# Analyzing the Energy Efficiency of Cluster Scheduling Schemes by Probabilistic Model Checking*

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
Wolfgang.Schreiner@risc.jku.at

Tamás Bérczes
Faculty of Informatics
University of Debrecen, Hungary
berczes.tamas@inf.unideb.hu

Ádám Tóth
Faculty of Informatics
University of Debrecen, Hungary
adamtoth102@gmail.com

December 15, 2014

**Abstract**

We report in this paper on our results of modeling and analyzing with the probabilistic model checker PRISM the energy efficiency of various cluster scheduling schemes. These schemes were originally introduced by Do, Vu, Tran, and Nguyen in their paper "A generalized model for investigating scheduling schemes in computational clusters" and analyzed by simulation there. Our investigations with PRISM validate the reported results and also subsequent results that were achieved with the performance analysis tool MOSEL-2. Furthermore, some measures that could be determined by simulation but not with MOSEL-2 could be also derived with PRISM.

*Supported by the project HU 10/2012 of the Austrian Academic Exchange Service (ÖAD) and by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project. This project has been supported by the European Union, co-financed by the European Social Fund.

# Contents

# 1. Introduction

In this paper we extend our investigations started in [6] on modeling and analyzing various scheduling schemes for computational clusters that were originally presented in [2] and analyzed be simulation there. We apply in our investigations the probabilistic model checker PRISM [3, 5] by constructing and solving precise mathematical system models. While in [6] we have dealt with timing properties, we focus in this paper on the energy consumption properties of the proposed scheduling schemes. To make this presentation self-contained, with repeat our introduction from [6]:

> The system presented in [2] consists of $K$ classes of servers; the servers in each class have identical performance and energy consumption; typically, the higher the performance of a server is, the more energy it consumes. The devised scheduling schemes are based on a ranking of server classes, either by high performance (HP) or by energy efficiency (EE). Roughly speaking, when a new job arrives, highly ranked servers are preferred over lowly ranked ones. In more detail, there are three schemes:
>
> 1. *Separate Queue Scheme:* each server has a separate queue from which it accepts and processes jobs. A new job is placed into the shortest queue; if there are multiple such queues, the queue of the most highly ranked server is chosen.
> 2. *Class Queue Scheme:* each server class has a separate queue from which the servers of this class accept and process jobs. A new job is placed into the shortest queue; if there are multiple such queues, the queue of the most highly ranked class is chosen.
> 3. *Common Queue Scheme:* there is a single common queue from which all servers accept and process jobs. A new job is forwarded to the most highly ranked free server; if there is no such server, the job is placed into the common queue.
>
> The various schemes are compared in [2] with respect to various criteria for both the HP and the EE ranking. The paper assumes an infinite source model where jobs arrive with a constant rate $\lambda$ that is determined to establish a desired system utilization $U$ in the range of 50% to 90%.

In this paper, we undertake to analyze the models with respect to the energy related measures presented in [2], in particular the average amount of energy consumed per job in the different schemes under different ranking criteria; we also investigate the effect of applying Dynamic Voltage/Frequency Scaling (DVFS) discussed in that paper. While in [2] only infinite source models are discussed, we generalize our investigations also to the finite source case (based on the models that we have previously introduced in [6]); this allows us to compare the results we those presented in the forthcoming paper [7] where the finite source models are described and analyzed with the help of the performance modeling tool MOSEL-2 [1].

The rest of the paper is organized as follows: in Section 2, we present the infinite source models and their analysis while in Section 3 we extend our considerations to the finite source

case. In Section 4, we present our conclusions and discuss further work. Appendix A contains the full listing of the infinite source models and of the properties that were used for the analysis; Appendix B contains the corresponding listings for the finite source variants.

## 2. Infinite Source Models

In this section, we extend the timing analysis of some of the infinite source models presented in [6] to the analysis of the energy consumption of the proposed cluster scheduling schemes. This analysis corresponds to the analysis originally presented in [2] based on a couple of measures described below.

**Preliminaries** Let $K$ be the number of server classes and let $M(i)$ be the number of servers of class $i$. Let $P_{ac,i}$ be the power consumption of each server of class $i$ when the server is *active*, i.e., busy with processing a job; let $P_{id,i}$ be the power consumption of the server when it is *idle*, i.e., not processing a job. Let $\alpha_{i,j}(t)$ denote the sum of active time periods of server $j$ in class $i$ until time $t$; let $\iota_{i,j}(t)$ be the corresponding sum of idle time periods. Then the average energy consumption per job when idle servers are not switched off (but function in the idle state with lower power) until the departure time $t_n$ of job $n$ is

$$AE_{no-switch}(n) := (1/n) \cdot \sum_{i=1}^{K} \left( P_{ac,i} \cdot \sum_{j=1}^{M(i)} \alpha_{i,j}(t_n) + P_{id,i} \cdot \sum_{j=1}^{M(i)} \iota_{i,j}(t_n) \right)$$

and the average energy consumption per job when idle servers are switched off is

$$AE_{switch-off}(n) := (1/n) \cdot \sum_{i=1}^{K} \left( P_{ac,i} \cdot \sum_{j=1}^{M(i)} \alpha_{i,j}(t_n) \right)$$

The corresponding long term average energy consumption per job are

$$AE_{no-switch} := \lim_{n \to \infty} AE_{no-switch}(n)$$
$$AE_{switch-off} := \lim_{n \to \infty} AE_{switch-off}(n)$$

Our core goal in this paper is to determine these long term values.

To determine such quantities, PRISM allows to define a *reward structure* $r = (\underline{\rho}, \iota)$ which assigns to every state $s$ a reward $\underline{\rho}(s) \geq 0$ and to every transition $s \to s'$ a reward $\iota(s, s') \geq 0$ [4]. Given an execution path $\omega = s_0 \to s_1 \to s_2 \cdots$ that traverses the states $s_0, s_1, s_2 \ldots$, we may thus define the reward

$$X_{C \leq t}(\omega) := \sum_{i=0}^{j_t - 1} \left( t_i \cdot \underline{\rho}(s_i) + \iota(s_i, s_{i+1}) \right) + \left( t - \sum_{i=0}^{j_t - 1} t_i \right) \cdot \underline{\rho}(s_{j_t})$$

accumulated on that path up to time $t$ where $t_i$ denotes the time that the system stays in state $s_i$. Intuitively, this execution accumulates for every transition $s_i \to s_{i+1}$ the state reward $t_i \cdot \underline{\rho}(s_i)$ and the transition reward $\iota(s_i, s_{i+1})$.

The PRISM model checker allows by a query

```
R{"r"}=? [S]
```

to determine the quantity

$$\lim_{t\to\infty}(1/t)\cdot Exp(X_{C\leq t})$$

where $Exp(X)$ denotes the expectation of random variable $X$ with respect to the probability measure determined by the system, i.e., intuitively the expected value of the reward with respect to all possible execution paths.

**Class Queue Model** Appendix A.1 lists the PRISM version of the "class queue" model for $K = 3$ classes of $M = 8$ servers each; the state variables $q_1, q_2, q_3$ determine the number of jobs assigned to class $1, 2, 3$, respectively. We define the state rewards

```
rewards "TE_noswitch"
  true:
    pac1*min(M,q1)   + pac2*min(M,q2)   + pac3*min(M,q3) +
    pid1*max(0,M-q1) + pid2*max(0,M-q2) + pid3*max(0,M-q3);
endrewards

rewards "TE_switchoff"
  true:
    pac1*min(M,q1)   + pac2*min(M,q2)   + pac3*min(M,q3);
endrewards
```

which assign to each of the $\min(M, q_i)$ active servers of class $i$ the active energy consumption $P_{ac,i}$ and to each of the $\max(0, M - q_i)$ idle servers the idle energy consumption $P_{id,i}$. The PRISM queries

```
"TE_noswitch":  R{"TE_noswitch"}=?  [ S ] ;
"TE_switchoff": R{"TE_switchoff"}=? [ S ] ;
```

thus determine the quantities

$$\lim_{n\to\infty}(1/t_n)\cdot n\cdot AE_{no-switch}(n)$$
$$\lim_{n\to\infty}(1/t_n)\cdot n\cdot AE_{switch-off}(n)$$

respectively, i.e., the expected energy consumption rate of the system (Joule/second = Watt).

To determine the average energy consumption of the system per processed job (Joule/job = Watt second/job), we define the transition rewards

```
rewards "done"
  [done1] true : 1;
  [done2] true : 1;
  [done3] true : 1;
endrewards
```

which assign to every transition by which a job is terminated a value 1; the PRISM query

```
"done": R{"done"}=? [ S ] ;
```

thus determines the quantity

$$\lim_{n\to\infty} (1/t_n) \cdot n$$

By the queries

```
"AE_noswitch":  "TE_noswitch"/"done";
"AE_switchoff": "TE_switchoff"/"done";
```

we can thus determine the desired values

$$AE_{no-switch} = \lim_{n\to\infty} AE_{no-switch}(n) = \frac{\lim_{n\to\infty}(1/t_n) \cdot n \cdot AE_{no-switch}(n)}{\lim_{n\to\infty}(1/t_n) \cdot n}$$

$$AE_{switch-off} = \lim_{n\to\infty} AE_{switch-off}(n) = \frac{\lim_{n\to\infty}(1/t_n) \cdot n \cdot AE_{switch-off}(n)}{\lim_{n\to\infty}(1/t_n) \cdot n}$$

The top row of diagrams in Figure 1 represent the energy consumption for the class queue scheme and the EE (energy efficiency) policy respectively HP (high performance) policy; the results correspond very closely to those presented in Figures 10 and 11 of [2].

**Common Queue Model** Appendix A.2 lists the PRISM version of the corresponding "common queue" model[1] for $K = 3$ classes of $M = 8$ servers each; here the variables $q_1, q_2, q_3$ represent the number of servers in classes $1, 2, 3$ to which new jobs have been assigned on arrival and that thus are active. This simplifies the state rewards to

```
rewards "TE_noswitch"
  true:
    pac1*q1     + pac2*q2     + pac3*q3 +
    pid1*(M-q1) + pid2*(M-q2) + pid3*(M-q3);
  ...
endrewards

rewards "TE_switchoff"
  true:
    pac1*q1     + pac2*q2     + pac3*q3;
  ...
endrewards
```

However, servers that become idle may also process jobs from a common queue (represented by system variable $q$) to which jobs have been assigned that did not immediately find a free server. The system state does not indicate that some of the seemingly "idle" servers may actually process jobs from the common queue; the state reward may thus underestimate the power consumption of the system by considering servers as idle that are actually active. To compensate the difference, we assign to the transitions that indicate the completion of jobs from the common queue the expected difference in energy consumption:

[1] The listed model uses the DVFS parameters discussed in the following subsection.

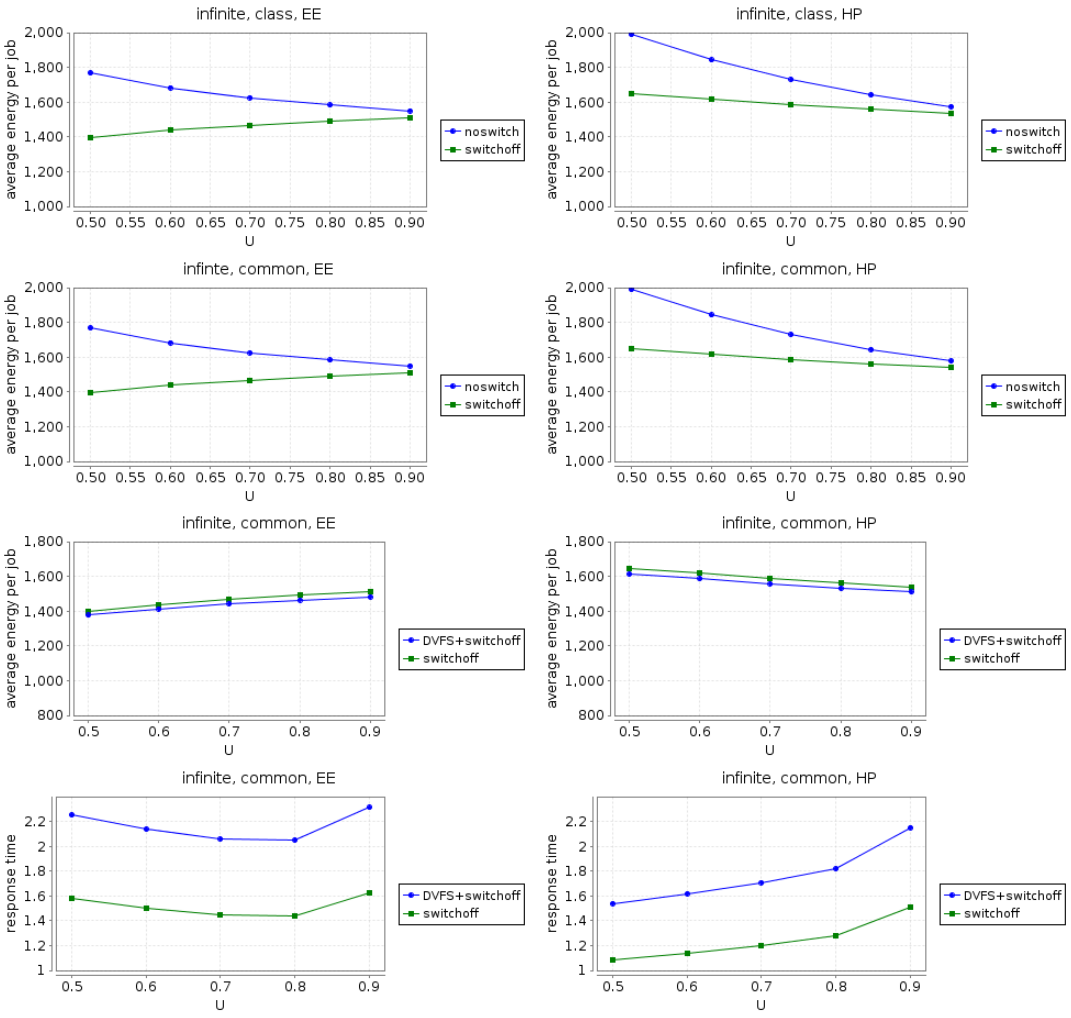


Figure 1: Infinite Source Class/Common Queue ($3 \cdot 8$ Servers, $Q = 24/72$)

```
rewards "TE_noswitch"
  true:
    ...
  [done1b] true : (pac1-pid1)*(1/mu1);
  [done2b] true : (pac2-pid2)*(1/mu2);
  [done3b] true : (pac3-pid3)*(1/mu3);
endrewards

rewards "TE_switchoff"
  true:
    ...
  [done1b] true : pac1*(1/mu1);
  [done2b] true : pac2*(1/mu2);
  [done3b] true : pac3*(1/mu3);
endrewards
```

Please note that for each such state transition the power consumption is weighted with the expected time $1/\mu_i$ that the server of class $i$ spent in processing the request; the solution is not very elegant but satisfies its purpose. The second row in Figure 1 depicts the results for the common queue model; the difference to the class queue model are negligible which corresponds to the results presented in Figures 10 and 11 of [2].

**Dynamic Voltage/Frequency Scaling** Dynamic Voltage/Frequency Scaling (DVFS) is a technique where the execution frequency of a processor is deliberately reduced in order to save energy at the price of simultaneously reducing its performance. In [2], the impact of DVFS is analyzed by adjusting the performance/power consumption figures for active processors according to published specifications (about 70% of full performance/power). The common queue model presented in Appendix A.2 reflects these data.

The third row of diagrams in Figure 1 depicts the energy consumption of the common queue model and switch-off with both DVFS employed and not employed; the results correspond to those presented in Figure 13 of [2]. The bottom row of diagrams in Figure 1 depicts the consequences of DVFS with respect to service response time; the results correspond to Figure 12 of [2].

As can be seen from these results, by the application of DVFS the response time increases by about 40% in proportion to the reduced processor frequency ($1/0.7 \approx 1.4$) which is as expected. On the other side, the average amount of energy consumed per job is only slightly decreased. Actually, if the reduction in both performance and power were indeed the same, there would be no reduction in the average consumption at all (because the decreased amount of energy per time consumed would be exactly compensated by the correspondingly increased execution time of a job). The only reason why there is some improvement in the average energy consumption is that the reduction in power is slightly higher than that of execution frequency (68.8% versus 70.4%); only the relative difference (2.3%) is responsible for the slight average reduction. This simple explanation which shows that the results concerning DVFS are not really very deep was not given in [2], thus it is unclear whether the authors have actually understood this relationship.

## 3. Finite Source Models

In [6] we have adapted the infinite source models presented in [2] to the finite source case; in this section we extend the analysis of some of these models to consider energy consumption. In the forthcoming paper [7] a similar analysis is performed with the performance modeling tool MOSEL-2.

The analysis of both the class queue model and the common queue model presented in Section 2 can be easily transferred to the finite source case; in this section we also discuss the separate queue model. In this model, every server $j$ of class $i$ has a separate queue modeled by the system variable $q_{i,j}$. We can model the performance consumption of the system by a corresponding state reward

```
rewards "TE_noswitch"
  q11 = 0 : pid1;
  q11 > 0 : pac1;
  q12 = 0 : pid1;
  q12 > 0 : pac1;
  q13 = 0 : pid1;
  q13 > 0 : pac1;
  ...
endrewards

rewards "TE_switchoff"
  q11 > 0 : pac1;
  q12 > 0 : pac1;
  q13 > 0 : pac1;
  ...
endrewards
```

which accumulates for every active server of class $i$ the power consumption $P_{ac,i}$ and correspondingly for every idle server the power consumption $P_{id,i}$.

The first and the third rows of diagrams in Figure 2 depict the amount of energy consumed per time unit for both the class queue and the separate queue models; the results correspond to those presented in Figures 3.10–3.17 of [7].

However, while in [7] with MOSEL-2 only the energy consumption rate could be analyzed, we can also analyze (as described in the previous section) the average energy per job by utilizing transition rewards. For the class queue model this transition reward remains the same; for the separate queue model it becomes

```
rewards "done"
  [done11] true : 1;
  [done12] true : 1;
  [done13] true : 1;
  ...
endrewards
```

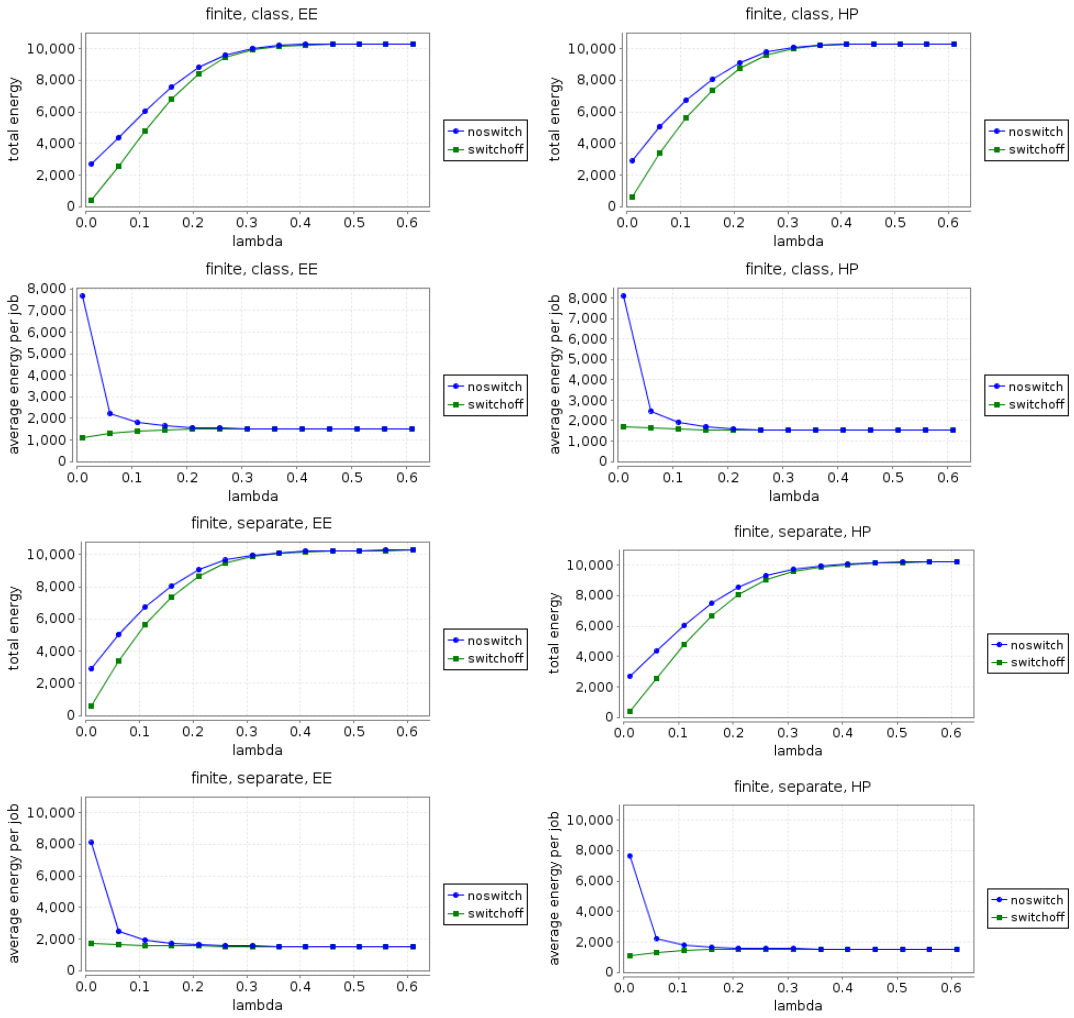


Figure 2: Finite Source Class/Separate Queue ($3 \cdot 3$ Servers, $N = 36$, $Q = 24/4$)

where to each transition indicating the termination of a job a transition rate of 1 is assigned. The corresponding results are depicted in the second and fourth row of Figure 2.

## 4. Conclusions

This paper has extended our previous analysis of the timing properties of various cluster scheduling schemes to the energy consumption properties. The property description and querying language of PRISM has proved to be very well up to this task; this has allowed us to validate the results that were previously determined by simulation by actual calculations.

As it turns out, for the determination of the desired measures not only state rewards but also transition rewards were required in order to relate a state-based quantity (energy consumed per time unit) to an event-based property (number of jobs consumed per time unit). The richer property description and querying language of PRISM thus has allowed to produce more results than would be possible with previously used tools such as MOSEL-2.

By our work, essentially all the measures derived by simulation, both time-related and energy-related, could be also derived by actual calculations. In further work, we may undertake the investigation of not yet considered properties in this and other models.

## References


[1] K. Begain, G. Bolch, and Herold H. *Practical Performance Modeling Application of the MOSEL Language*. Kluwer Academic Publisher, 2012.

[2] Tien v. Do, Binh T. Vu, Xuan T. Tran, and Anh P. Nguyen. "A Generalized Model for Investigating Scheduling Schemes in Computational Clusters". In: *Simulation Modelling Practice and Theory* 37 (2013), pp. 30–42.

[3] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-time Systems". In: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591.

[4] Marta Kwiatkowska, Gethin Norman, and David Parker. "Stochastic Model Checking". In: *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*. Ed. by M. Bernardo and J. Hillston. Vol. 4486. Lecture Notes in Computer Science (Tutorial Volume). Springer, June 2007, pp. 220–270.

[5] David A. Parker, ed. *PRISM — Probabilistic Symbolic Model Checker*. Department of Computer Science, University of Oxford, UK. 2013. URL: http://www.prismmodelchecker.org.

[6] Wolfgang Schreiner, Tamás Bérczes, and Ádám Tóth. *Analyzing Cluster Scheduling Schemes by Probabilistic Model Checking*. Technical Report. Johannes Kepler University Linz, Austria: Research Institute for Symbolic Computation (RISC), Sept. 2014. URL: http://www.risc.jku.at/publications/download/risc_5059/main.pdf.

[7] Ádám Tóth. *Véges forrású klaszter hálózatok hatékonyság analízise (Analysing the performance of finite-source networks of clusters, in Hungarian)*. Technical Report. To appear. University of Debrecen, Hungary: Department of Computer Science, 2015.

# A. The Infinite Source Models

## A.1. Class Queue

```
// -------------------------------------------------------------------
// InfiniteClassHP.prism
// A scheduling model for computational clusters.
//
// The model is the "class queue" model with
// "high-performance priority" described in
//
//   Tien v. Do, Binh t. Vu, Xuan T. Tran, Anh P. Nguyen:
//   "A generalized model for investigating scheduling schemes in
//   computational clusters", Simulation Modelling Practice and
//   Theory, 37 (2013), 30-42.
//
// Authors: Berczes Tamas <berczes.tamas@inf.unideb.hu> and
// Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
//
// Copyright (C) 2014 Department of Informatics Systems and Networks,
// University of Debrecen, Debrecen, Hungary (http://irh.inf.unideb.hu)
// and Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// -------------------------------------------------------------------

// checking parameters: "sparse", "Gauss", default epsilon

// continuous time markov chain (ctmc) model
ctmc

// -------------------------------------------------------------------
// system parameters
// -------------------------------------------------------------------

// system utilization (50%, 60%, 70%, 80%, 90%)
const double U0;

// queue size
const int Q = 24;

// number of server classes and number of servers per class
const int K = 3;
const int M = 8;

// arrival rates
const double lambda = U0*M*(rp1+rp2+rp3);

// ranking based on performance
const double rp1 = 1.0;
const double rp2 = 0.82;
const double rp3 = 0.43;

// ranking based on energy efficiency
const double re1 = 0.64;
```

```
const double re2 = 0.66;
const double re3 = 1.0;

// active power consumption per class
const double pac10 = 1700;
const double pac20 = 1275;
const double pac30 = 457;

// idle power consumption per class
const double pid10 = 364;
const double pid20 = 331;
const double pid30 = 108;

// reduced power consumption in DVFS mode
const double pfs10 = 1169;
const double pfs20 = 881;
const double pfs30 = 317;

// ------------------------------------------------------------------
// system parameters according to chosen ranking
// ------------------------------------------------------------------

// service rates according to chosen ranking
const double mu1 = rp1;
const double mu2 = rp2;
const double mu3 = rp3;

// active power consumption according to chosen ranking
const double pac1 = pac10;
const double pac2 = pac20;
const double pac3 = pac30;

// idle power consumption according to chosen ranking
const double pid1 = pid10;
const double pid2 = pid20;
const double pid3 = pid30;

// reduced power consumption according to chosen ranking
const double pfs1 = pfs10;
const double pfs2 = pfs20;
const double pfs3 = pfs30;

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module System
  q1: [0..Q] init 0;
  q2: [0..Q] init 0;
  q3: [0..Q] init 0;

  [done1] q1 > 0 -> min(M,q1)*mu1 : (q1' = q1-1);
  [done2] q2 > 0 -> min(M,q2)*mu2 : (q2' = q2-1);
  [done3] q3 > 0 -> min(M,q3)*mu3 : (q3' = q3-1);
```

```
  [] true -> lambda :
    (q1' =
       (q1 < M | (q2 >= M & q3 >= M & q1 <= q2 & q1 <= q3)) & q1 < Q ?
       q1+1 : q1) &
    (q2' =
       q1 >= M & (q2 < M | (q3 >= M & q2 < q1 & q2 <= q3)) & q2 < Q ?
       q2+1 : q2) &
    (q3' =
       q1 >= M & q2 >= M & (q3 < M | (q3 < q1 & q3 < q2)) & q3 < Q ?
       q3+1 : q3) ;
endmodule

// ------------------------------------------------------------------
// system rewards
// ------------------------------------------------------------------

rewards "stime"
  !(q1 = Q & q2 = Q & q3 = Q) :
  q1 < M ? (1/mu1) :
  q2 < M ? (1/mu2) :
  q3 < M ? (1/mu3) :
  q1 <= q2 & q1 <= q3 ? (1/mu1) :
             q2 <= q3 ? (1/mu2) :
                        (1/mu3) ;
endrewards

rewards "qload"
  true : q1+q2+q3;
endrewards

rewards "TE_noswitch"
  true:
    pac1*min(M,q1)   + pac2*min(M,q2)   + pac3*min(M,q3) +
    pid1*max(0,M-q1) + pid2*max(0,M-q2) + pid3*max(0,M-q3);
endrewards

rewards "TE_switchoff"
  true:
    pac1*min(M,q1)   + pac2*min(M,q2)   + pac3*min(M,q3);
endrewards

rewards "done"
  [done1] true : 1;
  [done2] true : 1;
  [done3] true : 1;
endrewards

// ------------------------------------------------------------------
// InfiniteClass.props
// ------------------------------------------------------------------

// rejection probability
"reject": S=? [ q1 = Q & q2 = Q & q3 = Q ] ;
```

```
// probability that all servers are busy
"busy": S=? [ q1 >= M & q2 >= M & q3 >= M ] ;

// mean service time
"stime": R{"stime"}=? [ S ] ;

// response time (of accepted jobs)
"qload":  R{"qload"}=? [ S ];
"rtime":  "qload"/(lambda*(1-"reject"));

// waiting time (of accepted jobs)
"wtime":  "rtime"-"stime";

// expected energy consumption
"TE_noswitch":  R{"TE_noswitch"}=?  [ S ] ;
"TE_switchoff": R{"TE_switchoff"}=? [ S ] ;

// the expected number of finished jobs per time unit
"done": R{"done"}=? [ S ] ;

// expected energy consumption per job
"AE_noswitch":  "TE_noswitch"/"done";
"AE_switchoff": "TE_switchoff"/"done";
```

## A.2. Common Queue with DVFS

```
// ------------------------------------------------------------------
// InfiniteCommonHP.prism
// A scheduling model for computational clusters.
//
// The model is the "common queue" model with
// "high-performance priority" with "dynamic voltage/frequency scaling"
// described in
//
//   Tien v. Do, Binh t. Vu, Xuan T. Tran, Anh P. Nguyen:
//   "A generalized model for investigating scheduling schemes in
//   computational clusters", Simulation Modelling Practice and
//   Theory, 37 (2013), 30-42.
//
// Authors: Berczes Tamas <berczes.tamas@inf.unideb.hu> and
// Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
//
// Copyright (C) 2014 Department of Informatics Systems and Networks,
// University of Debrecen, Debrecen, Hungary (http://irh.inf.unideb.hu)
// and Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// ------------------------------------------------------------------

// checking parameters: "sparse", "Jacobi", default epsilon

// continuous time markov chain (ctmc) model
```

```
ctmc

// ------------------------------------------------------------------
// system parameters
// ------------------------------------------------------------------

// system utilization (50%, 60%, 70%, 80%, 90%)
const double U0;

// arrival rates
const double lambda = U0*8*(rp1+rp2+rp3);

// queue size
const int Q = 72;

// number of server classes and number of servers per class
const int K = 3;
const int M = 8;

// ranking based on performance
const double rp1 = 1.0 *(4517449/6419253);
const double rp2 = 0.82*(3706521/5286503);
const double rp3 = 0.43*(1961557/2790966);

// ranking based on energy efficiency
const double re1 = 0.64;
const double re2 = 0.66;
const double re3 = 1.0;

// active power consumption per class
const double pac10 = 1700;
const double pac20 = 1275;
const double pac30 = 457;

// idle power consumption per class
const double pid10 = 364;
const double pid20 = 331;
const double pid30 = 108;

// reduced power consumption in DVFS mode
const double pfs10 = 1169;
const double pfs20 = 881;
const double pfs30 = 317;

// ------------------------------------------------------------------
// system parameters according to chosen ranking
// ------------------------------------------------------------------

// service rates according to chosen ranking
const double mu1 = rp1;
const double mu2 = rp2;
const double mu3 = rp3;

// active power consumption according to chosen ranking
```

```
const double pac1 = pfs10;
const double pac2 = pfs20;
const double pac3 = pfs30;

// idle power consumption according to chosen ranking
const double pid1 = pid10;
const double pid2 = pid20;
const double pid3 = pid30;

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module System
  q: [0..Q] init 0;

  q1: [0..M] init 0;
  q2: [0..M] init 0;
  q3: [0..M] init 0;

  [done1a] q1 > 0 -> q1*mu1 : (q1' = q1-1);
  [done1b] q1 < M & q > 0 -> (M-q1)*mu1 : (q' = q-1);

  [done2a] q2 > 0 -> q2*mu2 : (q2' = q2-1);
  [done2b] q2 < M & q > 0 -> (M-q2)*mu2 : (q' = q-1);

  [done3a] q3 > 0 -> q3*mu3 : (q3' = q3-1);
  [done3b] q3 < M & q > 0 -> (M-q3)*mu3 : (q' = q-1);

  [] true -> lambda :
    (q' =
      q1 = M & q2 = M & q3 = M & q < Q ?
        q+1 : q) &
    (q1' =
      q1 < M ?
        q1+1 : q1) &
    (q2' =
      q1 = M & q2 < M ?
        q2+1 : q2) &
    (q3' =
      q1 = M & q2 = M & q3 < M ?
        q3+1 : q3);
endmodule

// ------------------------------------------------------------------
// system rewards
// ------------------------------------------------------------------

rewards "stime"
  !(q = Q & q1 = M & q2 = M & q3 = M) :
  q1 < M ? (1/mu1) :
  q2 < M ? (1/mu2) :
  q3 < M ? (1/mu3) :
    (1/mu1); // higher value
```

```
    // (1/mu2); // medium value
    // (1/mu3); // lower value
endrewards

rewards "qload"
  true : q+q1+q2+q3;
endrewards

rewards "TE_noswitch"
  true:
    pac1*q1     + pac2*q2     + pac3*q3 +
    pid1*(M-q1) + pid2*(M-q2) + pid3*(M-q3);
  [done1b] true : (pac1-pid1)*(1/mu1);
  [done2b] true : (pac2-pid2)*(1/mu2);
  [done3b] true : (pac3-pid3)*(1/mu3);
endrewards

rewards "TE_switchoff"
  true:
    pac1*q1     + pac2*q2     + pac3*q3;
  [done1b] true : pac1*(1/mu1);
  [done2b] true : pac2*(1/mu2);
  [done3b] true : pac3*(1/mu3);
endrewards

rewards "done"
  [done1a] true : 1;
  [done1b] true : 1;
  [done2a] true : 1;
  [done2b] true : 1;
  [done3a] true : 1;
  [done3b] true : 1;
endrewards


// ------------------------------------------------------------------
// InfiniteCommon.props
// ------------------------------------------------------------------

// rejection probability
"reject": S=? [ q = Q & q1 = M & q2 = M & q3 = M ] ;

// mean service time
"stime": (R{"stime"}=? [ S ])/(1-"reject") ;

// mean response time
"rtime": (R{"qload"}=? [ S ])/(lambda*(1-"reject"));

// waiting time
"wtime":  max(0, "rtime"-"stime");

// expected energy consumption
"TE_noswitch":  R{"TE_noswitch"}=?  [ S ] ;
"TE_switchoff": R{"TE_switchoff"}=? [ S ] ;
```

```
// the expected number of finished jobs per time unit
"done": R{"done"}=? [ S ] ;

// expected energy consumption per job
"AE_noswitch":  "TE_noswitch"/"done";
"AE_switchoff": "TE_switchoff"/"done";
```

# B. The Finite Source Models

## B.1. Class Queue

```
// -------------------------------------------------------------------
// FiniteClassHP.prism
// A scheduling model for computational clusters.
//
// The model is a finite source version of the "class queue" model with
// "high-performance priority" described in
//
//   Tien v. Do, Binh t. Vu, Xuan T. Tran, Anh P. Nguyen:
//   "A generalized model for investigating scheduling schemes in
//   computational clusters", Simulation Modelling Practice and
//   Theory, 37 (2013), 30-42.
//
// Authors: Berczes Tamas <berczes.tamas@inf.unideb.hu> and
// Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
//
// Copyright (C) 2014 Department of Informatics Systems and Networks,
// University of Debrecen, Debrecen, Hungary (http://irh.inf.unideb.hu)
// and Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// -------------------------------------------------------------------

// checking parameters: "sparse", "Gauss", default epsilon

// continuous time markov chain (ctmc) model
ctmc

// -------------------------------------------------------------------
// system parameters
// -------------------------------------------------------------------

// arrival rates
const double lambda;

// queue size
const int Q = 24;

// number of server classes and number of servers per class
const int K = 3;
const int M = 3;

// population size
```

```
const int N = 36;

// ranking based on performance
const double rp1 = 1.0;
const double rp2 = 0.82;
const double rp3 = 0.43;

// ranking based on energy efficiency
const double re1 = 0.64;
const double re2 = 0.66;
const double re3 = 1.0;

// active power consumption per class
const double pac10 = 1700;
const double pac20 = 1275;
const double pac30 = 457;

// idle power consumption per class
const double pid10 = 364;
const double pid20 = 331;
const double pid30 = 108;

// reduced power consumption in DVFS mode
const double pfs10 = 1169;
const double pfs20 = 881;
const double pfs30 = 317;

// ------------------------------------------------------------------
// system parameters according to chosen ranking
// ------------------------------------------------------------------

// service rates according to chosen ranking
const double mu1 = rp1;
const double mu2 = rp2;
const double mu3 = rp3;

// active power consumption according to chosen ranking
const double pac1 = pac10;
const double pac2 = pac20;
const double pac3 = pac30;

// idle power consumption according to chosen ranking
const double pid1 = pid10;
const double pid2 = pid20;
const double pid3 = pid30;

// reduced power consumption according to chosen ranking
const double pfs1 = pfs10;
const double pfs2 = pfs20;
const double pfs3 = pfs30;

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------
```

```
module Sources
  sources: [0..N] init N;
  [server]  sources > 0 -> lambda*sources : (sources' = sources-1);
  [done1] sources < N -> (sources' = sources+1);
  [done2] sources < N -> (sources' = sources+1);
  [done3] sources < N -> (sources' = sources+1);
endmodule

module System
  q1: [0..Q] init 0;
  q2: [0..Q] init 0;
  q3: [0..Q] init 0;

  [done1] q1 > 0 -> min(M,q1)*mu1 : (q1' = q1-1);
  [done2] q2 > 0 -> min(M,q2)*mu2 : (q2' = q2-1);
  [done3] q3 > 0 -> min(M,q3)*mu3 : (q3' = q3-1);

  [server] !(q1 = Q & q2 = Q & q3 = Q) ->
    (q1' =
       (q1 < M | (q2 >= M & q3 >= M & q1 <= q2 & q1 <= q3)) & q1 < Q ?
       q1+1 : q1) &
    (q2' =
       q1 >= M & (q2 < M | (q3 >= M & q2 < q1 & q2 <= q3)) & q2 < Q ?
       q2+1 : q2) &
    (q3' =
       q1 >= M & q2 >= M & (q3 < M | (q3 < q1 & q3 < q2)) & q3 < Q ?
       q3+1 : q3);
endmodule

// ------------------------------------------------------------------
// system rewards
// ------------------------------------------------------------------

rewards "stime"
  !(q1 = Q & q2 = Q & q3 = Q) :
  q1 < M ? (1/mu1) :
  q2 < M ? (1/mu2) :
  q3 < M ? (1/mu3) :
  q1 <= q2 & q1 <= q3 ? (1/mu1) :
             q2 <= q3 ? (1/mu2) :
                        (1/mu3) ;
endrewards

rewards "qload"
  true : q1+q2+q3;
endrewards

rewards "TE_noswitch"
  true:
    pac1*min(M,q1)   + pac2*min(M,q2)   + pac3*min(M,q3) +
    pid1*max(0,M-q1) + pid2*max(0,M-q2) + pid3*max(0,M-q3);
endrewards
```

```
rewards "TE_switchoff"
  true:
    pac1*min(M,q1)   + pac2*min(M,q2)   + pac3*min(M,q3);
endrewards

rewards "done"
  [done1] true : 1;
  [done2] true : 1;
  [done3] true : 1;
endrewards

// ------------------------------------------------------------------
// InfiniteClass.props
// ------------------------------------------------------------------

// rejection probability
"reject": S=? [ q1 = Q & q2 = Q & q3 = Q ] ;

// probability that all servers are busy
"busy": S=? [ q1 >= M & q2 >= M & q3 >= M ] ;

// mean service time
"stime": R{"stime"}=? [ S ] ;

// response time (of accepted jobs)
"qload":  R{"qload"}=? [ S ];
"rtime":  "qload"/(lambda*(1-"reject"));

// waiting time (of accepted jobs)
"wtime":  "rtime"-"stime";

// expected energy consumption
"TE_noswitch":  R{"TE_noswitch"}=?  [ S ] ;
"TE_switchoff": R{"TE_switchoff"}=? [ S ] ;

// the expected number of finished jobs per time unit
"done": R{"done"}=? [ S ] ;

// expected energy consumption per job
"AE_noswitch":  "TE_noswitch"/"done";
"AE_switchoff": "TE_switchoff"/"done";
```

## B.2. Separate Queue

```
// ------------------------------------------------------------------
// FiniteSeparateHP.prism
// A scheduling model for computational clusters.
//
// The model is a finite source version of the "separate queue" model with
// "high-performance priority" described in
//
//   Tien v. Do, Binh t. Vu, Xuan T. Tran, Anh P. Nguyen:
//   "A generalized model for investigating scheduling schemes in
```

```
//   computational clusters", Simulation Modelling Practice and
//   Theory, 37 (2013), 30-42.
//
// Authors: Berczes Tamas <berczes.tamas@inf.unideb.hu> and
// Wolfgang Schreiner <Wolfgang.Schreiner@risc.jku.at>
//
// Copyright (C) 2014 Department of Informatics Systems and Networks,
// University of Debrecen, Debrecen, Hungary (http://irh.inf.unideb.hu)
// and Research Institute for Symbolic Computation, Johannes Kepler
// University, Linz, Austria (http://www.risc.jku.at)
// ------------------------------------------------------------------

// check with sparse (1GB), "Jacobi", default epsilon

// continuous time markov chain (ctmc) model
ctmc

// ------------------------------------------------------------------
// system parameters
// ------------------------------------------------------------------

// arrival rates
const double lambda;

// queue size
const int Q = 4;

// population size
const int N = 36;

// number of server classes and number of servers per class
const int K = 3;
const int M = 3;

// ranking based on performance
const double rp1 = 1.0;
const double rp2 = 0.82;
const double rp3 = 0.43;

// ranking based on energy efficiency
const double re1 = 0.64;
const double re2 = 0.66;
const double re3 = 1.0;

// active power consumption per class
const double pac10 = 1700;
const double pac20 = 1275;
const double pac30 = 457;

// idle power consumption per class
const double pid10 = 364;
const double pid20 = 331;
const double pid30 = 108;
```

```
// reduced power consumption in DVFS mode
const double pfs10 = 1169;
const double pfs20 = 881;
const double pfs30 = 317;

// ------------------------------------------------------------------
// system parameters according to chosen ranking
// ------------------------------------------------------------------

// service rates according to chosen ranking
const double mu1 = rp1;
const double mu2 = rp2;
const double mu3 = rp3;

// active power consumption according to chosen ranking
const double pac1 = pac10;
const double pac2 = pac20;
const double pac3 = pac30;

// idle power consumption according to chosen ranking
const double pid1 = pid10;
const double pid2 = pid20;
const double pid3 = pid30;

// ------------------------------------------------------------------
// system model
// ------------------------------------------------------------------

module Sources
  sources: [0..N] init N;
  [server]   sources > 0 -> lambda*sources : (sources' = sources-1);
  [done11] sources < N -> (sources' = sources+1);
  [done12] sources < N -> (sources' = sources+1);
  [done13] sources < N -> (sources' = sources+1);
  [done21] sources < N -> (sources' = sources+1);
  [done22] sources < N -> (sources' = sources+1);
  [done23] sources < N -> (sources' = sources+1);
  [done31] sources < N -> (sources' = sources+1);
  [done32] sources < N -> (sources' = sources+1);
  [done33] sources < N -> (sources' = sources+1);
endmodule

module Servers
  q11: [0..Q] init 0;
  q12: [0..Q] init 0;
  q13: [0..Q] init 0;

  q21: [0..Q] init 0;
  q22: [0..Q] init 0;
  q23: [0..Q] init 0;

  q31: [0..Q] init 0;
  q32: [0..Q] init 0;
  q33: [0..Q] init 0;
```

```
[done11] q11 > 0 -> mu1 : (q11' = q11-1);
[done12] q12 > 0 -> mu1 : (q12' = q12-1);
[done13] q13 > 0 -> mu1 : (q13' = q13-1);

[done21] q21 > 0 -> mu2 : (q21' = q21-1);
[done22] q22 > 0 -> mu2 : (q22' = q22-1);
[done23] q23 > 0 -> mu2 : (q23' = q23-1);

[done31] q31 > 0 -> mu3 : (q31' = q31-1);
[done32] q32 > 0 -> mu3 : (q32' = q32-1);
[done33] q33 > 0 -> mu3 : (q33' = q33-1);

[server] !(q11 = Q & q12 = Q & q13 = Q &
           q21 = Q & q22 = Q & q23 = Q &
           q31 = Q & q32 = Q & q33 = Q ) ->
  (q11' =
       q11 <= q11 & q11 <= q12 & q11 <= q13 &
       q11 <= q21 & q11 <= q22 & q11 <= q23 &
       q11 <= q31 & q11 <= q32 & q11 <= q33 &
       q11 < Q ? q11+1 : q11 ) &
  (q12' =
       q12 <  q11 & q12 <= q12 & q12 <= q13 &
       q12 <= q21 & q12 <= q22 & q12 <= q23 &
       q12 <= q31 & q12 <= q32 & q12 <= q33 &
       q12 < Q ? q12+1 : q12 ) &
  (q13' =
       q13 <  q11 & q13 <  q12 & q13 <= q13 &
       q13 <= q21 & q13 <= q22 & q13 <= q23 &
       q13 <= q31 & q13 <= q32 & q13 <= q33 &
       q13 < Q ? q13+1 : q13 ) &

  (q21' =
       q21 <  q11 & q21 <  q12 & q21 <  q13 &
       q21 <= q21 & q21 <= q22 & q21 <= q23 &
       q21 <= q31 & q21 <= q32 & q21 <= q33 &
       q21 < Q ? q21+1 : q21 ) &

  (q22' =
       q22 <  q11 & q22 <  q12 & q22 <  q13 &
       q22 <  q21 & q22 <= q22 & q22 <= q23 &
       q22 <= q31 & q22 <= q32 & q22 <= q33 &
       q22 < Q ? q22+1 : q22 ) &
  (q23' =
       q23 <  q11 & q23 <  q12 & q23 <  q13 &
       q23 <  q21 & q23 <  q22 & q23 <= q23 &
       q23 <= q31 & q23 <= q32 & q23 <= q33 &
       q23 < Q ? q23+1 : q23 ) &

  (q31' =
       q31 <  q11 & q31 <  q12 & q31 <  q13 &
       q31 <  q21 & q31 <  q22 & q31 <  q23 &
       q31 <= q31 & q31 <= q32 & q31 <= q33 &
       q31 < Q ? q31+1 : q31 ) &
```

```
    (q32' =
         q32 <  q11 & q32 <  q12 & q32 <  q13 &
         q32 <  q21 & q32 <  q22 & q32 <  q23 &
         q32 <  q31 & q32 <= q32 & q32 <= q33 &
         q32 < Q ? q32+1 : q32 ) &
    (q33' =
         q33 <  q11 & q33 <  q12 & q33 <  q13 &
         q33 <  q21 & q33 <  q22 & q33 <  q23 &
         q33 <  q31 & q33 <  q32 & q33 <= q33 &
         q33 < Q ? q33+1 : q33 ) ;
endmodule

// ------------------------------------------------------------------
// system rewards
// ------------------------------------------------------------------

rewards "stime"
    !(q11 = Q & q12 = Q & q13 = Q &
      q21 = Q & q22 = Q & q23 = Q &
      q31 = Q & q32 = Q & q33 = Q ) :
    min(q11, q12, q13) <=
      min(q21, q22, q23,
          q31, q32, q33) ? (1/mu1) :
    min(q21, q22, q23) <=
      min(q31, q32, q33) ? (1/mu2) : (1/mu3);
endrewards

rewards "sources"
  true : sources;
endrewards

rewards "TE_noswitch"
  q11 = 0 : pid1;
  q11 > 0 : pac1;
  q12 = 0 : pid1;
  q12 > 0 : pac1;
  q13 = 0 : pid1;
  q13 > 0 : pac1;
  q21 = 0 : pid2;
  q21 > 0 : pac2;
  q22 = 0 : pid2;
  q22 > 0 : pac2;
  q23 = 0 : pid2;
  q23 > 0 : pac2;
  q31 = 0 : pid3;
  q31 > 0 : pac3;
  q32 = 0 : pid3;
  q32 > 0 : pac3;
  q33 = 0 : pid3;
  q33 > 0 : pac3;
endrewards

rewards "TE_switchoff"
  q11 > 0 : pac1;
```

```
  q12 > 0 : pac1;
  q13 > 0 : pac1;
  q21 > 0 : pac2;
  q22 > 0 : pac2;
  q23 > 0 : pac2;
  q31 > 0 : pac3;
  q32 > 0 : pac3;
  q33 > 0 : pac3;
endrewards

rewards "done"
  [done11] true : 1;
  [done12] true : 1;
  [done13] true : 1;
  [done21] true : 1;
  [done22] true : 1;
  [done23] true : 1;
  [done31] true : 1;
  [done32] true : 1;
  [done33] true : 1;
endrewards

// ------------------------------------------------------------------
// InfiniteSeparate.props
// ------------------------------------------------------------------

// rejection probability
"reject": S=? [
      q11 = Q & q12 = Q & q13 = Q &
      q21 = Q & q22 = Q & q23 = Q &
      q31 = Q & q32 = Q & q33 = Q ] ;

// mean service time (not considering rejection)
"stime": R{"stime"}=? [ S ] ;

// mean service time
"stime0": "stime"/(1-"reject") ;

// number of currently not serviced sources
"sources": R{"sources"}=? [ S ];

// response time (not considering rejection)
"rtime":  (N/"sources"-1)/lambda;

// response time
"rtime0": "rtime"/(1-"reject");

// waiting time (not considering rejection)
"wtime":  "rtime"-"stime";

// waiting time
"wtime0":  "wtime"/(1-"reject");

// expected energy consumption
```

```
"TE_noswitch":  R{"TE_noswitch"}=?  [ S ] ;
"TE_switchoff": R{"TE_switchoff"}=? [ S ] ;

// the expected number of finished jobs per time unit
"done": R{"done"}=? [ S ] ;

// expected energy consumption per job
"AE_noswitch":  "TE_noswitch"/"done";
"AE_switchoff": "TE_switchoff"/"done";
```