Experiences with Cluster-Based Model Checking of Stochastic Systems

Boudewijn R. Haverkort
Department of Computer Science
University of Twente, Enschede, the Netherlands

Alexander Bell
Department of Mathematics,
University of Twente, Enschede, the Netherlands

http://dacs.cs.utwente.nl/

Overview

- Background
- Cases and machinery
- Generation
- Solution
- Concluding remarks

Background: High-level specifications

- man-made discrete systems can be described well using high-level specification languages:
  – communication protocols, distributed systems (process algebras)
  – computer systems, manufacturing systems (Petri nets)
- not all system properties can be verified directly from the high-level specification
- many properties can only be verified using the large underlying STS
- derivation of STS is a time-consuming task
- important aspect of "model checking"

Background: Adding (stochastic) time

- model specification is enhanced with stochastic timing information
- underlying STS can be interpreted as a *continuous-time Markov chain* $\cdots$ which can be solved numerically
- various model specification techniques can be imagined:
  e.g., stochastic process algebras, or *stochastic Petri nets*
- variety of logic-based property specifications possible

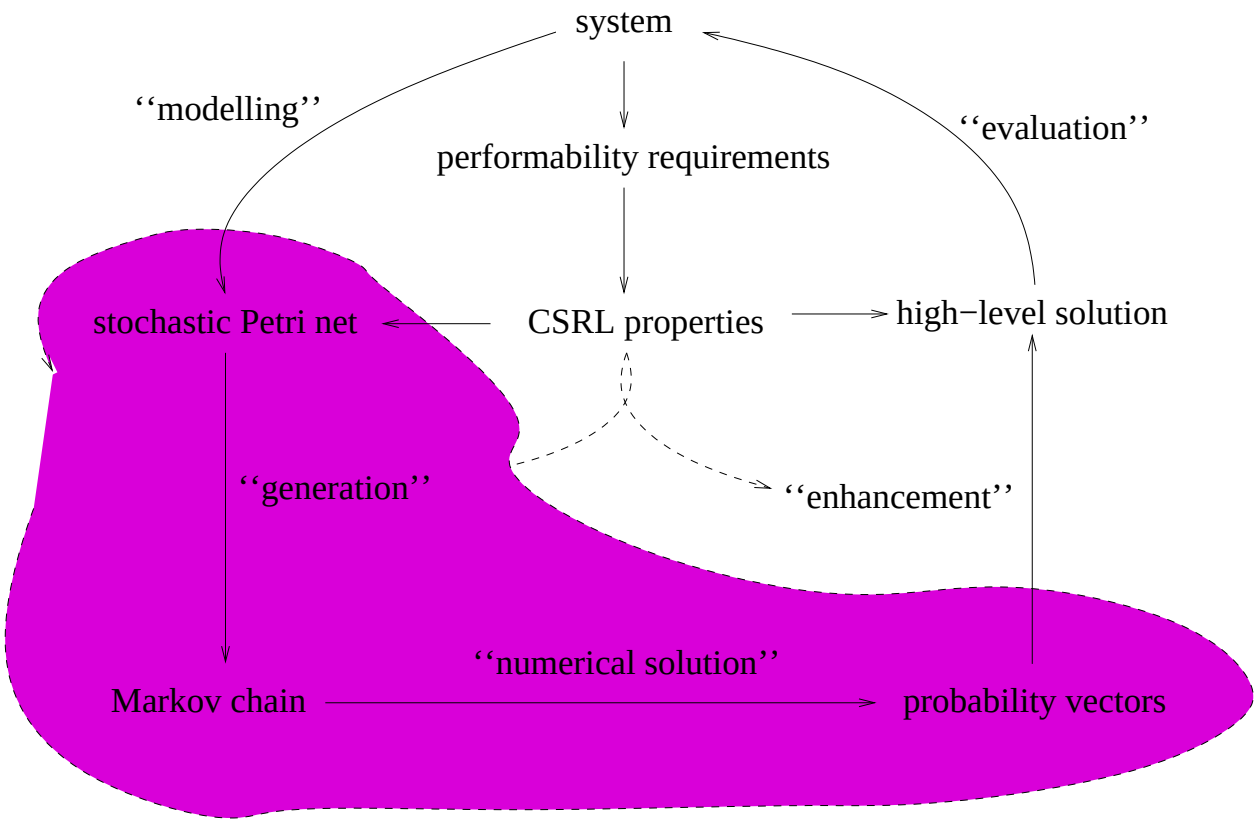
Hence, we combine...

- aspects of CTL-type model checking
- with aspects of numerical analysis of Markov chains

$\Rightarrow$ **CSL and CSRL model checking of Markovian models**

- try to combine best techniques of both worlds!
- in any case: computational and storage requirements are enormous!

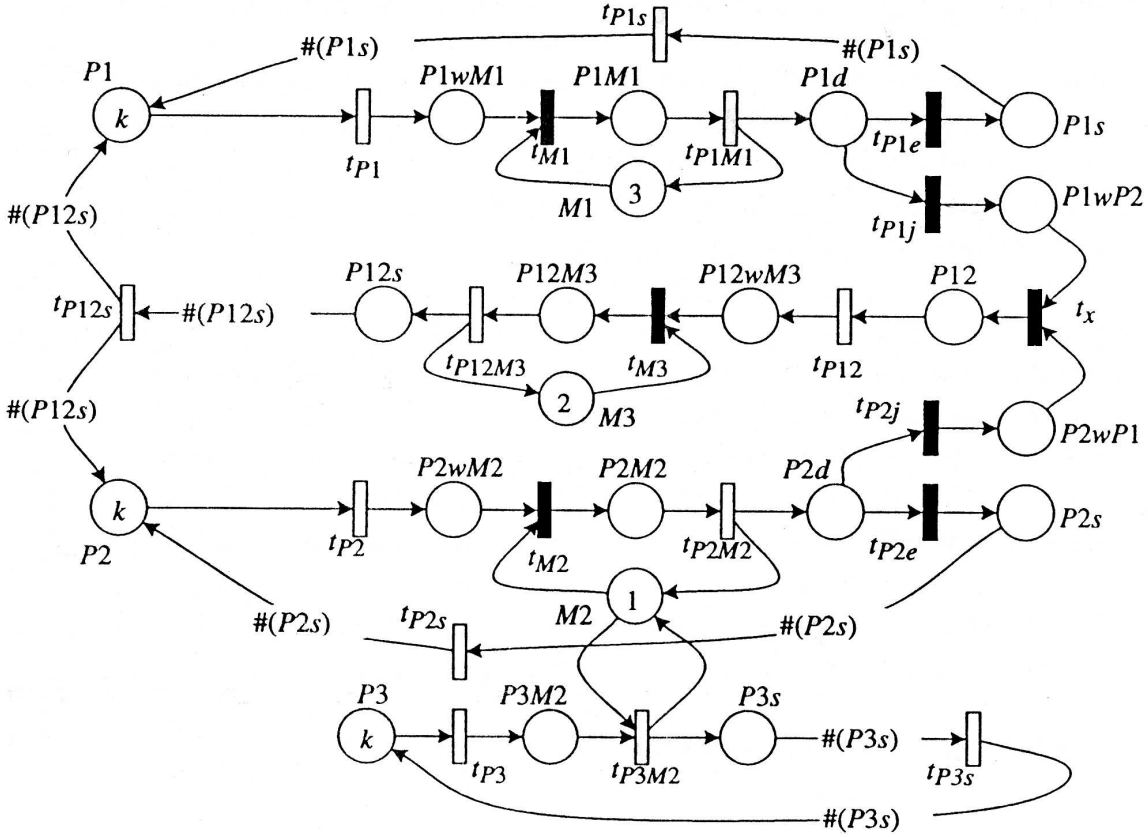
Modelling & evaluation cycle



Overview

- √ Background
- Cases and machinery
- Generation
- Solution
- Concluding remarks

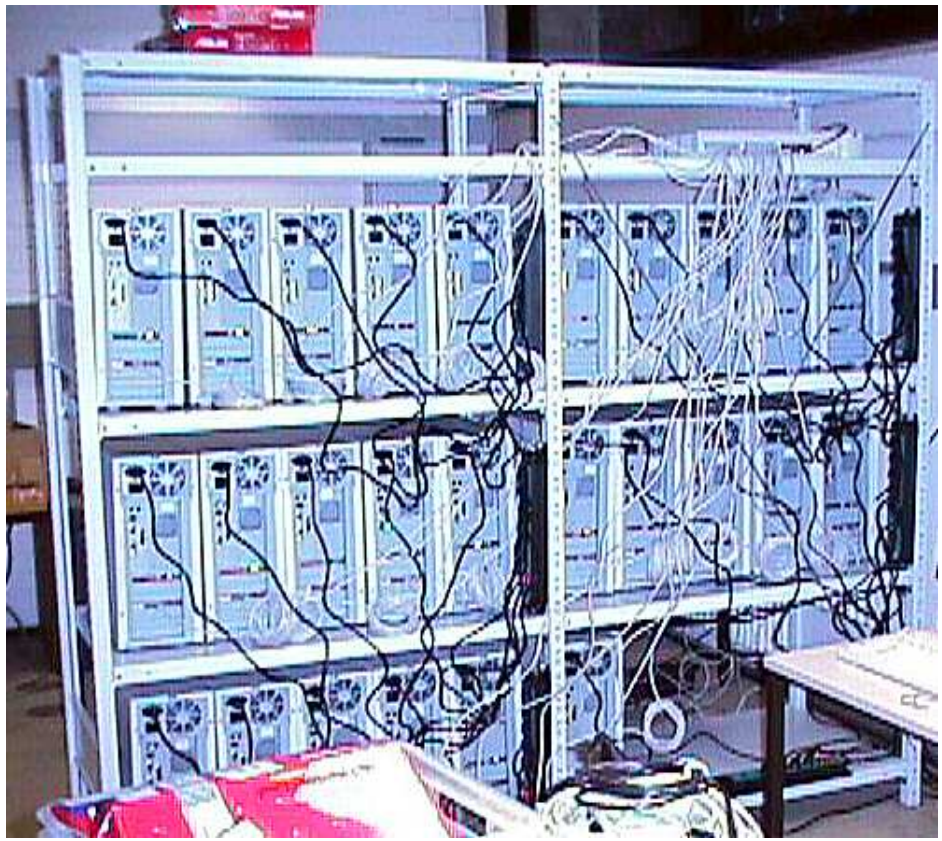
Cases and Machinery: the FMS case (I)



Cases and Machinery: the FMS case (II)

pallets k	states n	arcs$a$	a/n	a/n^2
1	54	155	2.87	5.31e-02
3	6 520	37 394	5.73	8.80e-04
5	152 712	1 111 482	7.28	4.77e-05
7	1 639 440	13 552 968	8.27	5.04e-06
9	11 058 190	99 075 405	8.96	8.10e-07
11	54 682 992	518 030 370	9.47	1.73e-07
13	216 427 680	2 611 411 257	12.07	5.57e-08
15	724 284 864	9 134 355 680	12.61	1.74e-08

Cases and Machinery: Cluster at RWTH



Cases and Machinery: From DAS-2 to DAS-3

Overview

- √ Background
- √ Cases and machinery
- Generation
- Solution
- Concluding remarks

STS generation: Three possibilities

- explicitly, serially
- explicitly, distributed
- implicitly

Serial, explicit STS generation

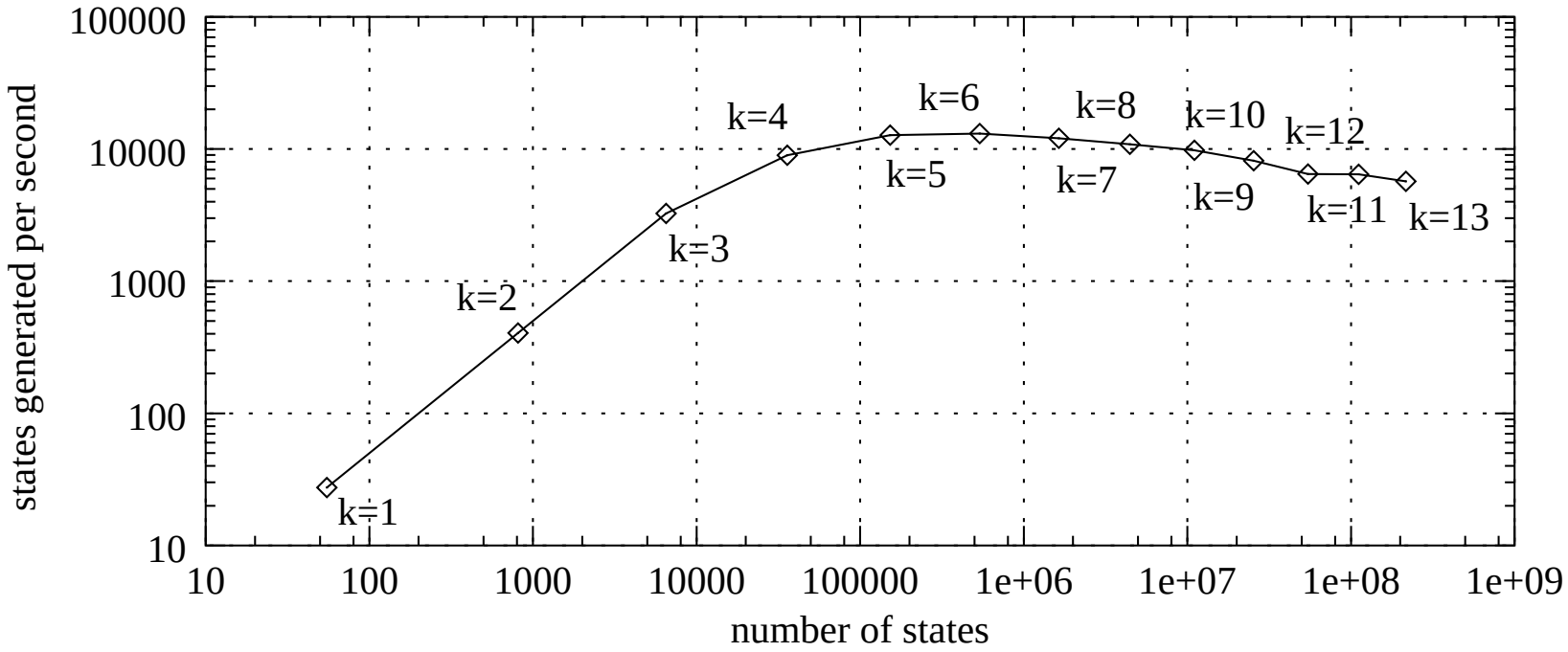
- standard tree-based search algorithm

- key issues:
  - 1: data structure for state space
  - 2: data structure for intermediate states
  - 3: next state computation (enabled(s)?)
  - 4: does state s exist?

- choices:
  - 1: hash table: open addressing with double hashing
  - 2: stack with top in main memory, all of body on disk
  - 3, 4: depends strongly on high-level model (SPNs: good!)

- transition relation directly stored on disk

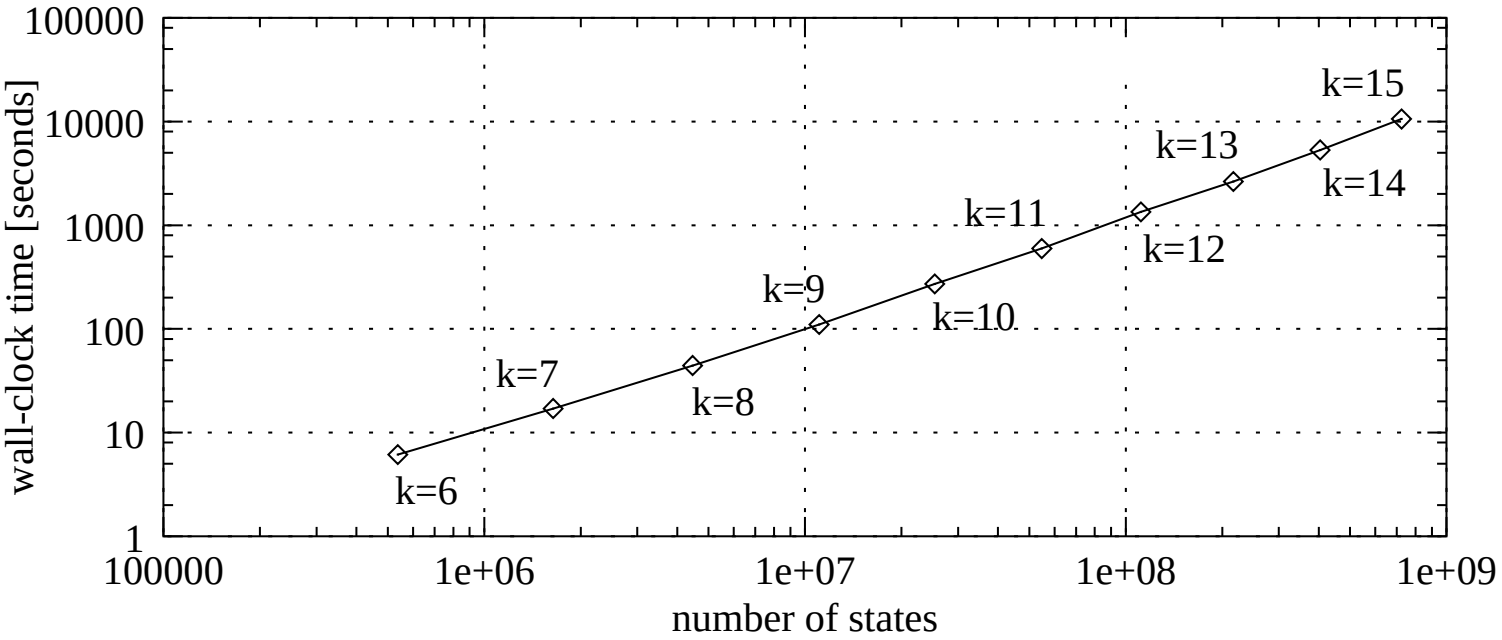
Serial STS generation speed for FMS(k)



Distributed, explicit STS generation

- use an extra hash function to map states onto processors (nodes)
- find good:
  – state balance: equal number of states per node
  – while avoiding cross-arcs = minimizing communication
- good, balanced solutions do exist
- divides state space and transition relation over all available nodes
- shows very good speed-ups

26-node distributed explicit STS generation for FMS(k)



Implicit generation

- use of BDDs, MTBDDs or MxDs (matrix diagrams)
- extremely fast even on single nodes, for many millions of states
- the next step, i.e., model checking or Markov chain solution, is much more computationally intensive

$\Rightarrow$ no work on parallellisation required here, unless state space reduction is performed on-the-fly

- better concentrate on the true memory, communication and computation bottlenecks!

Overview

- √ Background
- √ Cases and machinery
- √ Generation
- • Solution
- • Concluding remarks

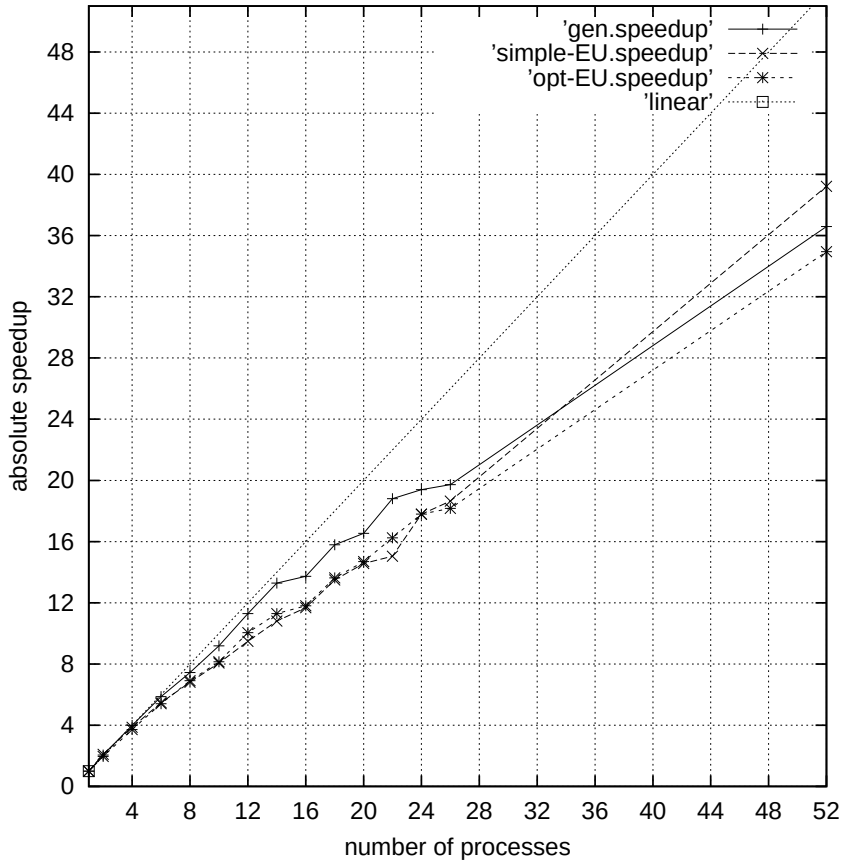
What does "solution" mean?

- CTL model checking of Markov chain interpreted as STS
- CSL and CSRL model checking, meaning, the solution of large equation systems

Distributed explicit CTL model checking of Petri nets

- generalised the algorithms for STS generation
- for all the operators of CTL (E[X.], E[.U.], A[.U.])
- do need the state space in some form readily available
- but recompute the transition relation on the fly
  (both backward and forward, depending on the CTL query)
- found very good speed-ups and efficiencies

Kanban model with $11\,261\,376$ states; S-EU and O-EU



The other type of STS "solution"

- the underlying STS is a Markov chain now, from which we want to compute probabilistic information
- the CSL semantics contains two important operators:
  – steady-state operator: requires steady-state probabilities
  – probability operator: requires transient-state probabilities
- both operators require series of matrix-vector multiplications to be performed
- hence, doing MVMs quickly, serially or distributed, is the key issue!

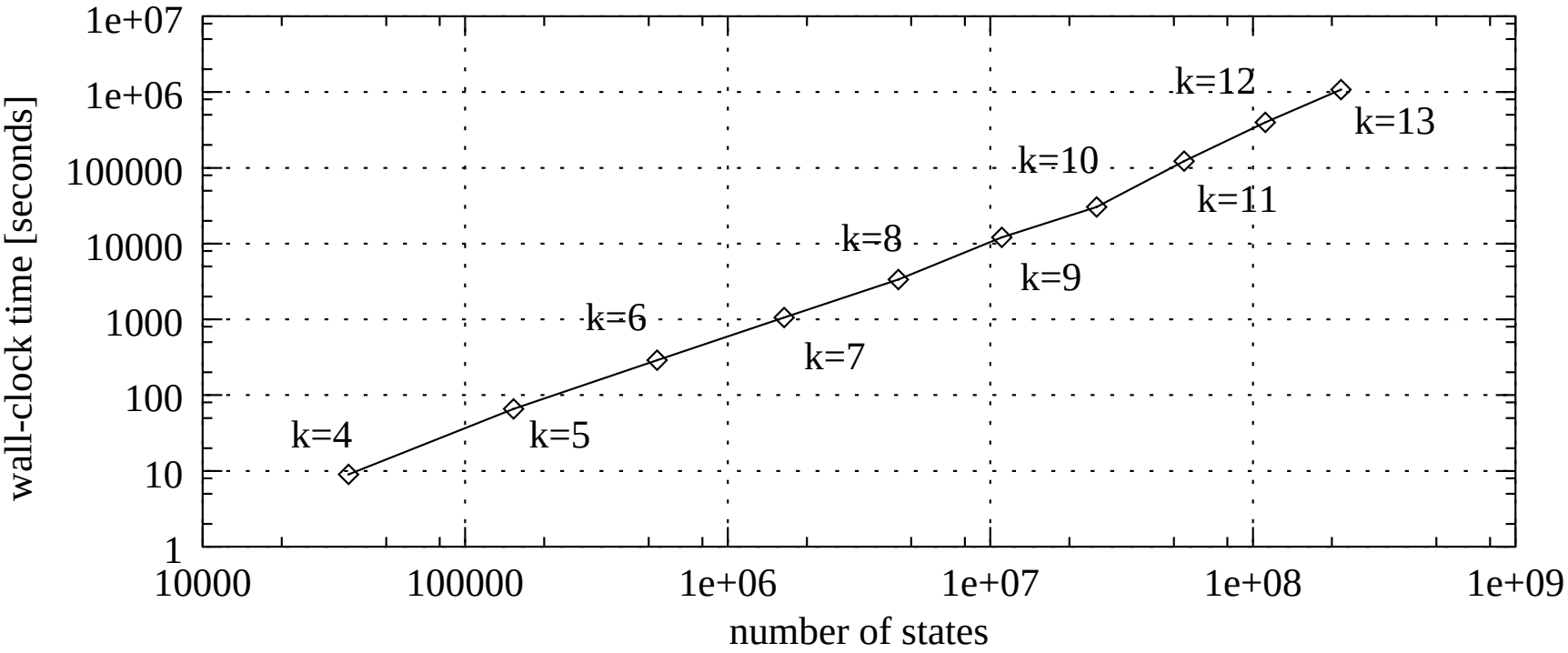
Steady-state solution (I)

- solution of $\underline{\pi} \cdot \mathbf{Q} = \underline{1}$, with $\sum_i \pi_i = 1$
- only iterative methods can be used: $\underline{x}^{(i+1)} := \underline{x}^{(i)} \cdot \mathbf{Q}^*$
  - Jacobi: simple, slow method, 2 iteration vectors
  - Gauss-Seidel: faster, 1 iteration vector
  - Conjugate Gradient Squared; requires more iteration vectors
- vector $\underline{\pi}$ is **very big** but must be **in-core**, otherwise too slow
- storage of $\underline{\pi}$ is a key problem
- **out-of-core** or implicit matrix $\mathbf{Q}^*$; it does not fit in main memory!

Serial steady-state solution FMS(k)

k	Jacobi				CGS			
	$\epsilon = 10^{-6}$		$\epsilon = 10^{-9}$		$\epsilon = 10^{-6}$		$\epsilon = 10^{-15}$	
	steps	time	steps	time	steps	time	steps	time
1	260	0:00:00	536	0:00:00	32	0:00:00	230	0:00:00
2	212	0:00:00	473	0:00:00	85	0:00:00	517	0:00:00
3	312	0:00:00	682	0:00:01	137	0:00:01	849	0:00:05
4	402	0:00:09	889	0:00:20	181	0:00:09	1465	0:01:13
5	491	0:01:02	1105	0:02:19	236	0:01:06	2424	0:11:40
6	584	0:05:51	1344	0:13:27	255	0:04:50	2202	0:52:12
7	686	0:18:33	1589	0:43:00	309	0:17:39	3981	3:43:53
8	784	1:03:24	1829	2:27:58	338	0:56:00	2905	8:03:47
9	881	3:15:45	2073	7:40:50	392	3:08:03	—	—
10	980	10:30:59	2366	25:23:24	373	8:27:45	—	—
11	1080	48:29:32	2734	122:45:26	363	34:10:40	—	—

Serial steady-state solution FMS; CGS(4–11), J(12–13); $\epsilon = 10^{-6}$



Distributed steady-state solution: basics

- solution of $\underline{\pi} \cdot \mathbf{Q} = \underline{1}$, with $\sum_i \pi_i = 1$
- only iterative methods can be used: $\underline{x}^{(i+1)} := \underline{x}^{(i)} \cdot \mathbf{Q}^*$
  - Jacobi: simple, slow method, 2 iteration vectors, easy to parallellise
  - Gauss-Seidel: faster, 1 iteration vector, very difficult to parallellise
  - CGS: requires more iteration vectors, but good to parallellise
- **out-of-core**: the matrix $\mathbf{Q}^*$ does not fit in main memory, but can be partitioned over many nodes
- solution vector $\underline{\pi}$ can be distributed over many nodes

Distributed steady-state solution: splitting the work

- each processor stores part of $\mathbf{Q}^*$ (disk) and computes part of $\underline{\pi}$ (memory)
- non-local probabilities are explicitly requested (non-blocking send and receive) from other nodes (for 2 nodes):

$$(\underline{\pi}_1, \underline{\pi}_2)^{\text{new}} := (\underline{\pi}_1, \underline{\pi}_2)^{\text{old}} \cdot \begin{pmatrix} \mathbf{Q}^*_{11} & \mathbf{Q}^*_{12} \\ \mathbf{Q}^*_{21} & \mathbf{Q}^*_{22} \end{pmatrix}$$

$$\Rightarrow \underline{\pi}_1^{\text{new}} := \underbrace{\underline{\pi}_1^{\text{old}}}_{\text{local memory}} \cdot \underbrace{\mathbf{Q}^*_{11}}_{\text{local disk}} + \underbrace{\underline{\pi}_2^{\text{old}}}_{\text{remote memory}} \cdot \underbrace{\mathbf{Q}^*_{21}}_{\text{local disk}}$$

- two threads interleave communication and computation
- barrier synchronisation after each iteration

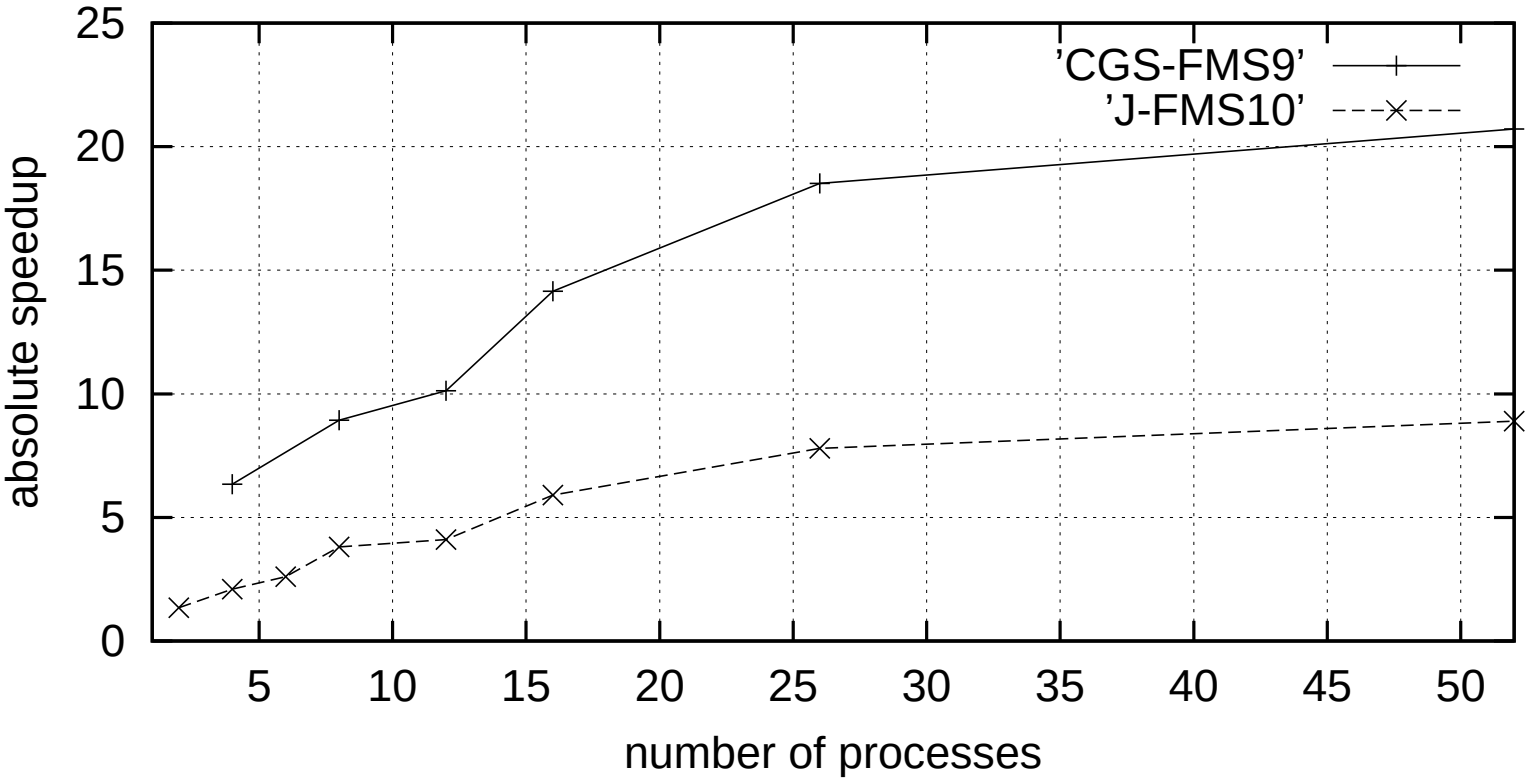
Distributed steady-state solution: I/O overhead

- FMS($k = 15$): $724\,284\,864$ states, $9\,134\,355\,680$ arcs
- solution method is really I/O-bound
- typical I/O-*overhead* per iteration:
  - 1200 MB read from local disk
  - 800 MB sent to other nodes
  - 800 MB received from other nodes
- 1 iteration $\approx$ 7 minutes; around 2000 required!
- for FMS($k = 10$): $25\,397\,658$ states: 7 seconds/iteration

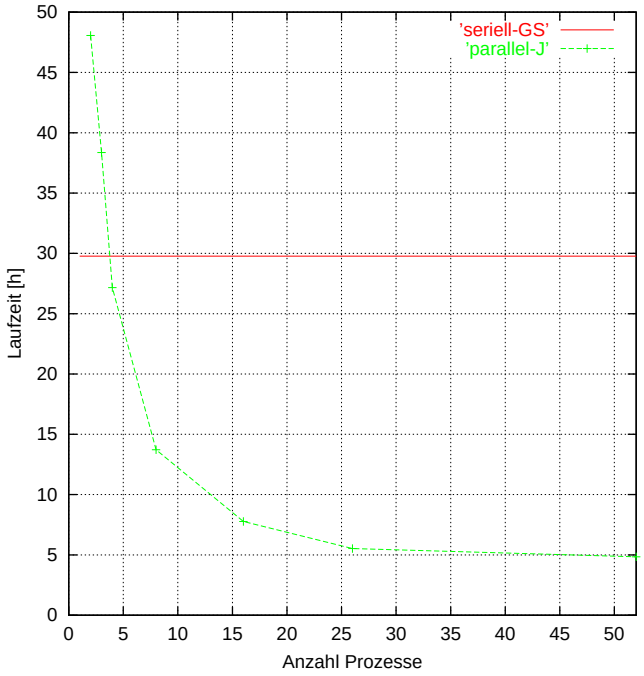
FMS distributed steady-state solution, $\epsilon = 10^{-8}$, $N = 26$

k	Jacobi				CGS			
	$\epsilon = 10^{-6}$		$\epsilon = 10^{-9}$		$\epsilon = 10^{-6}$		$\epsilon = 10^{-15}$	
	steps	time	steps	time	steps	time	steps	time
6	886	0:06:24	1507	0:10:53	371	0:05:22	893	0:12:55
7	1034	0:21:06	1783	0:36:23	150	0:07:09	1253	0:51:28
8	1181	0:43:56	1977	1:13:34	503	0:40:19	1469	1:57:44
9	1336	2:31:54	2208	4:11:03	863	2:52:59	1623	5:25:20
10	1492	7:36:36	2466	12:34:41	903	9:25:31	3559	37:08:53
11	1652	18:21:37	2722	30:15:08	665	15:04:44	3124	70:50:13
12	1818	47:59:51	2979	78:38:59	1219	65:00:17	2184	116:27:52
13	1990	68:53:55	3257	112:45:56	857	90:58:09	1987	210:55:02
14	2169	141:08:33	4448	289:26:34	325	72:36:53	—	—
15	2335	292:16:37	—	—	—	—	—	—

Distributed steady-state solution: speed-up



Kanban(7) distributed steady-state solution at $\epsilon = 10^{-7}$



[serial Gauss-Seidel: 624 iterations; distributed Jacobi: 1244 iterations]

Distributed steady-state solution: cross-arc impact

- Kanban model ($k = 7$): $41\,644\,800$ states, $450\,455\,040$ arcs
- with cross-arc percentage $X = 50\%$:

processors N	52	26	12	8
time/iteration	20	22	34	45

- with $N = 52$ processors:

cross-arcs [%]	98	94	70	51	30
time/iteration	34	28	23	20	14

Distributed steady-state solution: processor impact

- solution time and cross-arc percentage as function of the number of employed processors:

processors N	52	26	16	8	4	3	2
cross-arcs [%]	30	28.4	28.5	26.9	21.9	19	14.6
time/iteration	14	16	22.5	39.7	78.6	111	139.1

- serial solution with Gauss-Seidel: 171.5 seconds per iteration

- Jacobi requires more than 512 MB RAM; no serial solution available

Overview

- √ Background
- √ Cases and machinery
- √ Generation
- √ Solution
- • Concluding remarks

Summary

- completed first project on distributed evaluation of CTMCs from SPNs
- excellent performance for serial and distributed generation
- also excellent performance for distributed CTL model checking
- good performance for the distributed numerical solution, however, this remains the bottleneck
- main memory storage of solution vector(s) is the key issue
- further study of work-division strategy in solution speed

GRID-Based Challenges

- extension toward GRID-based model checking for CSRL:
  more complex algorithms and data structures (not just MVMs)
- three "levels" of access time: in node, in cluster, in grid
- complicated trade-off between communication (cross-arcs) and convergence speed
- combination with other techniques, e.g., for on-th-fly state-space reduction, bisimulations, etc.

Literature

- B.R. Haverkort, A. Bell, H.C. Bohnenkamp, "On the Efficient Sequential and Distributed Generation of Very Large Markov Chains from Stochastic Petri Nets", *Proc. IEEE PNPM 1999*, pp.12–21, Zaragosa, Spain.
- A. Bell, B.R. Haverkort, "Serial and parallel out-of-core solution of linear systems arising from generalised stochastic Petri net models", in: *Proc. HPC 2001*, pp.242–247, Seattle, USA.
- A. Bell, B.R. Haverkort, "Distributed CTL model checking of Petri net specifications", *Electronic Notes in Theoretical Computer Science* **68**(4), 2002.
- A. Bell, B.R. Haverkort, "Distributed disk-based algorithms for model checking very large Markov chains", *submitted for publication*, 2005.
- A. Bell, Ph.D. thesis, *Distributed Evaluation of Stochastic Petri Nets*, RWTH Aachen, 2003.