Discrete-Event Modeling of a High-Performance Computing Cluster with Service Rate Control

Alexander Rumyantsev, Taisia Morozova
Institute of Applied Mathematical Research, Karelian Research Centre of RAS
Petrozavodsk State University
Petrozavodsk, Russia
ar0@krc.karelia.ru, tiamorozova@mail.ru

Robert Basmadjian
Universität Passau
Passau, Germany
robert.basmadjian@uni-passau.de

***Abstract*—We present a stochastic recursion based discrete-event model of a high-performance computing cluster with service rate switching capabilities. The model is easily adopted to many common settings of modern supercomputers, such as specific scheduling disciplines and various control policies. We also provide some illustrative numerical experiments and discuss further generalizations of the model.**

I. Introduction

Energy-aware computing started with the advent of mobile devices that had limited power availability. However, in the last decade, energy-awareness became an important topic in GreenIT. This is because, reports about energy demand of data centres have shown astonishing figures, making data centres the biggest consumers in the ICT sector [1]. For instance, in 2014 the energy demand of data centres in the U.S. was about 70 TWh which represented about 2% of the overall energy consumption [2]. Also, it is estimated that the energy demand of data centres in Western Europe to exceed 100 TWh by 2020 [3].

In contrast to other computing paradigms (e.g. cloud computing and its concept of virtual machines), in high-performance computing computations are executed on dedicated physical resources (e.g. servers). Consequently, among different types of data centres, HPC ones consume the most due to the growing demand for higher performance and leaving only limited possibilities for energy reduction. Table I presents recent worse power consumption numbers from the TOP500 list of HPC systems [4], where QUARTETTO which is used for university purposes has about 20 MW of power demand!

The energy reduction in HPC data centres can be achieved through software and hardware solutions. Software solutions comprise of intelligent energy management systems, the basis of which is the provision of energy-aware job scheduling algorithms. The main objective here is to re-schedule jobs such that idle servers can be put into standby mode to save energy. From hardware perspective, dynamic voltage and frequency scaling (DFVS) technique [5] was proposed which has the aim of dynamically switching the frequency and voltage of the processor. Major hardware vendors like Intel and AMD have developed their own DVFS implementations, namely SpeedStep [6] and CoolnQuiet [7] respectively. The main objective behind DVFS is to conserve power demand and hence reduce the amount of heat generated by the chip. It has numerous application scenarios such as mobile devices to preserve battery, computing systems to decrease their cooling cost and noise, and data centres to reduce the power demand at the times of low workload utilization.

TABLE I. Power consumption of HPC systems [4]

System	Power (kW)	Power Efficiency (GFlops/watts)
Lenovo ThinkServer RD650 (China)	3750	0.183
Cluster Platform DL380p Gen8 (US)	3840	0.163
Lenovo NeXtScale nx360M5 (China)	3510	0.157
Lenovo NeXtScale nx360M5 (China)	3645	0.155
QUARTETTO (Japan)	19431	0.052

A. High-performance computing

A high-performance computing (HPC) is a computational paradigm build upon several nodes in such a way to perform parallel computing of a single/several tasks by multiple central processing units (CPUs) and fast inter-node data exchange. Parallelism of a task execution allows to reduce computing time by adopting special algorithmic and programming techniques (resulting in some cases in significant increase in implementation complexity). Being originally a highly homogeneous and well structured resource, modern HPC implements heterogeneity at various levels, including the computing units (CPUs or the so-called general purpose graphical processing units), memory, storage etc.

In general, the HPC is used as a shared resource with multiple users, submitting their tasks (software chunks) into the central queue(s) managed by the so-called queue management software (QM). A conventional QM dispatches the tasks to the computing nodes (servers) according to prescribed scheduling policy, queuing discipline, tasks requirements and limitations, predefined reservations of the computing capacity (such as scheduled downtimes) and many other parameters. However, the key feature of the HPC system is the possibility of a task to occupy multiple servers at once, starting and ending the computation at reserved resources simultaneously. At the same time, an HPC is highly expensive in terms of capital and operational expenses, and thus should be planned in advance, and managed with high accuracy.

The possibility to reduce operational expenses is related to embedded mechanisms of energy saving, present in a general HPC. Such mechanisms allow to vary the frequency (by means of the so-called dynamic voltage and frequency scaling, DVFS) of each CPU, as well as temporarily put some of the computing nodes into low power state (the so-called sleep states, including the sleep and hibernate, if available). However, reducing the energy consumption by both methods implies significant degradation of the performance of a system in terms of delay, sojourn time etc. Therefore, obtaining a solution for various constrained optimization problems related to energy efficiency is crucial for efficient system management.

B. Related work

Analysis of a multiserver system with simultaneous service (however, not in the context of HPC modeling), to the best of our knowledge, was first performed in the work [8] under several severe restrictions. A detailed analytical study of a queuing system with two servers and simultaneous server occupation/release by a customer was performed in [9] by means of the so-called system point approach. This analysis was extended several decades later in the work [10], where also the simulation issues were addressed. We also note the works [11], [12], addressing the stability criterion of a simultaneous service multiserver system with arbitrary number of servers.

Among the first works related to modeling of a multiserver system with simultaneous service should be mentioned the work [13], where the so-called matrix-analytic approach was applied to obtaining performance metrics. The work [14] addressed the HPC model as an extension of the celebrated Kiefer–Wolfowitz workload recursion [15]. A discrete-event model of an HPC was first presented recently in [16]. Many aspects related to modeling the driving sequences of an HPC, such as task interarrival times and service times, daily load cycles etc. may be found in [17], [18].

In order to reduce energy consumption, research for HPC systems has mainly focused on the following topics:

- Energy-efficient or energy proportional hardware
- DVFS technique
- Shutting down hardware components at low system utilization
- Power Capping
- Thermal Management

By designing energy-efficient hardware, the components themselves are energy-aware, i.e., they consume less energy than the standard ones. Significant improvements have been achieved with respect to idle power consumption of servers thanks to advanced techniques of shutting down (e.g. C states of processors) certain components of a computing node.

Since the processor's power demand is a significant portion of the total system's power consumption (about 50 % under load [19]), the DVFS technique is used for controlling the CPU power. By running a processor at lower frequency/voltage energy savings can be achieved, but the job execution time is increased. In our previous work [20], we analyzed the impact of DVFS-enabled processors using M/M/1-FCFS queing systems and showed that Linux ondemand governor does not perform in optimal way, taking into account both power demand and performance. Pierson et al. [21] model service placement on DVFS-enabled hosts as a Mixed-Integer Linear Program. Power consumption of hosts for different voltage / frequency combinations of DVFS is measured in idle and in loaded state. Numerical results show that using more than two DVFS power states leads to marginal improvement in the job placing problem only. The main disadvantage of this approach is that service performance is only considered by defining service resource requirements. In [22], Markov modelling was used for DVFS in multi-core processors. It works by detecting phases in the workload subjected by an application. These phases have corresponding performance and power attributes. Once the phases have been detected, a static power schedule is constructed and frequency scaling is performed based on the schedule. This strategy works best for predictable and iterative workload.

In a typical HPC data centre, servers consume nearly as much energy in idle state as when running an application (e.g. 60-70%). At phases of low system utilization, some servers or their components could be shut down or switched to a low-power state. This strategy thus tries to minimize the number of active servers of a system while still satisfying incoming application requests. Since this approach is highly dependent on the workload, the challenge is when to shut down components and how to provide a suitable job slowdown value. In this regard, we mention several works studying energy efficiency of datacenters, as well as multiserver systems [23], [24], [25], [26], [27].

However, to the best of our knowledge, this work is a first attempt to incorporate both the distinguished feature of an HPC (the simultaneous service) and energy efficiency in a discrete-event based simulation model under general assumptions.

C. Model description

Hereafter we consider an HPC from as a queueing-theoretical model, and in this regard we adopt the more common notion of *server* to address a minimal addressable computing unit, which might be a CPU or computing node node, and *customer* to denote a computational task in the queue, dispatched by a QM. Thus, the model of an HPC is a special subclass of multiserver queueing models.

Consider a c-server single queue system with an input of customers, ith customer arriving at epoch t_i has two characteristics: job amount $S_i \in \mathbb{R}_+$ and required number of servers $N_i \in \{1, \dots, c\}$ being occupied and released simultaneously once the job amount is completed. Each server is capable of serving customers at various rates, and in the sequel wolog. two rates $r_L < r_H$ that may be switched at each server in no time are considered. The following practically motivated assumption is implied: all c servers switch the service rate simultaneously (note that in other case the service time of a customer will be determined by the slowest server). To eliminate the language ambiguity, we say that i is the *label* of customer i (the number of customer in the order of arrival). We also assume, that each server working at rate r_x consumes the energy $e(r_x)$ per unit time, $x = \{L, H\}$, while the server consumes $e_0(r_x)$ being idle.

The idea of stochastic modeling of an HPC is to obtain the key performance measures in a simulation given the driving sequences $\{t_i, S_i, N_i\}, i \geqslant 1$. Hereafter is assumed that the system state does not infer the driving sequences, however, it is relatively easy to eliminate this assumption by implying some moderate dependency (such as dependency on a current and fixed amount of previous states). In this paper the stochastic model of an HPC is obtained by means of discrete-event simulation having the following per-customer performance measures in consideration:

- delay (time from arrival to service starting epoch);
- sojourn time (time from arrival to departure), also known as latency, or responce time;
- service time (time from service starting epoch to departure);
- energy budget (amount of energy consumed for service);
- slowdown (responce time divided by service time).

We also consider the following per-system performance measures:

- potential load (potential job completion rate);
- effective load (job completion rate from busy servers);
- job wasting (difference between potential and effective loads);
- energy consumption (amount of energy consumed per time interval).

All the respective performance measures might be averaged in order to obtain mean performance characteristics. However, the distribution of a performance measure might be more valuable for decision making. A more detailed study of performance metrics of an HPC might be found in [28].

We stress, that the model discussed in this paper is an extension of earlier presented HPC model introduced in [16]. The stochastic recursions obtained in this paper are applicable to more general settings and more queueing disciplines, and in concluding section we discuss the possibilities of further extensions of the model.

The structure of the paper is as follows. In Section II we briefly introduce the model for simplified settings of FCFS queueing discipline. In Section III we demonstrate possible extensions of the model w.r.t. various queueing disciplines as well as various control policies. We present some numerical experiments results for illustration purposes of the method in Section IV. In Section V we conclude the paper by demonstrating further possible model extensions.

II. FCFS Randomized Asynchronous Control System

In this section additionally the following restrictions are placed:

- customers do not share servers (thus the service time of customer i is in the interval $[S_j/r_H, S_j/r_L]$),
- customers start being served in the order of arrival (FCFS),
- the service rate switching occurs only at customer arrival and service completion epochs by randomized rule.

We adopt the following randomized switching rule. At an arrival (departure) an independent coin is flipped, and with probability p_H (p_L) the service rate increase (decrease) occurs.

A. Key Recurrent Relations

To set up discrete event simulation we first define set of possible events in the system. By assumption, the control epochs are co-located with customer arrival/departure epochs. Thus, the set of possible key events is restricted to arrival and departure events only. At the time T_i of ith key event occurrence we define the *system state* as a tuple $\{M_i; I_i^A; I_i^D; \{B_i(m), m \in M_i\}; R_i\}$, where

M_i	— (ordered) set of labels of customers present in the system;
I_i^x	— indicator of the event $x \in \{\text{A, D}\}$ (arrival, departure), $I_i^A + I_i^D = 1$;
$B_i(j)$	— remaining job amount for customer $j \in M_i$;
R_i	— service rate.

Let $M_i = \{m_{i,1}, m_{i,2}, \dots\}$ be the set of customer labels at T_i, where $m_{i,1} < m_{i,2} < \dots$ are customer labels in the order of customer arrival times. Then the set of labels of customers being served is the following subset of M_i:

$$\mathcal{M}_i = \max_{k \geqslant 1}\{m_{i,1}, \dots, m_{i,k} : \sum_{t=1}^{k} N_{m_{i,t}} \leqslant c\} \subseteq M_i. \quad (1)$$

The epoch T_{i+1} is then recursively defined as follows

$$T_{i+1} = \min\left\{t_{A(T_i)+1}, T_i + B_i(m_{i,j})/R_i\right\}, m_{i,j} \in \mathcal{M}_i, \quad (2)$$

where the counting process $A(t) = k$, $t_k \leqslant t < t_{k+1}$, $k \geqslant 1$ is the number of arrivals up to time t; and $T_i + B_i(m_{i,j})/R_i$ is the potential departure time of the customer $m_{i,j}$ being served (if any) at instant T_i (conventionally, $B_i(\varnothing) = \infty$). Define the inter-event time

$$\tau_i = T_{i+1} - T_i, \ i \geqslant 1.$$

Note that since the arrival and departure epochs do not coincide, we automatically assign $I_{i+1}^A = 1$ if T_{i+1} is an arrival, and $I_{i+1}^D = 1$ otherwise, such that $I_{i+1}^A + I_{i+1}^D = 1$.

For convenience denote the possibly departing task as the task with minimal remaining processing time:

$$\delta(\mathcal{M}_i) := \arg\min_{m \in \mathcal{M}_i}\{B_i(m)\}.$$

The set of numbers of customers present in the system at T_{i+1} is changed at key epochs as follows

$$M_{i+1} = M_i \cup \{A(T_{i+1}) : I_{i+1}^A = 1\} \backslash \{\delta(\mathcal{M}_i) : I_{i+1}^D = 1\}, \quad (3)$$

where the notion $\{A : B\}$ denotes a conditional set, empty if the condition is not satisfied. Then the remaining work is evolved as follows

$$\begin{aligned} B_{i+1}(m) = B_i(m) - \tau_i R_i I\{m \in \mathcal{M}_i \cap M_{i+1}\} \\ + S_m I\{m = A(T_{i+1})\} I_{i+1}^A, \end{aligned} \quad (4)$$

where $B_{i+1}(\delta(\mathcal{M}_i))$ is nothing, if $I_{i+1}^D = 1$. To explain (4) we recall, that the remaining job is decreased by the amount of $\tau_i R_i$ only for customers being served, and is initialized by the initial (given) job amount for a newly arrived customer, if any.

Finally, the speed R_{i+1} is given as follows

$$\begin{aligned} R_{i+1} = r_L I^D_{i+1}\beta(p_L) + r_H I^A_{i+1}\beta(p_H) + \\ + R_i(1 - I^D_{i+1}\beta(p_L) - I^A_{i+1}\beta(p_H)), \end{aligned} \tag{5}$$

where $\beta(p)$ is an independent Bernoulli trial with success probability p.

It remains to define the recursion basis at time epoch $T_1 = t_1 = 0$:

$$M_1 = \{1\},\ B_1(1) = S_1,\ I^A_1 = 1,$$

$$R_1 = r_H\beta(p_H) + r_L(1 - \beta(p_H))$$

B. Performance Measures

To derive the per-customer performance measures, we first define some key time epochs of some fixed customer j. Recall that t_j is the arrival time of jth customer. Then the service starting time of customer j is defined as

$$t^{(S)}_j = \min\{T_i \geqslant t_j : j \in \mathcal{M}_i\}. \tag{6}$$

We note that the time $t^{(S)}_j \geqslant t_j$ is not available at customer arrival time, and the equation (6) takes into consideration the *future* of the process, thus, this time epoch may be defined only after obtaining the results of a simulation run. A possible alternative for implementation might be inline auxiliary control of the set $\mathcal{M}_i$ at each time epoch T_i, which would give the following recurrent relation

$$t^{(S)}_j = \begin{cases} T_i, \text{if } I^D_i = 1, j \in \mathcal{M}_i \backslash \mathcal{M}_{i-1}\},\ i \geqslant 2, \\ t_j, \text{if } I^A_i = 1, j \in \mathcal{M}_i \backslash \mathcal{M}_{i-1}\},\ i \geqslant 2. \end{cases} \tag{7}$$

We stress, that the service starting time of a customer j corresponds to either its arrival instant, or the departure instant of some preceding customer, and moreover, a single departure epoch might correspond to multiple service starting times. Similarly, the departure time is defined as

$$t^{(D)}_j := \min\{T_i \geqslant t_j : j \notin \mathcal{M}_i\}. \tag{8}$$

At that, the per-customer performance measures follow immediately.

- delay $t^{(S)}_j - t_j$;
- sojourn time $t^{(D)}_j - t_j$;
- service time $t^{(D)}_j - t^{(S)}_j$;
- slowdown $(t^{(D)}_j - t_j)/(t^{(S)}_j - t_j)$.

However, defining an energy budget of customer j requires to involve the trajectory of the system between service starting and departure epochs. It follows that energy budget may be defined as

$$N_j \sum_{i:t^{(S)}_j \leqslant T_i < t^{(D)}_j} e(R_i)\tau_i. \tag{9}$$

We note, that in fact, the service starting epoch of a customer, defined in (6), requires to know the future of the process.

Now we define some auxiliary quantities derived at time epoch T_i:

- workload (remaining job in the system): $W_i = \sum_{m \in M_i} B_i(j)$;
- number of customers in the system: $\nu_i = |M_i|$;
- number of customers in the queue: $Q_i = |M_i \backslash \mathcal{M}_i|$;
- number of busy servers: $\varphi_i = \sum_{m \in \mathcal{M}_i} N_m,\ i \geqslant 1$ (with obvious convention $\sum_{\varnothing} = 0$).

We are ready to derive the per-system performance measures at time epoch T_i:

- potential load: $R_i(cI\{Q_i > 0\}) + \varphi_i I\{Q_i = 0\})$, where $I\{\cdot\}$ is the indicator function;
- effective load: $R_i\varphi_i$.

We turn to time interval-based perfomance measures of our model, which are defined at time epoch T_i and are related to the time interval $[T_i, T_{i+1})$. First we derive the job wasting. The job wasting phenomenon is related to the non work-conserving property of a queueing discipline under study. When the task first waiting in the queue requires more servers than are available at a particular time epoch, other tasks should not overtake the first one, even if they require less servers than are available. Thus the idle servers are wasting energy without serving any customer. Note that this feature is in contrast to a conventional multiserver queueing system. Since this characteristic is related to time period, we first recall that the number of idle servers at time T_i is $c - \varphi_i$. Thus, the job wasting equals

$$(c - \varphi_i)\tau_i R_i I\{Q_i > 0\},\ i \geqslant 1. \tag{10}$$

Finally, we derive the energy consumption (per unit time) for time interval $[T_i, T_{i+1})$. First we note, that once T_i is a departure leaving the system empty, then T_{i+1} is necessary an arrival, and thus the considered interval is idle time of the whole system. Thus

$$E_i = \varphi_i e(R_i) + (c - \varphi_i)e_0(R_i). \tag{11}$$

It remains to note, that in order to define the average per-customer performance measure, it is sufficient to calculate the sample average from all the customers in a simulation run. Obtaining average per-system performance measures requires appropriate aggregation over the time intervals, and normalization by overall simulation duration T. We demonstrate this aggregation by obtaining the mean number of customers in the system, and the mean energy consumption (other performance measures might be obtained similarly):

$$\overline{\mathcal{V}} = \frac{1}{T}\sum_{i:T_i \leqslant T} \tau_i\nu_i,$$

$$\overline{\mathcal{E}} = \frac{1}{T}\sum_{i:T_i \leqslant T} \tau_i E_i. \tag{12}$$

III. Model Flexibility

In this section we consider serveral straightforward extensions of the model presented in Section II. We embed some realistic scenarios which might give additional insight, however, we discuss even wider extensions of the model in Conclusion.

A. Rate switching policy

Assume the following service rate switching policy, known as hysteretic control. At arrival/departure instance T_i, the service rate is switched to r_H (r_L, respectively) provided the number of customers in the system ν_i exceeds k_H (becomes less or equal, than $k_L \leqslant k_H$). Thus, the only required change in the model is the service rate recursion relation (5), which transforms to

$$\begin{aligned} R_{i+1}^{HC} = I_{i+1}^{D}(r_L I\{\nu_i = k_L+1\} + R_i^{HC} I\{\nu_i \neq k_L+1\}) + \\ + I_{i+1}^{A}(r_H I\{\nu_i = k_H\} + R_i^{HC} I\{\nu_i \neq k_H\}), \end{aligned}$$

where we put a superscript HC to denote the hysteretic control policy. We recall that if $k_L = k_H$, the policy turns out to have a single threshold, and, moreover, the system without service rate control is related to $k_L = k_H = 0$ or $k_L = k_H = \infty$.

B. Queueing discipline

Now we violate the assumption of FCFS service discipline to demonstrate, how can various queueing disciplines be incorporated in the model. For comparison we take the widely used queueing disciplines, including the EASY Backfill [29]. We note that only the recursive relation (1) is to be changed at this stage. Thus, we provide the new relations for replacement of (1), and indicate it with an appropriate superscript.

Last Come First Served (LCFS) discipline allows the most recently arrived customers to enter service. Thus,

$$\mathcal{M}_i^{LCFS} = \max_{k \geqslant 0}\{m_{i,\nu_i}, \dots, m_{i,\nu_i-k} : \sum_{t=\nu_i-k}^{\nu_i} N_{m_{i,t}} \leqslant c\}.$$

Note that it follows from (4) that once a new customer arrives, it may postpone the computation of an earlier customer being served, thus this realization is related to the so-called Preemptive-Resume scheme of LCFS.

Longest Processing Time First (LPF) discipline allows the customers with the biggest remaining jobs to start service, once the resources are available. Thus, to define the rule for selecting the customers being served at epoch T_i, we first define an ordering $\pi_i = (\pi_{i,1}, \dots, \pi_{i,\nu_i}), \pi_{i,j} \in M_i$ such that $B_i(\pi_{i,j}) \geqslant B_i(\pi_{i,j+1})$ for $j \geqslant 1$. Thus, π_i is a permutation ordering the customers in M_i according to their remaining processing times. To prevent newly arriving customers to enter service, we split the formula with respect to indicator value I_{i+1}^D, allowing the ordering to be applied only at departure instants:

$$\begin{aligned} \mathcal{M}_{i+1}^{LPF} = \max_{k \geqslant 0}\{\pi_{i,1}, \dots, \pi_{i,k} : \sum_{t=1}^{k} N_{\pi_{i,t}} \leqslant c, I_{i+1}^D = 1\} \cup \\ \cup \{\mathcal{M}_i^{LPF} : I_{i+1}^A = 1\}. \end{aligned}$$

The queueing discipline Shortest Processing Time First may be analyzed similarly, but the permutation π_i should order the components of M_i according to increasing of $B_i(\pi_{i,j}),\ j \geqslant 1$.

Now we turn to Backfill scheduling discipline. The so-called EASY Backfill is a heuristic scheduling strategy that allows small customers to overtake the first customer waiting in the queue, provided they will not violate the reservation of such a customer (i.e. will these overtaking customers have to depart before the oldest waiting customer starts its service). Thus, this strategy only differs from FCFS at the event $\{Q_i > 1\}$. Below we consider such an event. In order to obtain the backfilling event, we need to know the current schedule. In fact, the schedule of the tasks is available at each time epoch T_i, and the so-called workload vector (remaining jobs at each server in ascending order, *including the future ones*) may be constructed. We briefly recall the procedure of obtaining the workload vector, given in [16], only noting that due to service rate switching, the workload vector is given in terms of jobs, but not times. Let

$$Z = \{x \in \mathbb{R}_+^c : x_1 \leqslant \dots \leqslant x_c\}$$

be the set of c-dimensional vectors with nonnegative components. For each such a vector, define a *scheduling mapping* $\sigma : Z \times \{1, \dots, c\} \times \mathbb{R}_+ \to Z$. We stress, that this mapping depends on the queueing discipline. Let $W \in Z$ be some workload vector, and assume that some customer requiring $n \in \{1, \dots, c\}$ servers for time $b \in \mathbb{R}_+$ is to be scheduled together with W. For the FCFS model,

$$\begin{aligned} &\sigma := \sigma[W, n, b] = \\ &\mathcal{R}(\underbrace{W_n + b, \dots, W_n + b}_{n}, W_{n+1}, \dots, W_c), \end{aligned} \tag{13}$$

where $\mathcal{R}(\cdot)$ puts the components of a vector in ascending order. Thus, the customer b, n will be served by n least busy servers, and W_n is the waiting time of this fixed customer. To obtain the workload vector at time T_i, we iteratively apply the mapping σ to all the customers $m_{i,1}, \dots, m_{i,\nu_i} \in M_i$ by the following recurrent relation:

$$\mathcal{W}_i(k) = \sigma[\mathcal{W}_i(k-1), B_i(m_{i,k}), N_{m_{i,k}}], \tag{14}$$

for $k = 1, \dots, \nu_i$, starting from $\mathcal{W}_i(0) := \mathbf{0} \in Z$. In fact, the customer $m_{i,k} \in M_i$ is planned to be served after all the earlier arrived customer will be served, with respect to the required number of servers for this customer. Thus, the residual job $\alpha(m_{i,k})$ that has to be complete before a customer $m_{i,k}$ may enter the service, is given by the $N_{m_{i,k}}$-th component of vector $W_i(k-1)$, i.e.

$$\alpha(m_{i,k}) = [\mathcal{W}_i(k-1)]_{N_{m_{i,k}}}. \tag{15}$$

Now let $Q_i > 1$. Define $\psi_i(k) = \psi_{i,1}, \dots, \psi_{i,k} \in M_i \backslash \mathcal{M}_i$ be the subsequence of labels of customers waiting in the queue at time T_i (if any), such that

$$B_i(\psi_{i,j}) < \alpha(m_{i,\nu_i - Q_i + 1}),$$

i.e. the subsequence of customers waiting in the queue, which, given sufficient resources, would depart before the first customer waiting in the queue could start its service. Thus, we first assume FCFS and perform the scheduling, and once we know the job amount that has to be complete before a customer waiting in the queue may start its service, we perform the backfilling. The appropriate customer selection rule is

$$\mathcal{M}_i^{BF} = \max_{k \geqslant 1}\{\psi_{i,1}, \dots, \psi_{i,k} : \sum_{t=1}^{k} N_{\psi_{i,t}} < c\}.$$

C. Workload-aware switching

In previous sections we defined the service rate switching discipline with respect to the number of customers in the system. Now we demonstrate another type of control related to the workload (remaining work). Note that this approach requires to know the amount of work for each customer in advance, which (in practical case) may be available only as an upper estimate, known as *deadline*. We briefly recall a workload-based hysteretic policy first presented in [16]. When the workload in the system hits the higher threshold w^H (lower w_L), the service rate is switched to r_H (r_L). This extension requires to define additional event type, since the rate switching to r_L may occur at intermediate times, not related to arrivals/departures (whereas switching to r_H epoch necessary coincides with an arrival). Let I_i^S be the indicator of switching (from r_H to r_L) at time epoch T_i. Then the recursion (2) should be modified as follows

$$T_{i+1} = \min\Big\{ t_{A(T_i)+1}, T_i + \min_{m \in \mathcal{M}_i} \frac{B_i(m)}{R_i}, \\ T_i + \frac{W_i - k_L}{I\{W_i > k_L\} R_i} \Big\}.$$

Thus, the $i+1$th time epoch is defined as the nearest event from the potential departure, potential switching to rate r_L and arrival events. The corresponding change in (5) due to hysteretic rate switching is as follows

$$\begin{aligned} R_{i+1} = & r_L I\{W_{i+1} \leqslant k_L\} + r_H I\{W_{i+1} > k_H\} \\ & + R_i I\{k_L < W_{i+1} \leqslant k_H\}. \end{aligned}$$

We recall that the workload W_i is defined with respect to (4).

IV. Experiments

Implementation of the stochastic recursions model is performed in `hpcwld` package for R statistical software available at CRAN package repository. This package implements the following functionality:

a. performance evaluation of a stochastic model of an HPC;
b. fast delay computation for every customer;
c. stability criterion validation.

Hereafter we present some numerical experiments for illustration purposes. The experiments are based on the workload data obtained from the high-performance cluster of the High-performance Data Center of the Karelian Research Centre of Russian Academy of Sciences were taken as the input source. This dataset, included in the `hpcwld` package, contains data related to the tasks completed during the timespan from 02.04.2011 to 16.04.2012, a total of 9389 tasks. The data contains customer interarrival times, service times, cores (servers) that tasks used, and delays experienced by customers (submitted tasks), all in seconds, as recorded by the SLURM queue management software. Note that due to technical limitations, the data should be cleared before usage (to eliminate erroneous records, such as negative number of cores used), and appropriately modified (e.g. add several milliseconds random delay for consecutive tasks that were submitted with zero interarrival times). The corresponding values of constants during simulation were as follows: low and high speeds of servers (r_L and r_H) equal accordingly to 2.0 and 2.66 (GHz), energy consumption at the idle mode and at the working mode on low and high speeds accordingly $e_0(r_L) = 0.014, e_0(r_H) = 0.018, e(r_L) = 0.041, e(r_H) = 0.034$. These parameters correspond to the Quad-Core Intel Xeon 5430 2,66 GHz used in the HPC, as well as the energy consumption of the whole system per core per unit time.

The model of a multiserver system under consideration has $c = 80$ servers (corresponding to CPU cores of the HPC). The interarrival and service times, as well as number of servers required, were taken from the aforementioned dataset, however, for illustrative purposes we take various time intervals of the simulation. Also we selected $k_L = 10^5$ and $k_H = 3 \cdot 10^5$ as the values of hysteretic workload-aware rate switching thresholds. Workload data depicted at Fig. 1 shows the value of unfinished work W_i for each key time epoch T_i (arrival, departure or the speed switching), compared to the number of customers at service. In this picture we can clearly see the workload-based hysteretic control service rate switching policy. When the workload level hits the k_H threshold, the service rate turns to $r_H = 2.66$, and similarly when the workload level becomes less than the k_L threshold, the rate switches to $r_L = 2.0$, and the workload slope decreases. We also note, that changes in the slope of the workload decrease may also correspond to changes in the number of customers being served, which is clearly visible from the Fig. 1.

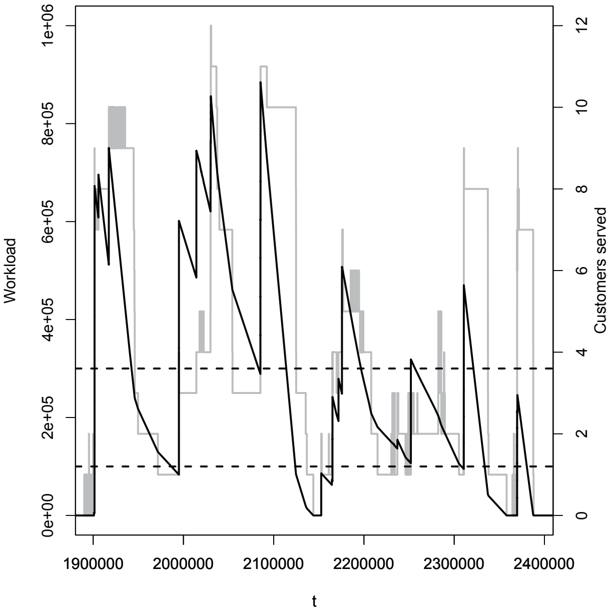

Fig. 1. Workload (black) and number of customers served (gray) at key time epochs, simulation run

Fig. 2 shows the schedule of the HPC compared to customer states (waiting, computing) for 500 customers, starting from customer #7000 of the dataset. The irregularity of the load is clearly seen, which results in heavy waiting times (gray lines at lower picture) corresponding to huge tasks occupying multiple servers (upper picture). We note that the simulation was performed for the FCFS scheduling discipline. It is also seen, that in many cases the servers are occupied by many single-server tasks run simultaneously.

Fig. 3 shows the dependence of average stationary energy consumption, evaluated by (11) and (12), and average stationary workload, on the service rate switching threshold for a

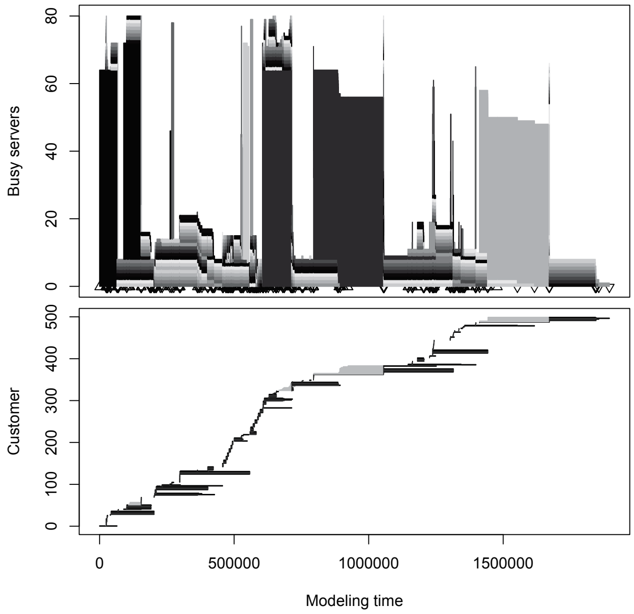

Fig. 2. Upper: number of busy servers occupied by customers (grayscale values correspond to different customers); lower: state of each customer in the system, waiting (gray) and computing (black)

single threshold policy, that is, the hysteretic workload-aware service rate switching, with coinciding thresholds $k_L = k_H = k$. The threshold values were taken as $k = 0, 10^5, \dots, 3 \cdot 10^6$. The tradeoff between two key measures, the average stationary workload, and the average energy consumption, can be clearly seen. We note that the irregularity of the shape of the graph is related to the fact, that it was obtained using a single trace (the driving sequences obtained from the aforementioned dataset). The dependence could be more smooth once obtained for a set of independent randomized simulation runs. However, we leave these experiments for future investigation.

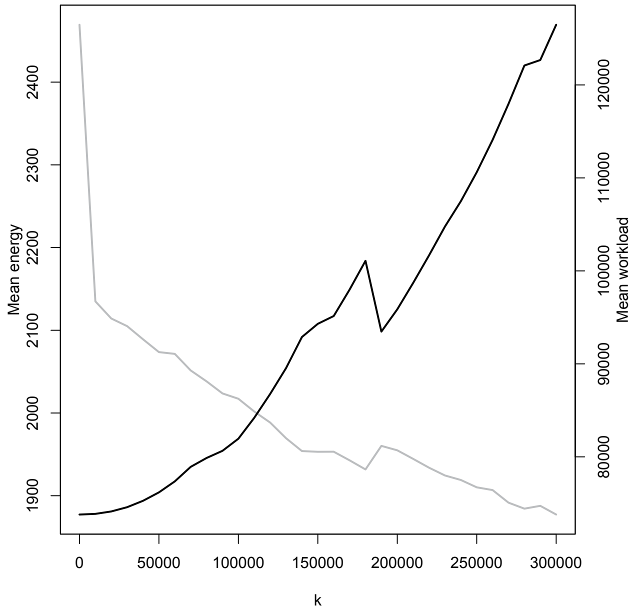

Fig. 3. Energy consumption for a single threshold workload-based switching

V. Conclusion

We presented an approach to implement stochastic simulation of an HPC by means of stochastic recursions. We name a few directions for possible and rather straightforward extensions of the model:

1) consider the characteristics of a customer as a multidimensional vector (corresponding to memory requirements, CPU requirements, restrictions to specific hardware in a real HPC system);
2) imply multiple queues to a single resource pool or dedicated pool partitions;
3) address the server sleep state-related policies by increasing the number of possible events and adopting related transitions.

Note however that aforementioned extensions, although implying additional complications in the state space of the model, do not provide significant complexity and may be easily implemented, as demonstrated in the Seciton III. The presented method is implemented in `hpcwld` package for R language, maintained in CRAN.

Acknowledgment

The study was carried out under state order to the Karelian Research Centre of the Russian Academy of Sciences (Institute of Applied Mathematical Research KRC RAS). This research is partially supported by RF President's grant MK-1641.2017.1 and RFBR, projects 16-07-00622, 18-07-00147, 18-07-00156.

References

[1] Smart 2020: Enabling the low carbon economy in the low carbon economy in the information age. Technical report, The Climate Group, 2008.

[2] United states data center energy usage report. Technical report, Lawrence Berkeley National Laboratory, 2016.

[3] Code of conduct on data centres energy efficiency, version 2.0. Technical report, European Commission. Institute for Energy, Renewable Energies Unit, November 2009.

[4] https://www.top500.org/green500/list/2017/11/?page=4. Top500 supercomputer sites.

[5] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[6] Enhanced Intel. Speedstep® technology for the intel® pentium® m processor white paper, march 2004. Technical report, Recovered 30/1/2011 from World Wide Web: ftp://download. intel. com/design/network/papers/30117401. pdf, 2004.

[7] AMD. Cool'n'quiet technology @ONLINE, January 2014.

[8] Richard V. Evans. Queuing when Jobs Require Several Services which Need Not be Sequenced. *Management Science*, 10(2):298–315, January 1964.

[9] Percy H. Brill and Linda Green. Queues in which customers receive simultaneous service from a random number of servers: a system point approach. *Management Science*, 30(1):51–68, 1984.

[10] S. R. Chakravarthy and H. D. Karatza. Two-server parallel system with pure space sharing and Markovian arrivals. *Computers & Operations Research*, 40(1):510 – 519, 2013.

[11] Alexander Rumyantsev and Evsey Morozov. Stability criterion of a multiserver model with simultaneous service. *Annals of Operations Research*, 252(1):29–39, 2017.

[12] Evsey Morozov and Alexander Rumyantsev. Stability Analysis of a MAP/M/s Cluster Model by Matrix-Analytic Method. In Dieter Fiems, Marco Paolieri, and N. Agapios Platis, editors, *Computer Performance Engineering: 13th European Workshop, EPEW 2016, Chios, Greece, October 5-7, 2016, Proceedings*, pages 63–76. Springer International Publishing, Cham, 2016.

[13] Sung Shick Kim. *M/M/s Queueing System Where Customers Demand Multiple Server Use*. PhD thesis, Southern Methodist University, 1979.

[14] E. V. Morozov and A. S. Rumyantsev. Multi-server models to analyze high performance cluster [in russian]. *Transactions of Karelian Research Centre of the Russian Academy of Sciences*, 5:75–85, 2011.

[15] J. da Kiefer and J. Wolfowitz. On the theory of queues with many servers. *Transactions of the American Mathematical Society*, pages 1–18, 1955.

[16] A. S. Rumyantsev, K. A. Kalinina, and T. E. Morozova. Stochastic modeling of high-performance cluster with hysteretic control of service rate [in russian]. *Transactions of Karelian Research Centre of the Russian Academy of Sciences*, 8:66–75, 2017.

[17] Dror G. Feitelson. *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[18] D. Zotkin and P. J. Keleher. Job-length estimation and performance in backfilling schedulers. In *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No.99TH8469)*, pages 236–243, 1999.

[19] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, may 2010.

[20] Robert Basmadjian, Florian Niedermeier, and Hermann de Meer. Modelling performance and power consumption of utilisation-based DVFS using M/M/1 queues. In *Proceedings of the Seventh International Conference on Future Energy Systems of e-Energy ’16*, pages 84–89. ACM, May 2016.

[21] Jean-Marc Pierson and Henri Casanova. On the utility of dvfs for power-aware job placement in clusters. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, Euro-Par’11, pages 255–266, Berlin, Heidelberg, 2011. Springer-Verlag.

[22] Joshua Dennis Booth, Jagadish Kotra, Hui Zhao, Mahmut T. Kandemir, and Padma Raghavan. Phase detection with hidden markov models for dvfs on many-core processors. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015.

[23] Esa Hyytiä, Douglas Down, Pasi Lassila, and Samuli Aalto. Dynamic Control of Running Servers. In Reinhard German, Kai-Steffen Hielscher, and Udo R. Krieger, editors, *Measurement, Modelling and Evaluation of Computing Systems*, pages 127–141, Cham, 2018. Springer International Publishing.

[24] M. E. Gebrehiwot, S. Aalto, and P. Lassila. Energy Efficient Load Balancing in Web Server Clusters. In *2017 29th International Teletraffic Congress (ITC 29)*, volume 3, pages 13–18, September 2017.

[25] Tibor Horvath and Kevin Skadron. Multi-mode energy management for multi-tier server clusters. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 270–279. ACM, 2008.

[26] Esa Hyyti, Rhonda Righter, and Samuli Aalto. Task assignment in a heterogeneous server farm with switching delays and general energy-aware cost structure. *Performance Evaluation*, 7576(0):17–35, May 2014.

[27] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 157–168. ACM, 2009.

[28] Dror G. Feitelson and Larry Rudolph. Metrics and benchmarking for parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 1–24. Springer, 1998.

[29] Jérôme Lelong, Valentin Reis, and Denis Trystram. Tuning EASY-Backfilling Queues. In *21st Workshop on Job Scheduling Strategies for Parallel Processing*, 31st IEEE International Parallel & Distributed Processing Symposium, Orlando, United States, May 2017.