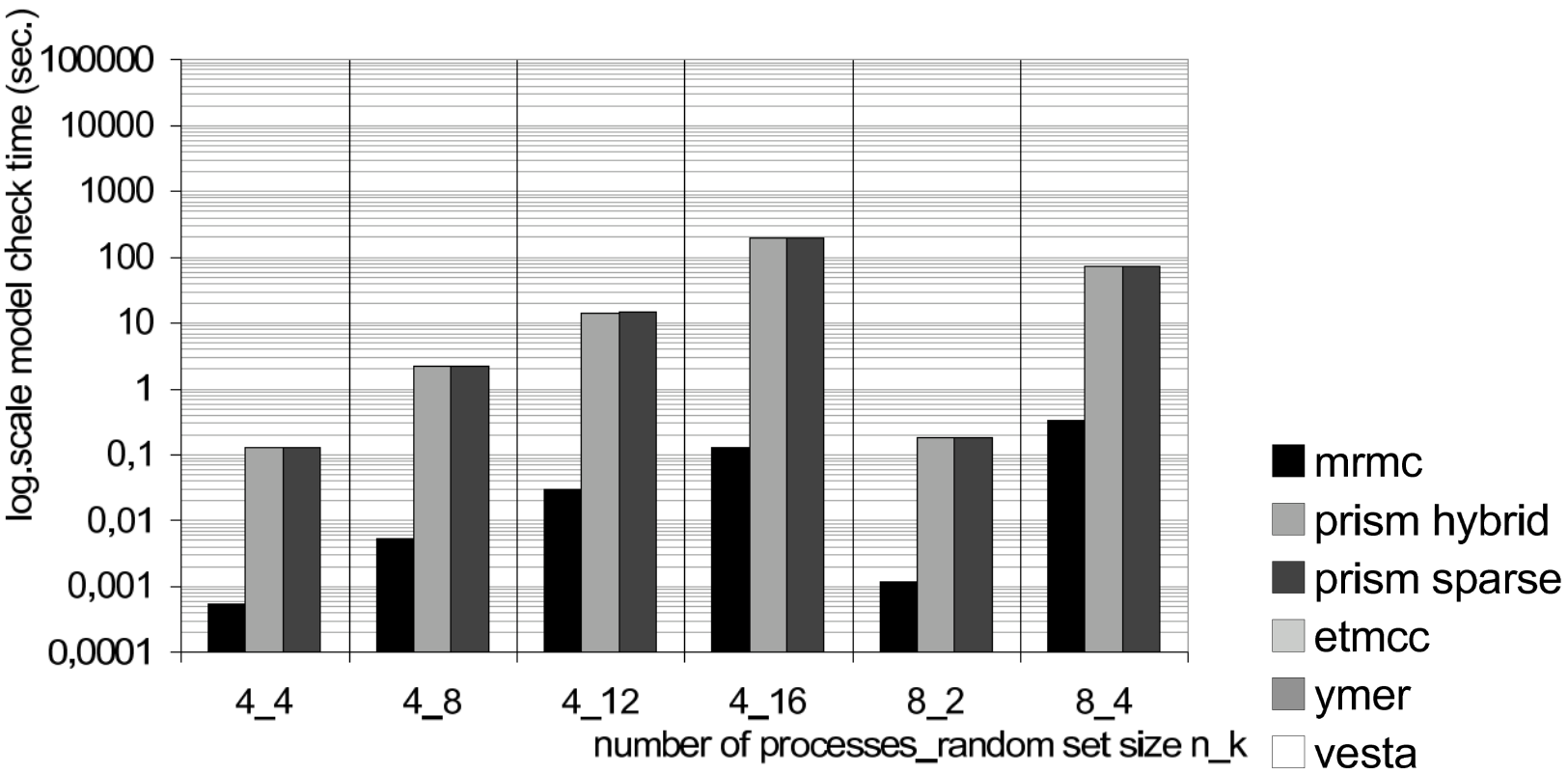
A server has two queues. One for incoming requests, and one for outgoing response. Incoming requests arrives at random, then gets served at random and placed in the outgoing response queue, where they leave at random.

Cyclic Server Polling System

A cyclic server polling system consists of n stations able to hold a single request each. Requests arrive at stations at random. The server polls requests from stations

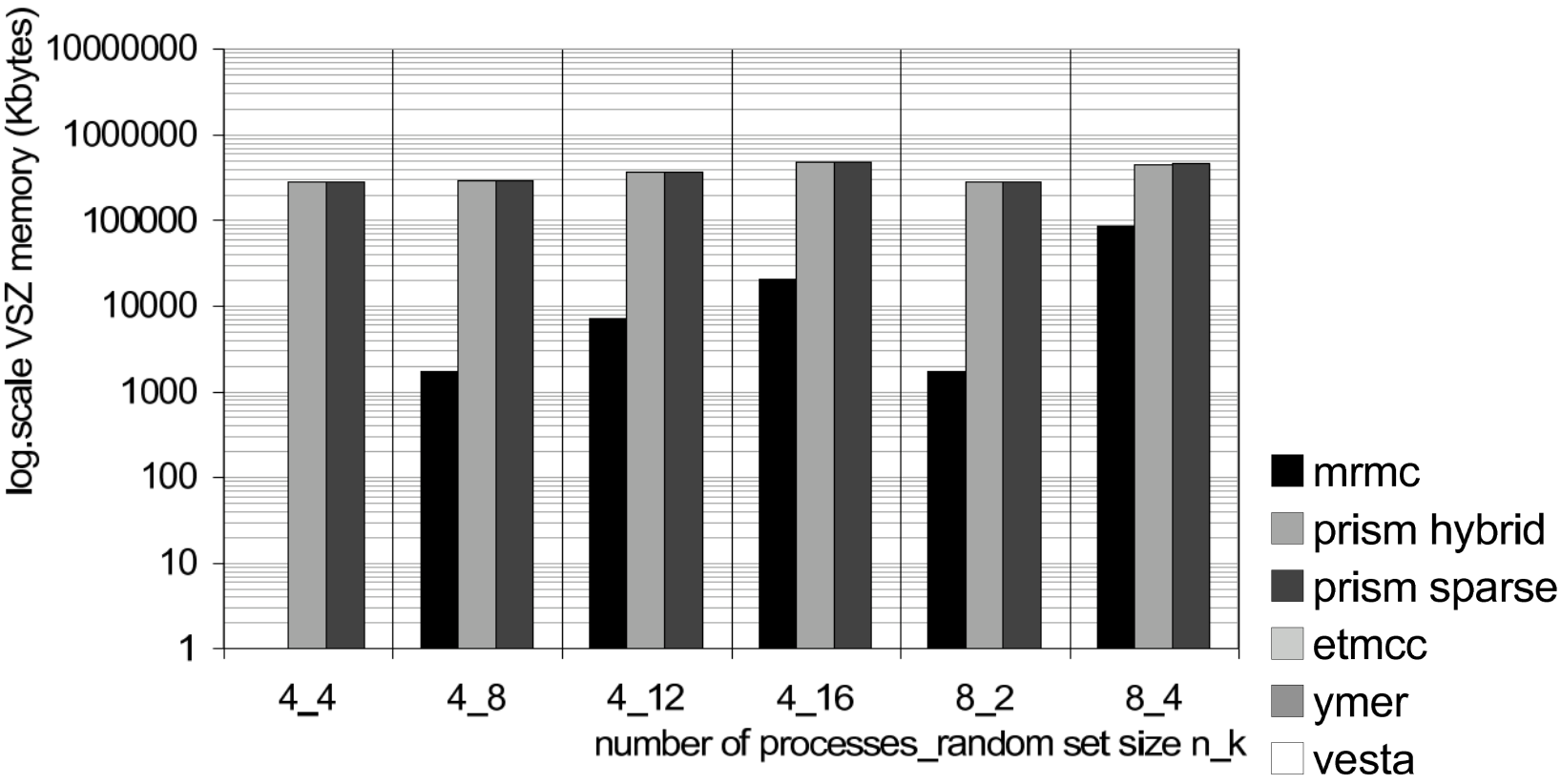
Model Check Time - Synchronous Leader Election

$\mathcal{P}_{\geq 1}(\diamond \texttt{elected})$, or `P{>=1}[tt U elected]`.



Peak Memory - Synchronous Leader Election

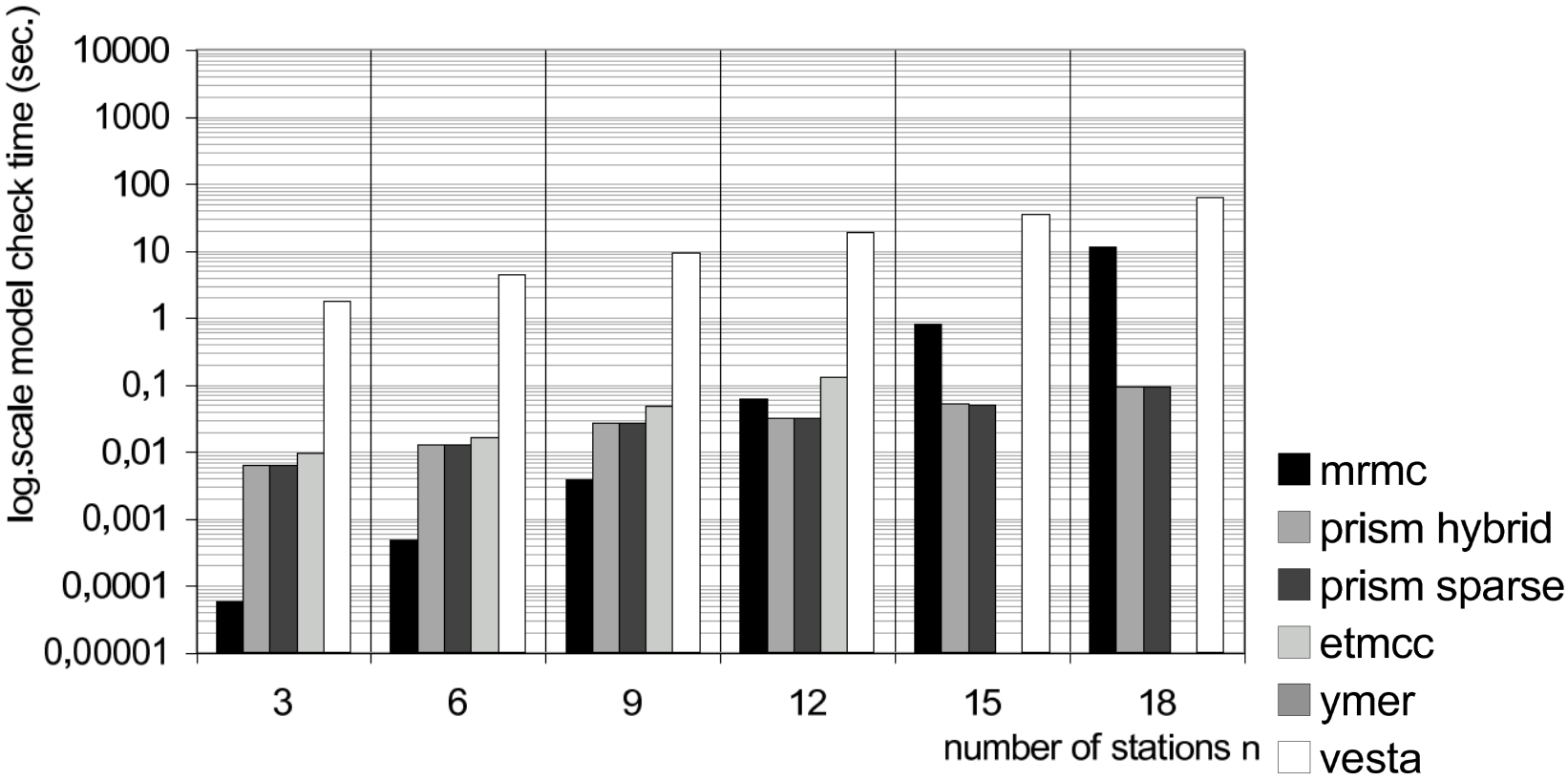
$\mathcal{P}_{\geq 1}(\diamond \texttt{elected})$



Model Check Time - Cyclic Server Polling System

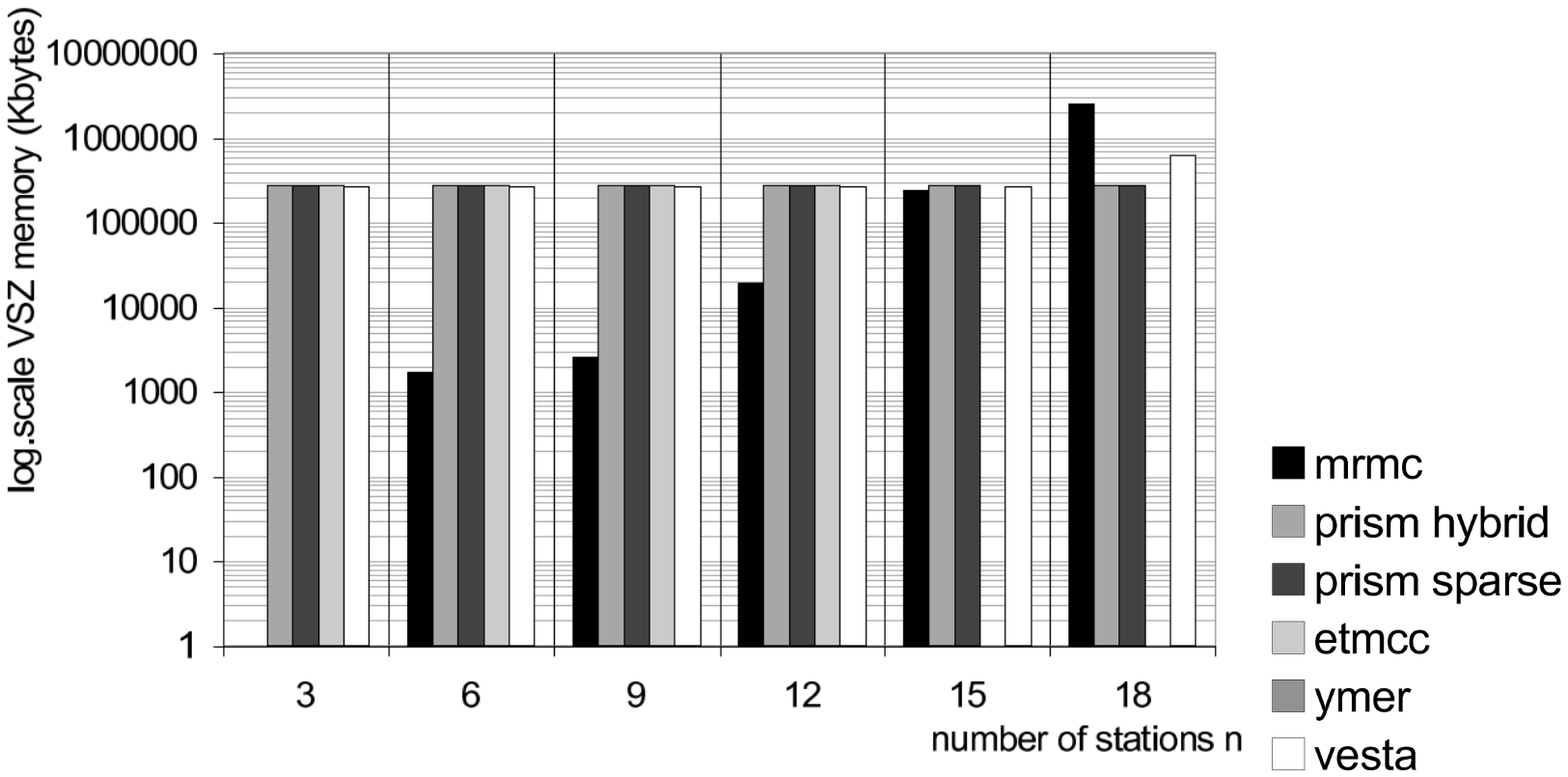
$bussy \Rightarrow \mathcal{P}_{\geq 1}(\diamond \texttt{poll})$

or `!bussy || (bussy && P{<=1}[tt U poll])`



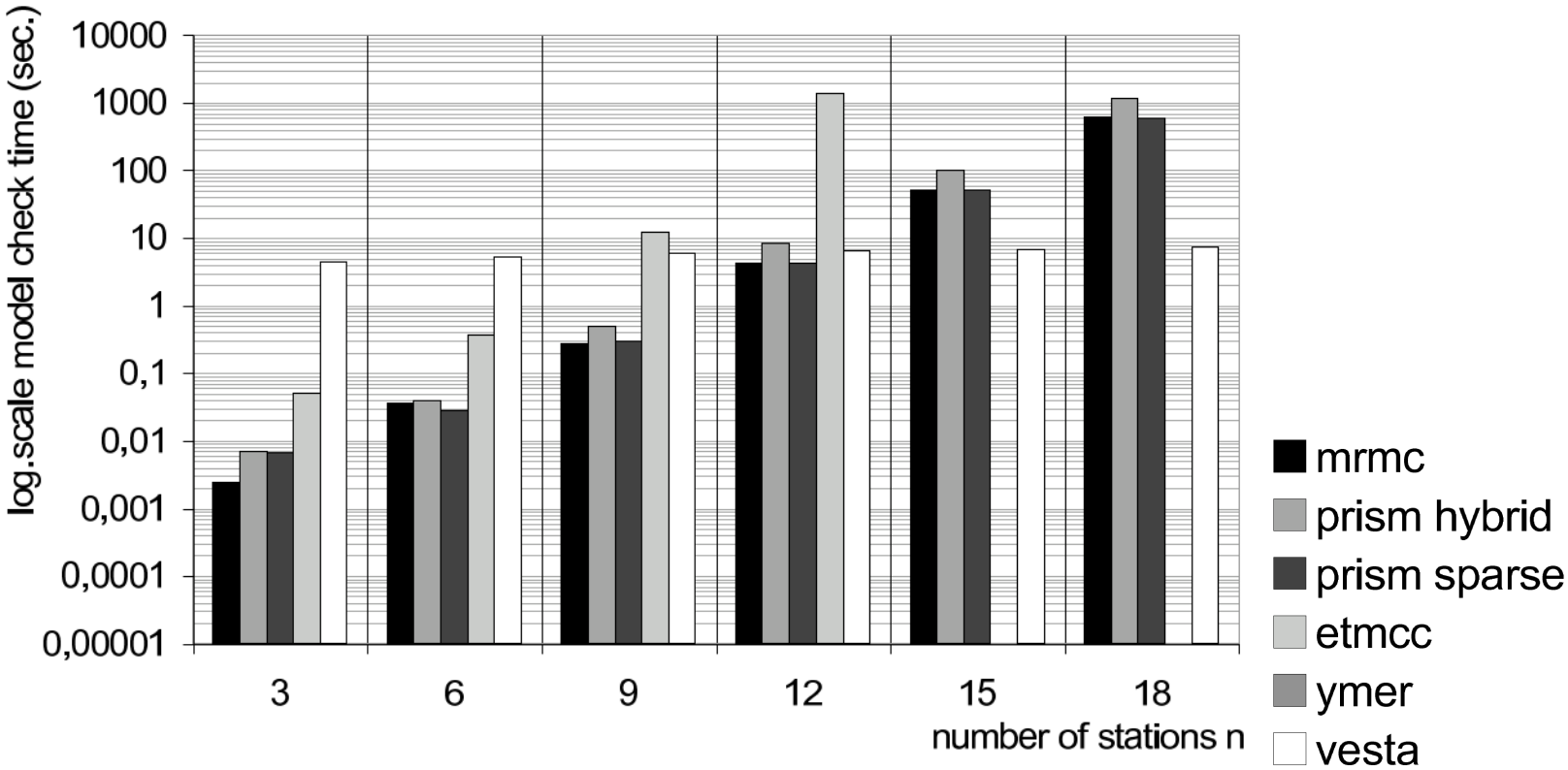
Peak Memory - Cyclic Server Polling System

$bussy \Rightarrow \mathcal{P}_{\geq 1}(\diamond \texttt{poll})$



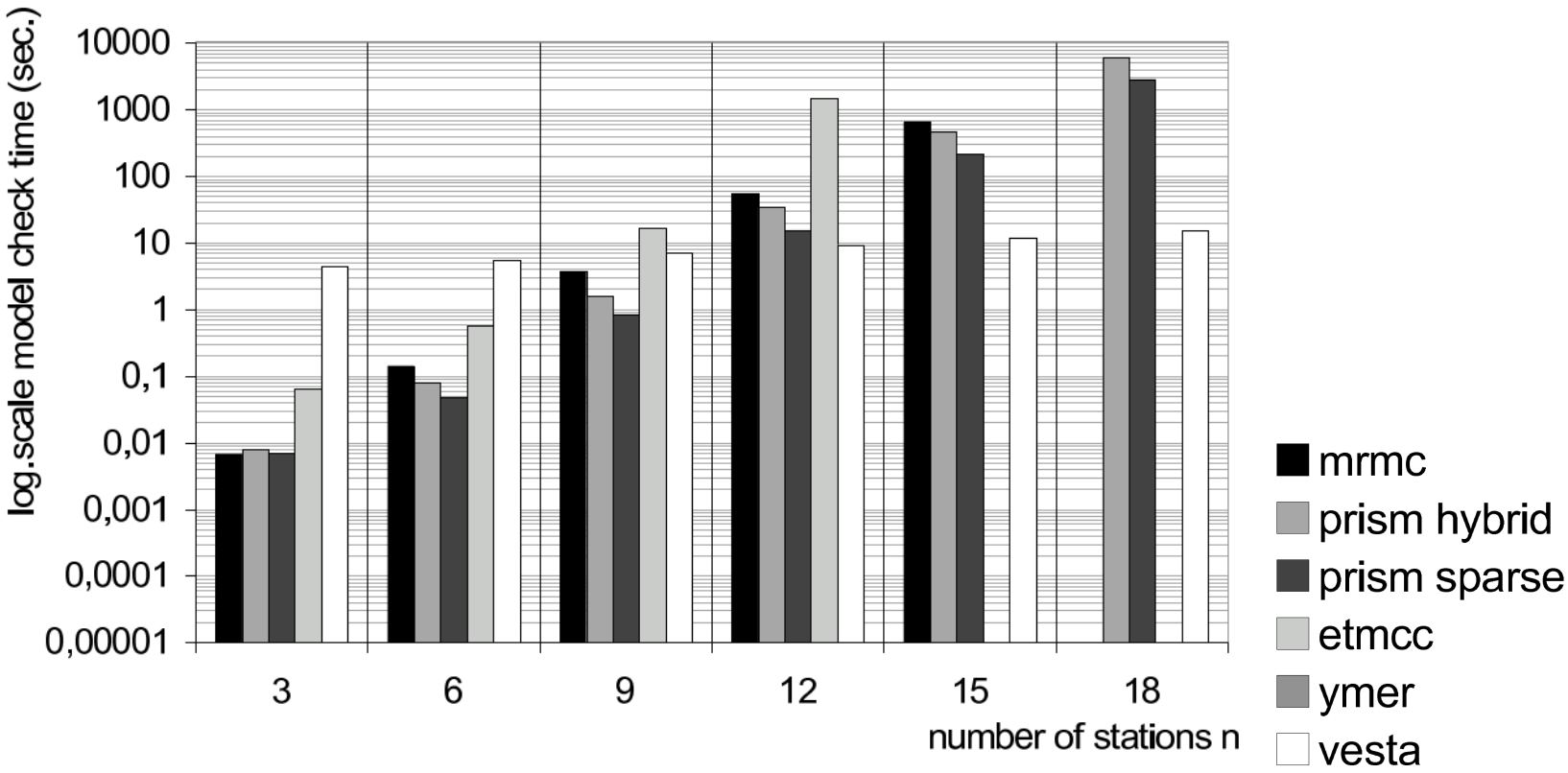
Model Check Time - Cyclic Server Polling System

$$\mathit{bussy} \Rightarrow \mathcal{P}_{\geq 0.5}(\diamond^{\leq t}\mathtt{poll}),\ t = 5$$



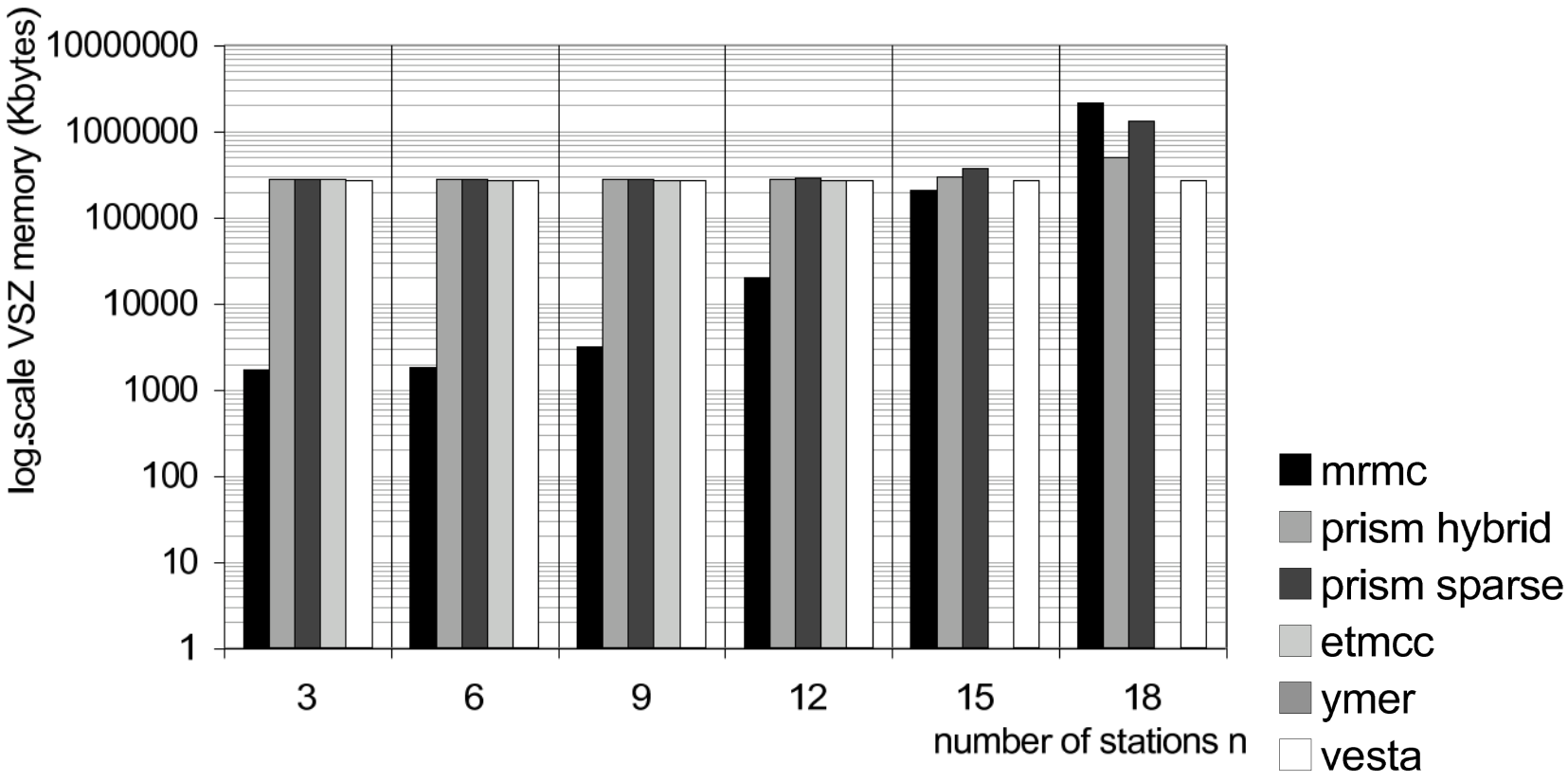
Model Check Time - Cyclic Server Polling System

$$bussy \Rightarrow \mathcal{P}_{\geq 0.5}(\diamond^{\leq t}\texttt{poll}),\ t = 80$$



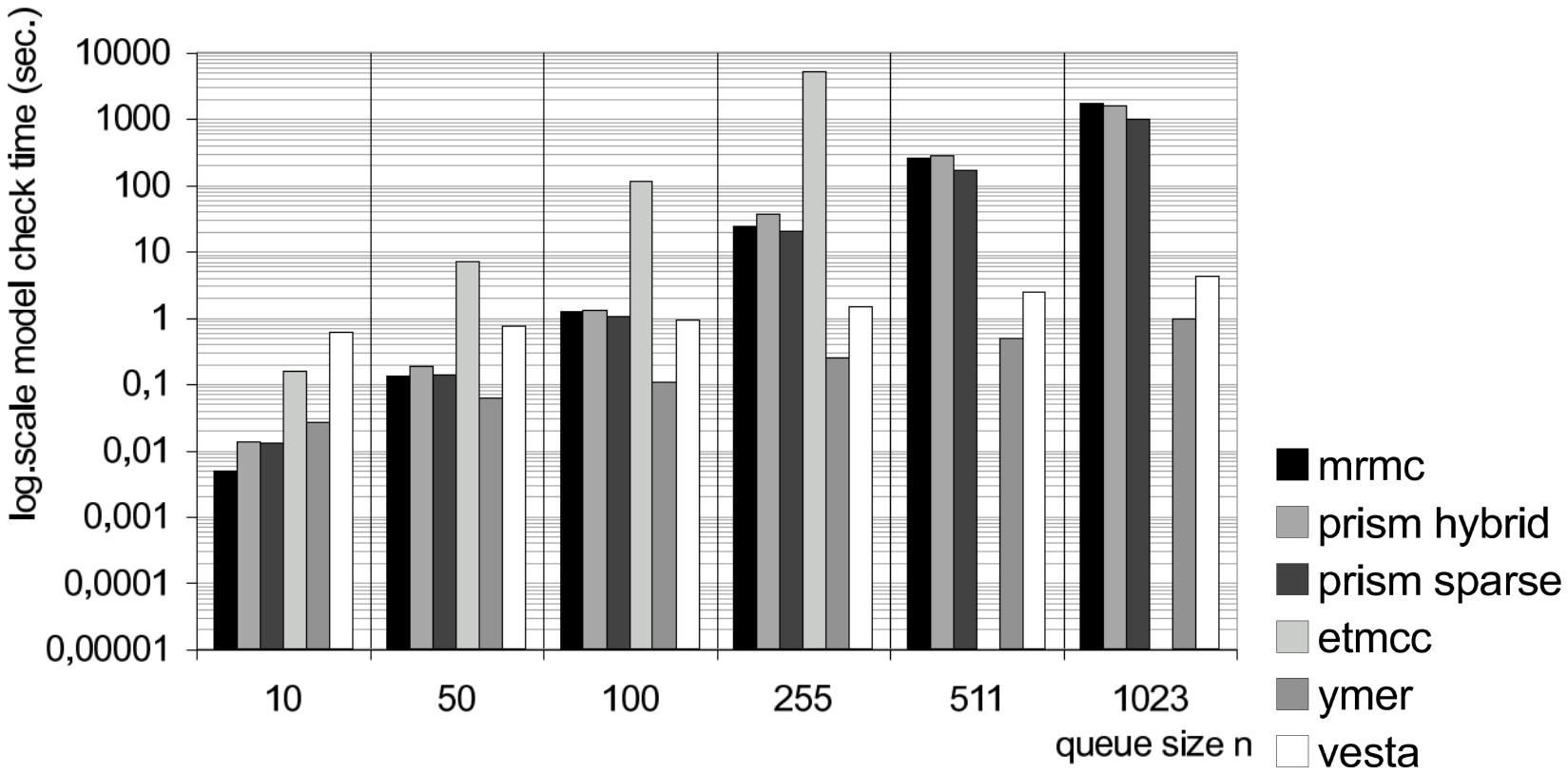
Peak Memory - Cyclic Server Polling System

$$bussy \Rightarrow \mathcal{P}_{\geq 0.5}(\diamond^{\leq t} \mathtt{poll})$$



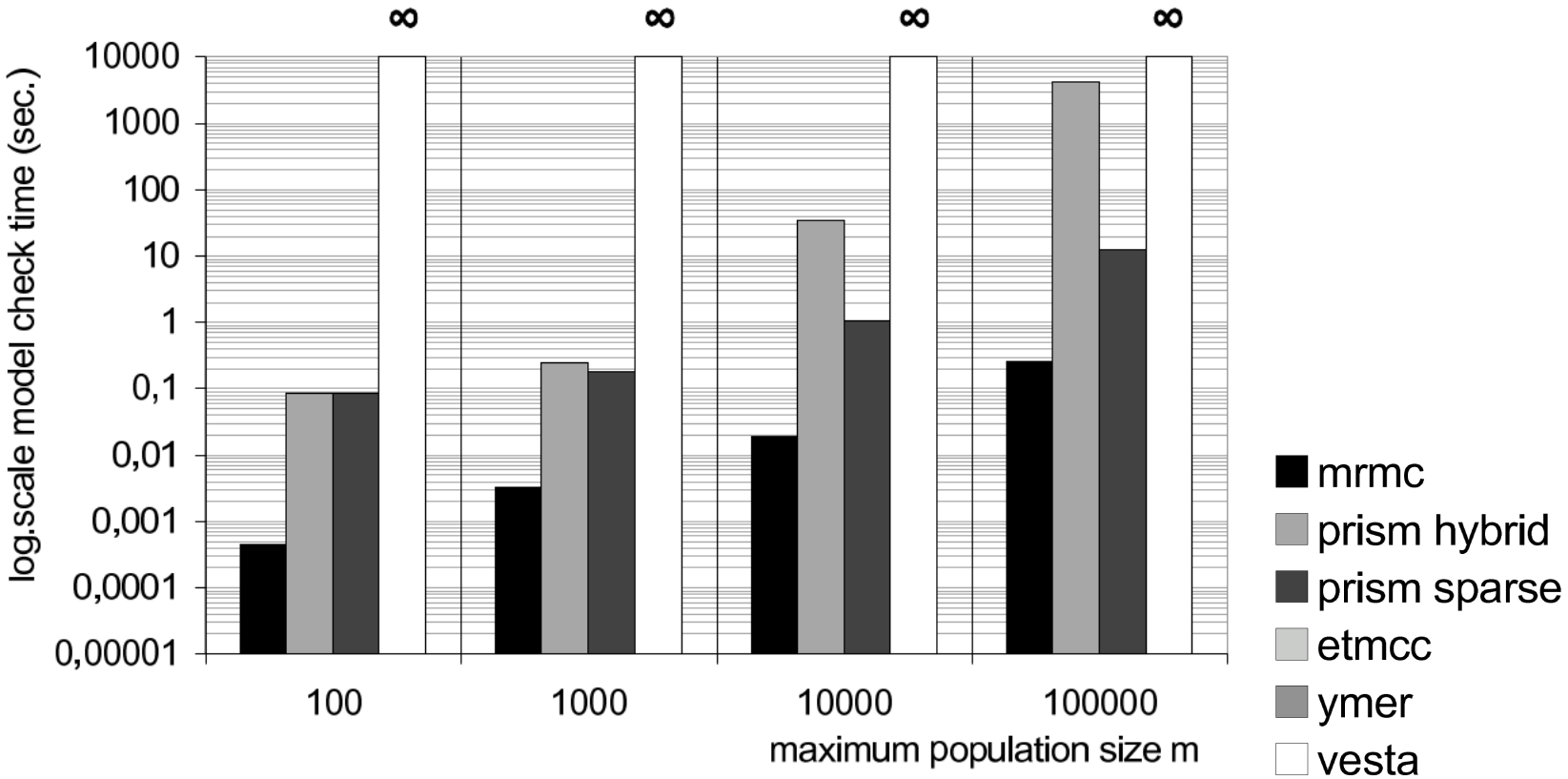
Model Checking Time - Tandem Queueing Network

$\mathcal{P}_{\leq 0.5}(\diamond^{\leq 2}\texttt{full})$



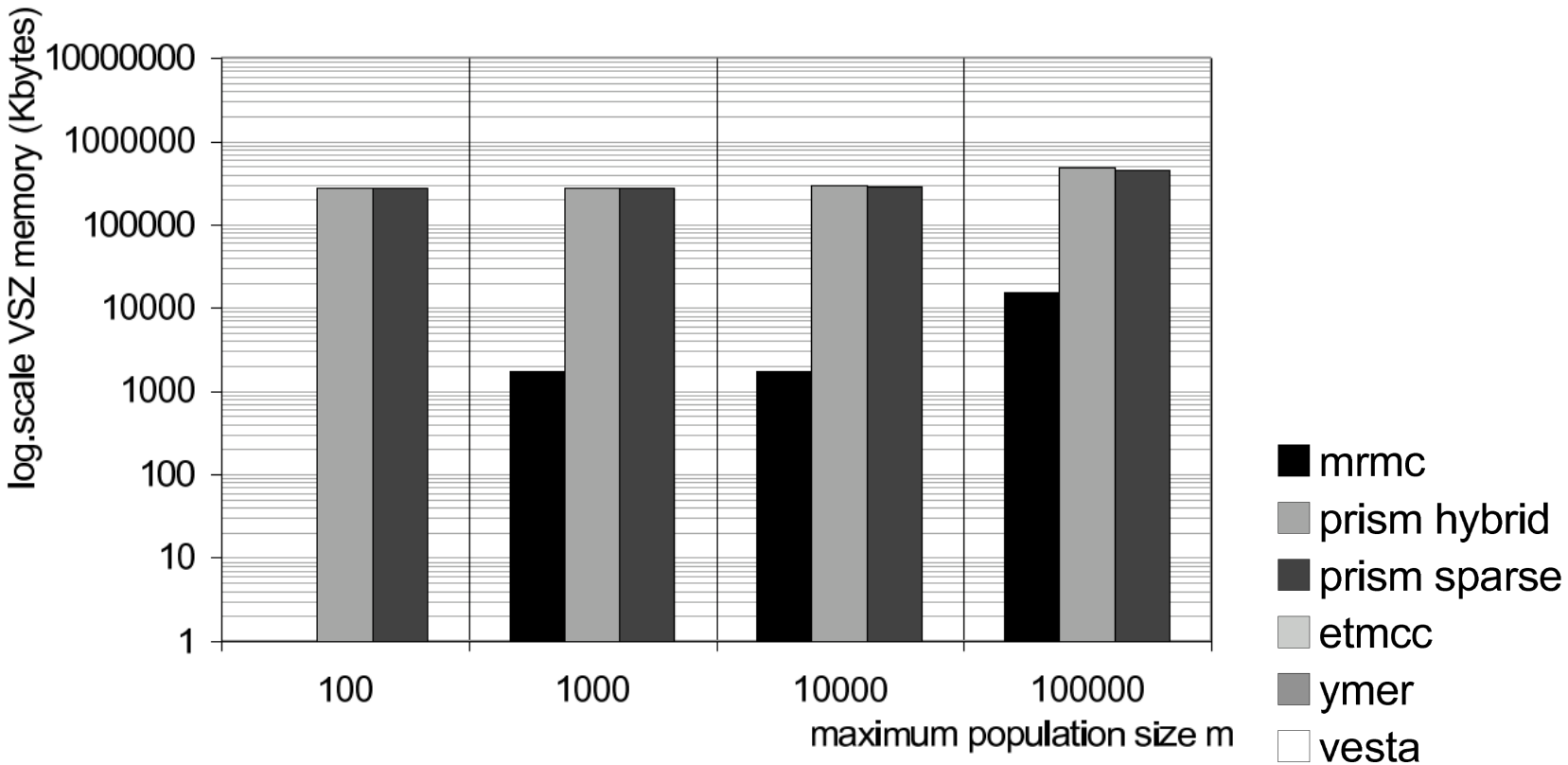
Model Checking Time - Birdth-Death Process

$$\mathcal{P}_{\geq 0.8}(\mathcal{P}_{\geq 0.9}(\diamond^{\leq 100}(n = 70))\mathcal{U}(n = 50))$$



Peak Memory - Birdth-Death Process

$$\mathcal{P}_{\geq 0.8}(\mathcal{P}_{\geq 0.9}(\diamond^{\leq 100}(n = 70))\mathcal{U}(n = 50))$$



Benchmark Table

speed	E⊢MC^2	MRMC	PRISM^S	PRISM^H	YMER	VESTA
steady state	−	++	+	0/+ [a]	N/A	N/A
bounded until	−	+ [b]	+/++	0/+ [a]	++	+
unbounded until	−	+ [b]	+/++	+/++ [a]	N/A	−/0
nested	−	++	+	0/+ [a]	N/A [c]	−− [d]

[a] The time heavily depends on the MTBDD size.
[b] MRMC was faster in most cases, PRISM^S on larger models.
[c] The property contained operators not supported by YMER.
[d] Based on one property, for which VESTA did not terminate.

memory	E⊢MC^2	MRMC	PRISM^S	PRISM^H	YMER	VESTA
steady state	−	+ [a]	+	+/++ [a] [b]	N/A	N/A
bounded until	−	+ [a]	+	+/++ [a] [b]	++	+ [c]
unbounded until	−	+ [a]	+/++	+/++ [a] [b]	N/A	0/+ [c]
nested	−	+ [a]	+	+/++ [a] [b]	N/A	N/A [d]

[a] MRMC used least memory in most cases. For larger models PRISM^S was between MRMC and PRISM^H, and PRISM^H was the best.
[b] The MTBDD size varied much with the case study.
[c] Fairly constant; inefficient for small models, efficient for large ones.
[d] Based on one property, for which VESTA did not terminate.

Contribution

- Benchmarks for comparing tools for verification of Markov chains.

Summary of HFFYPMC

- In some tests MRMC has the worst time complexity.
- Don't code model checkers in Java. (ETMCC is no good).
- The statistical model checker YMER is fast and uses almost constant memory. VESTA (also statistical) varies a lot in model checking time, but also has almost constant memory. (Statistical tools will be wrong from time to time).
- Everything should be better coupled with SPRING due to its user friendliness.

Next Section

Conclusion

MRMC:

- Good back-end
- Needs support for Markov decision processes
- PCTL seems too powerful for Markov chains.

HFFYPMC:

- Low time measures, we want to save hours, not seconds!
- MRMC appeared to have a bad time complexity.

. . . THE END.