Distributed Analysis with μCRL: A Compendium of Case Studies[*]

Stefan Blom[2], Jens R. Calamé[1], Bert Lisser[1], Simona Orzan[3], Jun Pang[4],
Jaco van de Pol[1,3], Mohammad Torabi Dashti[1], and Anton J. Wijs[1]

[1] CWI, Amsterdam, The Netherlands
[2] Institut für Informatik, Universität Innsbruck, Austria
[3] TU/e, Eindhoven, The Netherlands
[4] Carl von Ossietzky Universität, Oldenburg, Germany [***]

Abstract. Models in process algebra with abstract data types can be analysed by state space generation and reduction tools. The μCRL toolset implements a suite of distributed verification tools for clusters of workstations. We illustrate their application to large case studies from a wide range of application areas, such as functional analysis, scheduling, security analysis, test case generation and game solving.

1 Introduction

The μCRL toolset (`www.cwi.nl/~mcrl`, [2,3]) is equipped with several tools to analyse models of industrial systems, comparable to CADP [8], Spin [10], Uppaal [1] and Murϕ [7]. Most techniques rely on explicit state space generation. In order to overcome memory problems of a single machine, we have constructed distributed implementations of the μCRL tools. With this paper we illustrate that these distributed analysis tools are essential in a wide range of application areas. In particular, we discuss applications in functional analysis, scheduling, test generation, security analysis, and game solving.

Before doing so, we shortly mention the distributed tools, which all run on a cluster of workstations. First of all, there is the *state space generator* [11]. Besides generating all possible behaviour of a μCRL-model, it can check for simple properties, e.g. deadlock freeness. One manager and n clients perform a distributed breadth-first exploration, where a hash function is used to assign states to the clients. This exploration is done level by level simultaneously on all clients, whereby the clients, which have finished their part of the task, communicate destination states to the other clients. The manager synchronises the clients, hence enforcing the breadth-first character of the exploration.

A GUI is provided with the toolset to monitor running jobs. A running job can be killed at any time, to be restarted later on. Some *minimisation tools* [4] reduce a state space modulo strong or branching bisimulation. There is also a

---

[***] `Stefan.Blom@uibk.ac.at, {jens.calame,bertl,vdpol,dashti,a.j.wijs}@cwi.nl, s.m.orzan@tue.nl, jun.pang@informatik.uni-oldenburg.de`

distributed SCC contraction tool [12], which eliminates for instance all τ-cycles. Finally, it is possible to deal with priced reachability problems. We implemented minimal-cost search (where the search order is determined by the costs associated with actions), and beam search (which uses heuristics to guide the search) [14,15]. In the latter case, state space generation is not exhaustive.

2 Applications of Distributed Analysis

Functional Analysis – A Cache Coherence Protocol. Jackal is a fine-grained distributed shared memory implementation of the Java programming language. It aims to implement Java's memory model and allows multithreaded programs to run unmodified on a distributed memory system. It employs a self-invalidation based, multiple-writer cache coherence protocol, which allows processors to cache a region created on another processor.

A μCRL specification of the protocol was extracted from an informal (C-like language) description. It contains parallel processes for components such as threads, processors, buffers, and regions. These components interact with each other via message communications. Our analysis [13] has revealed two unanticipated errors in the implementation, which were confirmed and corrected by the developers of Jackal. The μCRL distributed state space generation tool has played a central role for this case study. It was used to generate state spaces for several large instances of the protocol. One of the two errors found by analysing them with CADP can only be detected on these instances.

Test Case Generation – Common Electronic Purse Specifications. The Common Electronic Purse Specifications (CEPS) define a protocol for electronic payment using a chip card as a wallet. A complete electronic purse system covers three roles: A card user, a card issuer (e.g. bank institute) and a card reader as a connection between these two.

We generated parameterisable test cases from a μCRL model of the card application as follows [5]: We first applied a so-called chaos abstraction to limit the infinite behaviour due to unbounded input values. However, even the abstracted version was very large. The full state space was generated on a cluster of machines, while the minimised state space fitted in one machine. Finally, we applied enumerative test generation with the (sequential) tool TGV.

Scheduling – Clinical Chemical Analyser. Opposed to more traditional *qualitative* model checking, where properties are checked resulting in a "yes" or "no" answer, in *quantitative* model checking, measurements are performed on a model. Scheduling problems can be seen as priced reachability problems, where costs are associated with actions (and states), and the goal is to find a successful termination in a state space where the traces represent all possible schedules.

We used μCRL to model the scheduling problem of a Clinical Chemical Analyser [15], which is used to automatically analyse patient samples, designed by TNO Industry and TU/e. As naïve breadth-first search could not cope with costs and large problem instances, we developed a set of distributed techniques, incorporating minimal-cost search and several pruning techniques [14], building on the traditional notion of beam search. We were able to find solutions for several problem instances on-the-fly.

Security Analysis – Digital Rights Management. Digital Rights Management (DRM) schemes have recently attracted much research because of their essential role in enabling digital business in the entertainment market. However, sobering experiences, such as the recent Sony-BMG case, have shown that DRM systems are inherently complicated, hence error-prone, and if not applied with ample scrutiny and analysis can infringe on both vendors' and customers' rights. We extended an existing concept of DRM-preserving content redistribution in [9], where users double as content distributors.

We used μCRL to formally specify a finite model of this scheme. The resulting system is highly non-deterministic, mainly due to several fall back scenarios for suffered parties. Particularly when an intruder is included in the model, it easily hits the boundaries of single-machine state space generation. We therefore resorted to the distributed setting for generating and minimising the corresponding state space, to later on model check security goals of the scheme.

Game solving - Sokoban. A rather surprising application of our verification techniques is in solving instances of the one-player maze puzzles of Sokoban. Squares of a Sokoban instance may be occupied by stones, or marked as targets. A person can walk around or push stones, in order to move them all to the target squares, minimising the number of pushes. Walking steps are not counted.

To solve a screen, we generate its state space, and look for the shortest number of pushes leading to the goal state. However, as walk steps don't count, they should be eliminated first. Due to the large number of move options at every step, for most instances the state spaces could only be generated on a cluster of workstations. By *hiding* the walking actions, we get a state space with many τ-cycles, on which the *distributed SCC elimination tool* [12] was applied, and led to a significant reduction. In the reduced state space we can simply count the pushes in the shortest path to the success state.

3 Evaluation and Conclusion

The μCRL toolset has the capabilities to do distributed analysis on a cluster of computers. In a number of experiments we successfully applied the toolset to the areas, which have been described in the previous section. Thereby, the case studies we worked on, led to large state spaces.

In general, we used a cluster of 2.2GHz AMD Athlon 64 bit single CPU computers with 1 GB RAM each and SuSE Linux 9.3 installed. In those cases, where the number of machines is given as $n{+}1$, we used a cluster of 1.4GHz AMD Opteron 64 bit computers running under Debian 3.1. The first n machines had two CPUs and 2 GB RAM each while the extra machine had four CPUs and 16 GB main memory. As can be seen in Table 1, most problems could not have been solved on a single machine, because computation would have taken

Table 1. Performance Results

Case Study	**States**	**Transitions**	**Machine(s)**	**Time (hours)**
Functional Analysis	97,451,014	1,061,619,779	31	02:38:26
after minimisation	3,634,036	39,603,188	1	n/a
Test Generation	3,023,121	17,475,646	5	00:09:26
after minimisation	1,626	5,487	1	00:07:32
Scheduling	341,704,322	n/a	16	n/a
with beam search	7,408	n/a	1	00:00:08
Security Analysis	28,206,430	114,824,743	8+1	16:04:16
after minimisation	1,979	36,667	1	00:07:44
Game Solving	29,933,087	72,309,227	9+1	00:51:54
after τ-cycle elimination	2,583,703	7,167,175	10	00:02:01

too long and would have consumed too much memory. Therefore, problems of this size can only be solved by a toolset supporting distributed analysis features.

References

1. G. Behrmann, T. Hune, and F.W. Vaandrager. Distributing Timed Model Checking - How the Search Order Matters. In *Proc. CAV'00*, volume 1855 of *LNCS*, pages 216–231, 2000.
2. S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A Toolset for Analysing Algebraic Specifications. In *Proc. CAV'01*, volume 2102 of *LNCS*, pages 250–254, 2001.
3. S.C.C. Blom, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. New developments around the μCRL tool set. *ENTCS*, 80, 2003.
4. S.C.C. Blom and S.M. Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. *STTT*, 7(1):74–86, 2005.
5. J.R. Calamé, N. Ioustinova, and J.C. van de Pol. Towards Automatic Generation of Parameterized Test Cases from Abstractions. Technical Report SEN-E0602, CWI, March 2006.
6. T. Chothia, S.M. Orzan, J. Pang, and M. Torabi Dashti. A framework for automatically checking anonymity with μCRL. In *Proc. TGC'06*, LNCS, 2007.
7. D. Dill. The Murϕ Verification System. In *Proc. CAV'96*, volume 1102 of *LNCS*, pages 390–393, 1996.
8. H. Garavel, R. Mateescu, D. Bergamini, A. Curic, N. Descoubes, C. Joubert et al. DISTRIBUTOR and BCG_MERGE: Tools for Distr. Explicit State Space Generation. In *TACAS'06*, volume 3920 of *LNCS*, pages 445–449, 2006.
9. H. Jonker, S. Krishnan Nair, and M. Torabi Dashti. Nuovo DRM paradiso. Technical Report SEN-R0602, CWI, Amsterdam, 2006.
10. F. Lerda and R. Sista. Distributed-Memory model checking with SPIN. In *Proc. SPIN'99*, volume 1680 of *LNCS*, pages 22–39, 1999.
11. B. Lisser. Distributed State Space Generator (preliminary). `http://www.cwi.nl/~mcrl/instantiators.pdf`, 2006.
12. S.M. Orzan and J.C. van de Pol. Detecting strongly connected components in large distributed state spaces. Technical Report SEN-E0501, CWI, 2005.

13. J. Pang, W. J. Fokkink, R. F.H. Hofman, and R. Veldema. Model checking a cache coherence protocol of a Java DSM implementation. *JLAP*, 71:1–43, 2007.
14. A.J. Wijs and B. Lisser. Distributed Extended Beam Search for Quantitative Model Checking. In *MoChArt'06*, LNAI, 2007.
15. A.J. Wijs, J.C. van de Pol, and E. Bortnik. Solving Scheduling Problems by Untimed Model Checking. In *Proc. FMICS '05*, pages 54–61. ACM Press, 2005.

A Appendix

Security Analysis - Anonymity. Anonymity is a non-standard security property, in the sense that it is not verifiable by model checking directly, but requires special techniques, where state space minimisation is essential [6].

The powerful *distributed state space generation* and *minimisation* tools of the μCRL toolset allowed us to automatically check anonymity for large instances of known protocols. For instance, the Dining Cryptographers protocol, used as case study for many tools, has so far been verified for a maximum of 8 participants. We succeed in verifying it for 15 participants in a few hours. Moreover, the anonymity property of the FOO voting protocol has never before been established in an automatic framework. Our toolset supports its verification for up to 7 voters.

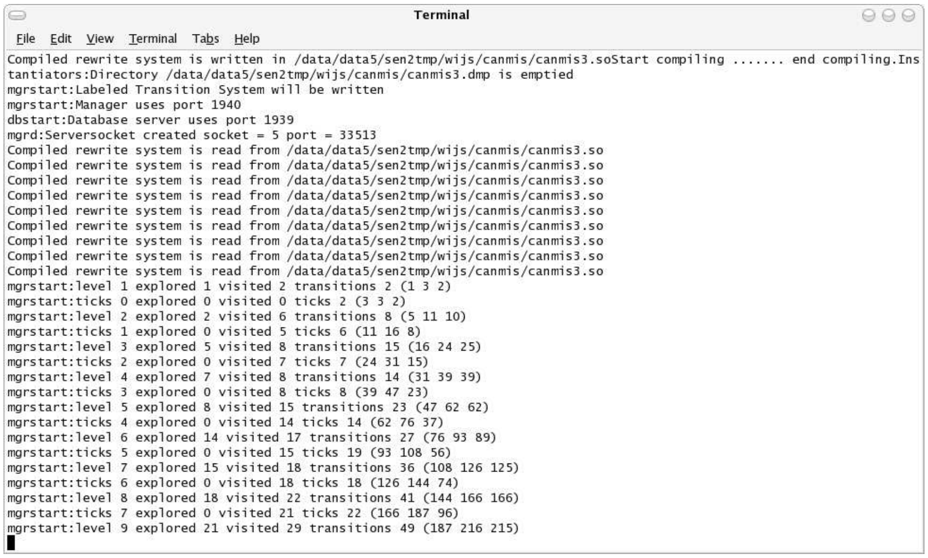
For this second security analysis case study, we generated a state space of 65,282,690 states and 221,299,564 transitions. It could then be minimised to 3,676,249 states and 9,628,686 transitions. On a cluster with 16 machines as described in Section 3, this took us 4 hours and 48 minutes.

The Toolset in Action. The toolset described in the paper is used on a regular basis in the area of computer science research. The toolset is available in open source from the website `http://www.cwi.nl/~mcrl/tacas2007/`.

The presentation of the toolset is planned as follows: First, we will give an introduction to the toolset in general before discussing its technical aspects. These aspects will be shown by an exemplary execution of the beam search example. This execution will be given as an animation as follows:

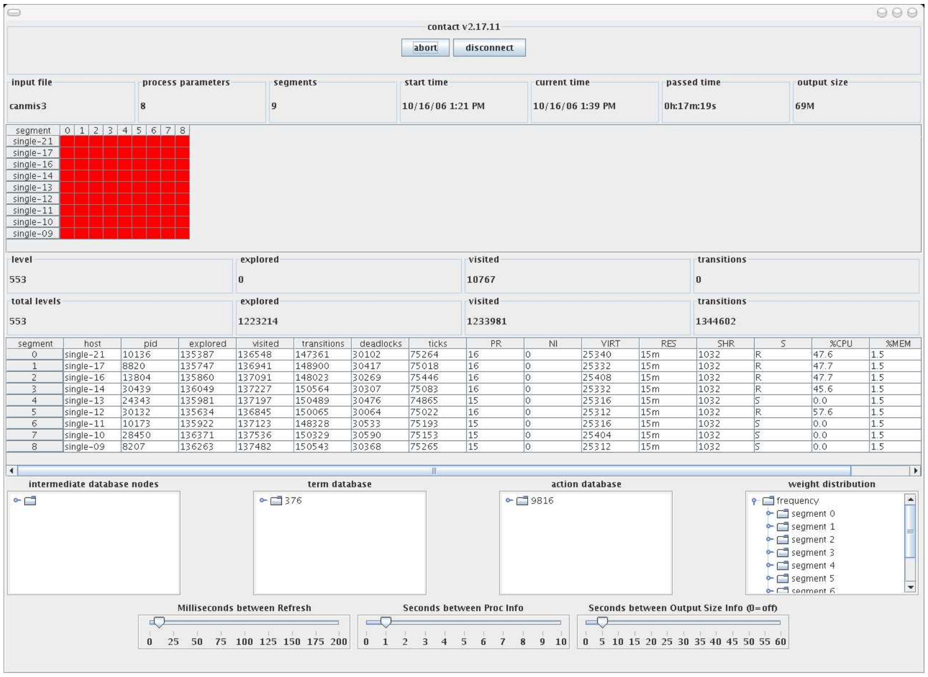
1. Starting a job on the cluster (Figure 1).
2. Starting the tool *contact*, a monitoring GUI for the toolset.
3. Discussion of the different states of job execution: *idle* (color white), *busy* (color red, see Figure 2), *communicating* (color yellow, see Figure 3) and *finished* (color green, ibid).
4. Interpretation of the results (showing output files, e.g. the state space directory, and explaining their meaning).

Afterwards, we will give a short introduction into each of the given case studies. This introduction will contain some information about the case study itself and about the results we achieved in the experiments.



```
Compiled rewrite system is written in /data/data5/sen2tmp/wijs/canmis/canmis3.soStart compiling ....... end compiling.Ins
tantiators:Directory /data/data5/sen2tmp/wijs/canmis/canmis3.dmp is emptied
mgrstart:Labeled Transition System will be written
mgrstart:Manager uses port 1940
dbstart:Database server uses port 1939
mgrd:Serversocket created socket = 5 port = 33513
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
Compiled rewrite system is read from /data/data5/sen2tmp/wijs/canmis/canmis3.so
mgrstart:level 1 explored 1 visited 2 transitions 2 (1 3 2)
mgrstart:ticks 0 explored 0 visited 0 ticks 2 (3 3 2)
mgrstart:level 2 explored 2 visited 6 transitions 8 (5 11 10)
mgrstart:ticks 1 explored 0 visited 5 ticks 6 (11 16 8)
mgrstart:level 3 explored 5 visited 8 transitions 15 (16 24 25)
mgrstart:ticks 2 explored 0 visited 7 ticks 7 (24 31 15)
mgrstart:level 4 explored 7 visited 8 transitions 14 (31 39 39)
mgrstart:ticks 3 explored 0 visited 8 ticks 8 (39 47 23)
mgrstart:level 5 explored 8 visited 15 transitions 23 (47 62 62)
mgrstart:ticks 4 explored 0 visited 14 ticks 14 (62 76 37)
mgrstart:level 6 explored 14 visited 17 transitions 27 (76 93 89)
mgrstart:ticks 5 explored 0 visited 15 ticks 19 (93 108 56)
mgrstart:level 7 explored 15 visited 18 transitions 36 (108 126 125)
mgrstart:ticks 6 explored 0 visited 18 ticks 18 (126 144 74)
mgrstart:level 8 explored 18 visited 22 transitions 41 (144 166 166)
mgrstart:ticks 7 explored 0 visited 21 ticks 22 (166 187 96)
mgrstart:level 9 explored 21 visited 29 transitions 49 (187 216 215)
█
```

Fig. 1. Starting a job



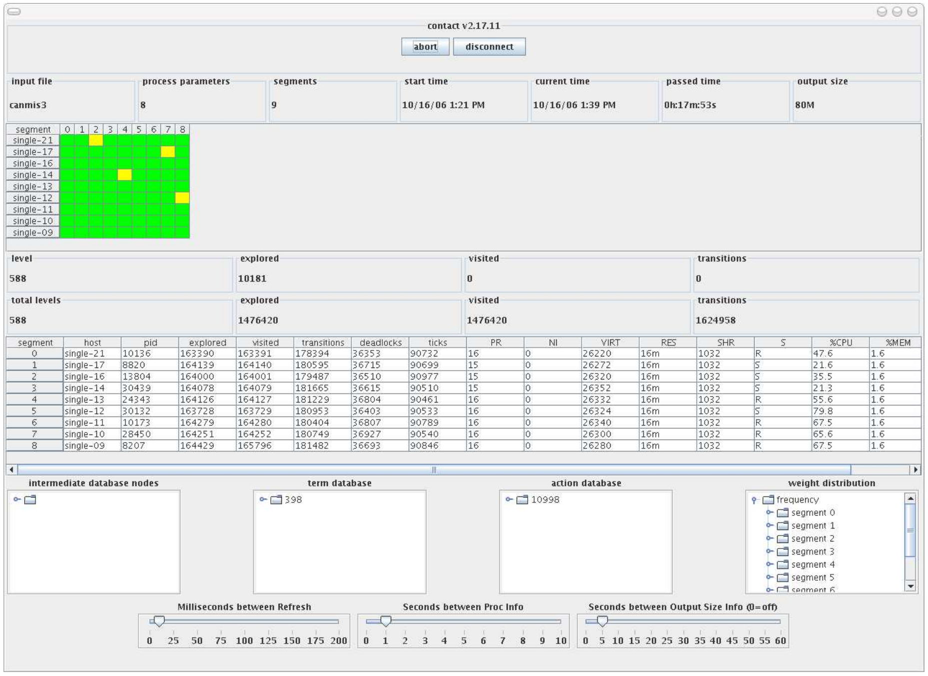
input file: canmis3
process parameters: 8
segments: 9
start time: 10/16/06 1:21 PM
current time: 10/16/06 1:39 PM
passed time: 0h:17m:19s
output size: 69M

segment	0	1	2	3	4	5	6	7	8
single-21	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-17	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-16	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-14	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-13	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-12	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-11	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
single-09	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥

level: 553
explored: 0
visited: 10767
transitions: 0
total levels: 553
explored: 1223214
visited: 1233981
transitions: 1344602

segment	host	pid	explored	visited	transitions	deadlocks	ticks	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
0	single-21	10136	135387	136548	147361	30102	75264	16	0	25340	15m	1032	R	47.6	1.5
1	single-17	8820	135747	136941	148900	30417	75018	16	0	25332	15m	1032	R	47.7	1.5
2	single-16	13804	135860	137091	148023	30269	75446	16	0	25408	15m	1032	R	47.7	1.5
3	single-14	30439	136049	137227	150564	30307	75083	16	0	25332	15m	1032	R	45.6	1.5
4	single-13	24343	135981	137197	150489	30476	74865	15	0	25316	15m	1032	S	0.0	1.5
5	single-12	30132	135634	136845	150065	30064	75022	16	0	25312	15m	1032	R	57.6	1.5
6	single-11	10173	135922	137123	148328	30533	75193	15	0	25316	15m	1032	S	0.0	1.5
7	single-10	28450	136371	137536	150329	30590	75153	15	0	25404	15m	1032	S	0.0	1.5
8	single-09	8207	136263	137482	150543	30368	75265	15	0	25312	15m	1032	S	0.0	1.5

Fig. 2. All processors calculating



input file: canmis3
process parameters: 8
segments: 9
start time: 10/16/06 1:21 PM
current time: 10/16/06 1:39 PM
passed time: 0h:17m:53s
output size: 80M

segment	0	1	2	3	4	5	6	7	8
single-21	🟩	🟩	🟨	🟩	🟩	🟩	🟩	🟩	🟩
single-17	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟨	🟩
single-16	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
single-14	🟩	🟩	🟩	🟩	🟨	🟩	🟩	🟩	🟩
single-13	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
single-12	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟨
single-11	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
single-10	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
single-09	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

level: 588
explored: 10181
visited: 0
transitions: 0

total levels: 588
explored: 1476420
visited: 1476420
transitions: 1624958

segment	host	pid	explored	visited	transitions	deadlocks	ticks	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
0	single-21	10136	163390	163391	178394	36353	90732	16	0	26220	16m	1032	R	47.6	1.6
1	single-17	8820	164139	164140	180595	36715	90699	15	0	26272	16m	1032	S	21.6	1.6
2	single-16	13804	164000	164001	179487	36510	90977	15	0	26320	16m	1032	S	35.5	1.6
3	single-14	30439	164078	164079	181665	36615	90510	15	0	26352	16m	1032	S	21.3	1.6
4	single-13	24343	164126	164127	181229	36804	90461	16	0	26332	16m	1032	R	55.6	1.6
5	single-12	30132	163728	163729	180953	36403	90533	16	0	26324	16m	1032	S	79.8	1.6
6	single-11	10173	164279	164280	180404	36807	90789	16	0	26340	16m	1032	R	67.5	1.6
7	single-10	28450	164251	164252	180749	36927	90540	16	0	26300	16m	1032	R	65.6	1.6
8	single-09	8207	164429	165796	181482	36693	90846	16	0	26280	16m	1032	R	67.5	1.6

intermediate database nodes

term database: 398

action database: 10998

weight distribution
- frequency
  - segment 0
  - segment 1
  - segment 2
  - segment 3
  - segment 4
  - segment 5
  - segment 6

Fig. 3. Some processors communicating