A Generalized Model for Investigating Scheduling Schemes in Computational Clusters

Tien V. Do [a,c,*] ,Binh T. Vu [b] , Xuan T. Tran [a,c] , Anh P. Nguyen [a]

[a] *Department of Networked Systems and Services, Budapest University of Technology and Economics, H-1117, Magyar tudósok körútja 2., Budapest, Hungary.*

[b]*Inter-University Centre for Telecommunications and Informatics, Budapest University of Technology and Economics, 4028 Debrecen, Kassai út 26., Hungary*

[c] *Vietnam International Research Institute of Sciences, Ho Hoan Kiem 1428, Hanoi, Vietnam*

Abstract

In this paper, we present a generalized model for the performance evaluation of scheduling compute-intensive jobs with unknown service times in computational clusters. We propose the application of parameters defined in the `SPECpower_ssj2008` benchmark of the Standard Performance Evaluation Corporation to construct a performance evaluation model. In addition, we also define a method to rank physical servers based on either the high performance priority or the energy efficiency priority, and measures to characterize the performance of computational clusters.

We investigate three schemes (separate queue, class queue and common queue) for buffering jobs in a computational cluster that is built from Commercial Off-The-Shelf (COTS) servers. Numerical results show that the buffering schemes do not have impact on performance measures related to the energy consumption of the investigated cluster. However, the buffering schemes play an important role in the quality of service parameters such the waiting time and the response time experienced by arriving jobs. Furthermore, Dynamic Voltage and Frequency Scaling should be carefully applied if one wants to reduce the energy consumption of computational clusters.

Keywords: heterogeneous cluster model, buffering scheme, separate queue, class queue, common queue, ranking of servers

T. V. Do et al. A Generalized Model for Investigating Scheduling Schemes in Computational Clusters. *Simulation Modell. Practice and Theory*, DOI:10.1016/j.simpat.2013.05.003, 2013

1 Introduction

The advance of high-speed networking and powerful computers together with the rapid decline of hardware costs led to the widespread application of distributed systems to offer services [1,2]. In such computational grid systems, job scheduling is multi-criteria in nature and is a vital task in state-of-the-art resource allocation studies. The rapid increase in the complexity of computational clusters and the number of users has a significant impact on the energy consumption, which is to be taken into account in the operation of grid systems.

In the literature job allocation algorithms are proposed to schedule arriving jobs in computational clusters. These algorithms are applied dominantly at two levels: grid- and cluster-level. Scheduling policies covering and combining both of these levels are listed in [3–5] and references therein. In addition, some algorithms are implemented with regard to the knowledge about characteristics of jobs. These may belong to either clairvoyant [6,7] or non-clairvoyant algorithms [8,9].

Nowadays, optimizing energy consumption has become as crucial as improving performance [10,11]. Several power management (PM) practices can be applied: the on-off technique completely shuts down the idle components (e.g., disk, CPU), while Dynamic Voltage and Frequency Scaling (DVFS) [12] lowers the operating voltage/frequency of CPU to reduce the energy consumption.

It is worth mentioning that PM is supported in modern COTS servers. Furthermore, the energy consumption of computational clusters can be reduced if the operator of computational clusters systematically exploits the resource heterogeneity of distributed systems with many different types of processors of different power characteristics and performance capacities. This was the main motivation of the investigations performed by Zikos and Karatza [13], where three policies applicable for cluster-level scheduling were compared: SQEE (Shortest Queue based policy with Energy Efficiency priority), SQHP (Shortest Queue based policy with High Performance priority), and PBP-SQ (Performance-Based Probabilistic - Shortest Queue). Their simulation results indicated that SQEE is the best from the aspect of energy consumption, SQHP outperforms the other two schemes at the price of higher energy consumption, and PBP-SQ proved to be the worst among the three schemes. Note that we

* Corresponding author. Tel.: +36 14632070.
Email address: do@hit.bme.hu (Tien V. Do).

have also examined the configuration studied by Zikos and Karatza [13] and can confirm the conclusions of [13]. In addition, Terzopoulos and Karatza [3], Gkoutioudi and Karatza [14] also investigated the scheduling in real-time grid systems.

In this paper, we follow the same approach applied in the study by Zikos and Karatza [13], where compute-intensive jobs with unknown service times are to be scheduled in a cluster with heterogeneous servers. Jobs are executed by servers in a cluster. Compared to previous works [13,3,14], this paper provides a generalized model based on parameters defined by Standard Performance Evaluation Corporation. We also define a method to rank physical servers based on either high performance priority or energy efficiency priority, and measures to characterize the performance of computational clusters. We investigate three schemes (Separate Queue, Class Queue and Common Queue) for buffering jobs in a computational cluster that is built from Commercial Off-The-Shelf (COTS) servers. These proposals allow a systematic way to investigate the performance of computational clusters.

Simulation results show that the buffering schemes do not have impact on performance measures related to the energy consumption of the investigated cluster, but are significant factors regarding the waiting time and the response time experienced by arriving jobs. In addition, we also investigate a saving on the energy consumption when a specific server is switched off if there is no job allocated to the specific server and when a specific server applies DVFS if a job is allocated to the server. Results show that DVFS should be carefully applied if one wants to reduce the energy consumption of computational clusters.

The rest of the paper is organized as follows. In Section 2, a generalized model along with methods to rank physical servers is proposed. Simulation results are presented in Section 3. Finally, Section 4 concludes the paper.

2 A Generalized Model and Scheduling Algorithms

We consider a computational cluster, which serves compute-intensive jobs. Following [13], we assume that jobs

- can be executed on any server,
- are attended to by the First Come First Served (FCFS) service policy,
- are non-preemptible, which means they cannot be suspended until completion,
- have service times unknown to the local scheduler.

Jobs are to be executed by physical servers according to a specific scheduling algorithm. In what follows, we describe scheduling algorithms that allocate arriving jobs to a computational cluster. To make the presentation comprehensible, the classification of servers is provided in Section 2.1. Then, the description of scheduling algorithms is given in Section 2.2.

2.1 Ranking of Servers

In a computational cluster, each physical server belongs to a specific server type. Let S denote the set of server types and $K = |S|$ be the number of server types. Server type s, $s \in S$, is characterized by parameters C_s, $P_{ac,s}$ and $P_{id,s}$, where C_s is the `ssj_ops` value (the number of operations finished during the measurement interval divided by the number of seconds defined for this interval, showing the throughput –workload operations per second– for this period at 100% target load) and $P_{ac,s}$ denotes the average active power measured according to the `SPECpower_ssj2008` benchmark of the Standard Performance Evaluation Corporation (SPEC) at 100% target load (99.7% actual load).

We assume that when a server is busy, it functions at the full load with the power consumption of $P_{ac,s}$. When a server is idle, the server stops CPU main internal clocks via software and $P_{id,s}$ is the power consumption of a server in idle state.

We introduce two functions as follows.

$$r_p(s) = \frac{C_s}{\max\limits_{i \in S} C_i}, \quad s \in S, \tag{1}$$

$$r_e(s) = \frac{\frac{C_s}{P_{ac,s}}}{\max\limits_{i \in S} \frac{C_i}{P_{ac,i}}} \quad s \in S. \tag{2}$$

It is worth emphasizing that $C_s/P_{ac,s}$, $s \in S$, is the performance to power ratio of class s.

Let S_p and S_e denote the ordered sets of server types that are ranked using function (1) and (2), respectively. In ranking S_p (*the high performance priority ranking*) and S_e (*the energy efficiency priority ranking*),

- server types are ranked from 1 to $|S| = K$ based on the computed values $r_p(s)$ and $r_e(s)$, $s \in S$, respectively;
- number one is assigned to server type arg $\max\limits_{s} r_p(s)$ and arg $\max\limits_{s} r_e(s)$, respectively;

- rank K, assigned to server type arg $\min_s r_p(s)$ and arg $\min_s r_e(s)$, respectively.

For ranking S_p, if there are two server types with the same `ssj_ops` value, the server type of higher average active power gets a higher index.

We organize physical servers based on their type. Physical servers of the same type form a class. The classes are ordered according to either ranking S_p (when *the high performance priority* is chosen) or ranking S_e (when *the energy efficiency priority* is preferred). Physical servers are indexed by pair (i, j), $(i = 1 \ldots, K;\ j = 1, \ldots, M(i))$. Note that server $(1, j)$, $j = 1, \ldots, M(1)$, has the highest priority and server (K, j), $j = 1, \ldots, M(K)$, has the lowest priority.

2.2 Scheduling

The task of scheduling algorithms is to allocate arriving jobs to physical servers. Scheduling algorithms can take into account several factors such as the performance, the power consumption, the number of waiting jobs. Furthermore, the organization of waiting space for jobs that are not immediately served upon their arrival is an important question. In [13], the authors assumed that an arriving job will wait in a specific physical server after the scheduling decision, which is quite straightforward from the aspect of implementation. In this paper, we call this queueing solution as a separate queue scheme which is illustrated in Figure 1. Furthermore, we also investigate two further schemes for buffering jobs.

- *Separate Queue Scheme*: an arriving job will wait in a specific physical server after the scheduling decision presented in Algorithm 1. As one observes, the scheduling algorithm chooses the server with the shortest queue. If there are more idle servers or servers of the same queue length, a server is selected based on the priority of its server type.
- *Class Queue Scheme*: there is a common buffer associated to each class (Figure 2). Jobs scheduled to a specific class wait in the buffer of a specific class when all servers in the specific class are busy. When a job departs from any physical server in the specific class, the first waiting job in the buffer of the specific class will be immediately routed to that server. This procedure is also performed when an arriving job that is routed to the buffer finds an empty server in the specific class. Algorithm 2 routes an arriving job based on criteria: idle servers, the priority and the shortest queue length of classes.
- *Common Queue Scheme*: there is one buffer for storing jobs. On a job arrival, if the LS finds all servers busy, the job will be stored in the common queue and will wait for an idle server. A job is immediately served if Algorithm 3

finds an idle server upon its arrival.

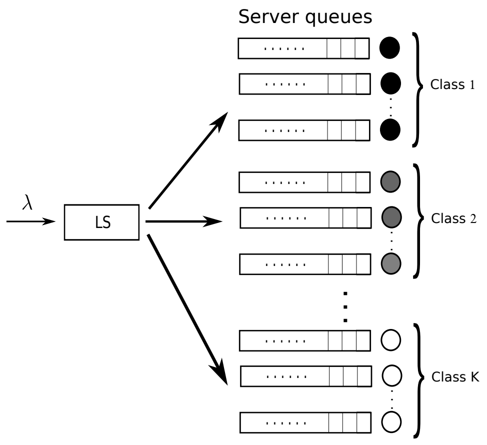


Fig. 1. Separate Queue

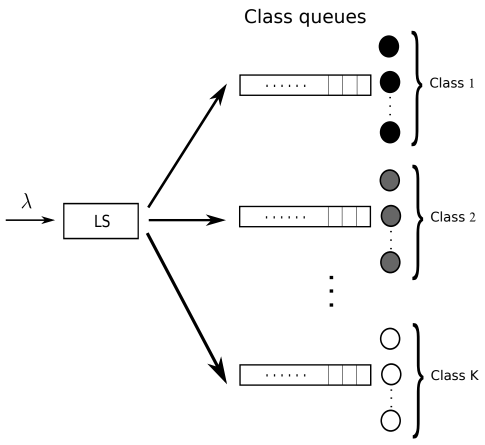


Fig. 2. Class Queue

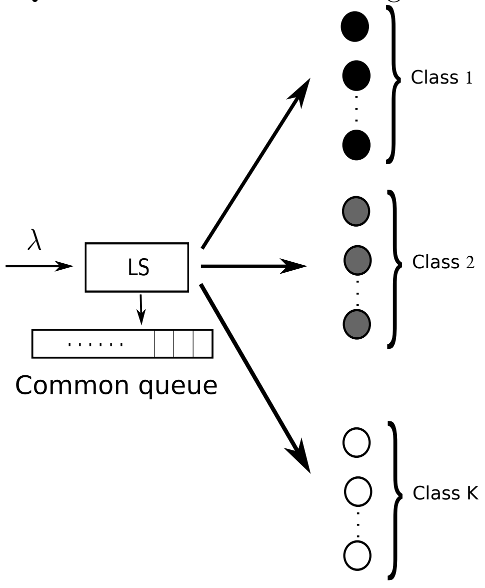


Fig. 3. Common Queue

Algorithm 1 The scheduling algorithm for Separate Queue

```
$best\_server \leftarrow (1, 1)$
for $i = 1 \rightarrow K$ do
    for $j = 1 \rightarrow M(i)$ do
        if server $(i, j)$ is FREE then            ▷ free server found in class $i$
            $best\_server \leftarrow (i, j)$
            GOTO SCHEDULE
        else
            if queue_length of server $(i, j)$ < queue_length of server $best\_server$ then
                $best\_server \leftarrow (i, j)$
            end if
        end if
    end for
end for
SCHEDULE: ROUTE job to queue of server $best\_server$
```

Algorithm 2 The routing algorithm for Class Queue

```
$best\_class \leftarrow 1$
for $i = 1 \rightarrow K$ do
    for $j = 1 \rightarrow M(i)$ do
        if server (i, $j$) is FREE then            ▷ free server found in class $i$
            $best\_class \leftarrow i$
            GOTO SCHEDULE
        else
            if queue_length of class $i$ < queue_length of class $best\_class$ then
                $best\_class \leftarrow i$
            end if
        end if
    end for
end for
SCHEDULE: ROUTE job to queue of class $best\_class$
```

Algorithm 3 The routing algorithm for Common Queue

```
for $i = 1 \to K$ do
    for $j = 1 \to M(i)$ do
        if server $(i, j)$ is FREE then                ▷ free server found in class $i$
            $free\_server \leftarrow (i, j)$
            GOTO SCHEDULE
        end if
    end for
end for
SCHEDULE:
if found $free\_server$ then
    ROUTE job to queue of class $free\_server$
else
    ROUTE job to Common Queue
end if
```

It is worth emphasizing that the practical implementation of the Separate Queue Scheme is the easiest. That is, waiting jobs can be placed inside each physical server. For example, jobs and parameters can be allocated in the local disk of each physical server.

To implement the Separate Queue Scheme, the Class Queue Scheme and the Common Queue Scheme we propose a practical method as follows.

- A file server is operated in a cluster. Files are accessed using Server Message Block/the Common Internet File System (SMB/CIFS) protocol[15] or Network File System (NFS) protocol [16].
- Each queue is allocated a separate spool area/directory in the file server. The spool areas are accessible by the Local Scheduler (LS) and the respective servers.
- The LS maintains the communication with the physical servers in the cluster with the help of a Grid Computing Framework that allows loading and executing tasks. The LS also has the knowledge on the information of the occupancy of each queue and the states of physical servers in the cluster. Based on the knowledge, the communication mechanism and the common spool areas, waiting jobs can be easily allocated to servers depending on the applied buffering approaches. The allocation can be performed quickly in the fast local network of the cluster, which minimally affects the performance of the cluster.

2.3 Performance Measures and Energy Metrics

Let w_l be the waiting time in queue of job l before service and s_l be the service time that takes the server to process job l. The mean waiting time $WT(n)$ and mean service time of n completed jobs are calculated as follows:

$$WT(n) = \frac{1}{n}\sum_{l=1}^{n} w_l$$

and

$$ST(n) = \frac{1}{n}\sum_{l=1}^{n} s_l.$$

Response time r_l of job l is the time period between the arrival instant and the departure instant of job l. We have $r_l = w_l + s_l$.

The average response time $RT(n)$ of n jobs that finished service is

$$RT(n) = \frac{1}{n}\sum_{l=1}^{n} r_l.$$

The average service time, the average waiting time and the average response time are defined as

$$\begin{aligned} ST &= \lim_{n\to\infty} ST(n), \\ WT &= \lim_{n\to\infty} WT(n), \\ RT &= \lim_{n\to\infty} RT(n). \end{aligned}$$

Let the departure time of job n be t_n. Let $\alpha_{i,j}(t)$ denote the total time of active and $\iota_{i,j}(t)$ be the sum of idle time periods of server (i,j) until time t, respectively. If the cluster starts the operation from time instant 0, then $t = \alpha_{i,j}(t) + \iota_{i,j}(t)$.

Since the idle and the active power consumption of server (i,j) are denoted by $P_{id,i}$ and $P_{ac,i}$, the average energy consumption per job when idle servers are not switched off (but functioning in the idle state with a lower power) until t_n is

$$AE_{no-switch}(t_n) = \frac{\sum_{i=1}^{K} P_{ac,i} \sum_{j=1}^{M(i)} \alpha_{i,j}(t_n) + P_{id,i} \sum_{j=1}^{M(i)} \iota_{i,j}(t_n)}{n}, \qquad (3)$$

and the average energy consumption per job until t_n is

$$AE_{switch-off}(t_n) = \frac{\sum_{i=1}^{K} P_{ac,i} \sum_{j=1}^{M(i)} \alpha_{i,j}(t_n)}{n}, \qquad (4)$$

The long term average energy consumptions per job ($AE_{no-switch}$ and $AE_{switch-off}$) are defined as

$$AE_{no-switch} = \lim_{n\to\infty} AE_{no-switch}(t_n), \qquad (5)$$

$$AE_{switch-off} = \lim_{n\to\infty} AE_{switch-off}(t_n). \qquad (6)$$

The parameters, along with the performance and energy metrics used in simulation computations, are summarized in Table 1.

3 Results

Due to the flexibility concerning the modification of codes, we have chosen the SimPack toolkit [17] for all our simulation studies. We have implemented

Table 1
Notations

K	Number of server classes
U	Average system utilization
$WT(n)$	the average waiting time up to job n
WT	the average waiting time
$ST(n)$	the average service time up to job n
ST	the average service time
$RT(n)$	the average response time up to job n
RT	the average response time
$AE(t)$	the average energy consumption per job up to time t
AE	the long term energy consumption per job

the system models and scheduling algorithms in Section 2. In what follows, if it is not stated otherwise, our simulations are performed with the confidence level of 99%. The accuracy (i.e. the ratio of the half-width of the confidence interval and the mean of collected observations) of the energy consumption metrics is of 10^{-5}, while the accuracy of waiting times is less than 0.001.

3.1 Input parameters

3.1.1 Parameter of Server Types

We assume that the considered computational cluster is built from three types of Commercial Off-The-Shelf (COTS) servers. These types are listed in Table 2.

Table 2
Server specifications

Server type	C_s	$P_{ac,*}$ (W)	$C_s/P_{ac,*}$	$P_{id,*}$ (W)
Acer AW2000h-Aw170h F2 (Intel Xeon E5-2670)[18]	6419253	1700	3776	364
Acer AW2000h-AW170h F2 (Intel Xeon E5-2660)[19]	5286503	1275	4146	331
PowerEdge R820 (Intel Xeon E5-4650L)[20]	2790966	457	6102	108

The computing capability (performance) and the power consumption of three server types based on the specification `SPECpower_ssj2008` of SPEC [21] are also presented in Table 2. Note that the computing capability ssj_ops [22] (number of operations finished per second) and the power consumption $P_{ac,*}$ are measured at 100% target load, while $P_{id,*}$ is the power consumption when the servers are in the idle state. The computed rankings based on the rule in Section 2.1 are reported in Table 3.

Table 3
Ranking function

Server type	**Ranking based on Performance**	**Ranking based on Energy Efficiency**
Intel Xeon E5-2670	$r_p(1) = 1.0$	$r_e(1) \approx 0.64$
Intel Xeon E5-2660	$r_p(2) \approx 0.82$	$r_e(2) \approx 0.66$
Intel Xeon E5-4650L	$r_p(3) \approx 0.43$	$r_e(3) = 1.0$

3.1.2 Load

Job arrivals follow an exponential distribution with λ parameter. The execution times of jobs are also exponentially distributed. Each job requires a computing capacity equivalent to 6419253 (ssj_ops) in average, which means the average execution time of jobs is one second if jobs are routed to a server of type Intel Xeon E5-2670. Let μ_1, μ_2 and μ_3 denote the service rates, if jobs are scheduled to a server of type Intel Xeon E5-2670,Intel Xeon E5-2660 and Intel Xeon E5-4650L, respectively. Then we obtain $\mu_1 = 1/s$, $\mu_2 = 0.82/s$ and $\mu_3 = 0.43/s$ as a consequence of the assumption of the average computing capacity required by jobs and Table 2.

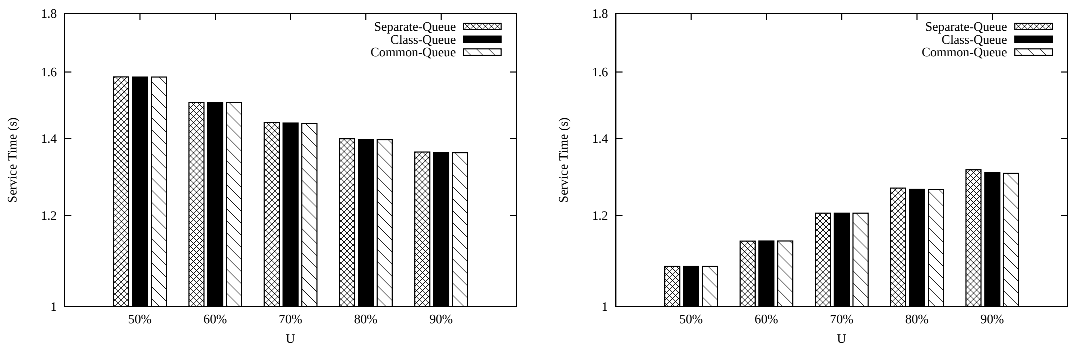
We investigate a computational cluster with three types of COTS servers and eight physical machines in each server class. Therefore, the total `ssj_ops` of the computational cluster is $8 * (6419253 + 5286503 + 2790966) = 14496722$. The investigation is carried out with load U equal to 50%, 60%, 70%, 80% and 90%, where $U = \lambda/(8*(6419253+5286503+2790966))/6419253 = \lambda/(8*(1+0.82+0.43))$. Therefore, the mean inter-arrival time of jobs in our simulation study takes the values of 0.1111 s , 0.0926 s, 0.0794 s, 0.0694 s and 0.0617 s, respectively.

3.2 Numerical results

In this section, we present results concerning the three buffering schemes with two scheduling policies based on Energy Efficiency (EE) priority and High Performance (HP) priority.

3.2.1 Quality of Service

Figure 4 illustrates the mean service time against system load for all system models and policies.



(a) EE policy applied (b) HP policy applied

Fig. 4. Mean service time vs. system load

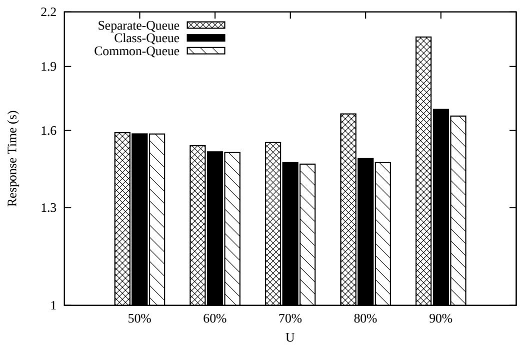
The results show that tendencies observed in case of EE and HP priority are opposing: system models under the EE policy yield a shorter service time as system loads increase, while under the HP policy they give a longer service time. However they all converge to the same service time rounded value of 1.35 (s) at high load.

This can be explained by the dominant utilization of high performing servers in case of high system loads under EE policy, and generally under HP policy. A better performing server requires a shorter amount of time to process the exact job as a lower performing, but less energy demanding server. The anticipation that the mean service times are not impacted by queueing schemes is confirmed by Figure 4.

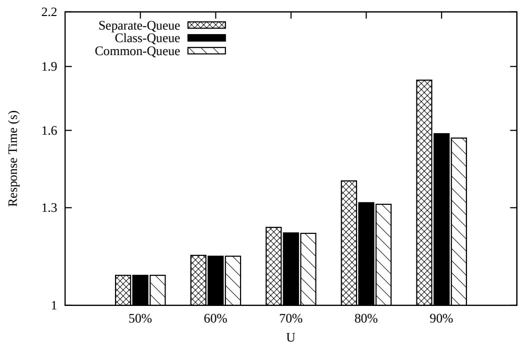
The mean response time is depicted in Figure 5. It can be seen that the proposed models outperform the traditional model under both policies and in every system load. In terms of response time, the Common Queue performs the best, especially under high system utilization. The difference in performance increase directly with system utilization: while no significant difference can be discerned at 50%, a noticeable gap appears at 90%.

It can be inferred from Figure 6 that the expected waiting time of a job waiting in a queue is highest for the Separate Queue buffering scheme, significantly lower in the Class Queue buffering scheme, and lowest in the Common Queue buffering scheme. This is explained by the fact that the Common Queue buffering scheme allows the most efficient utilization of resource because no job is waiting when there is an idle server.

From Figs. 4, 5 and 6, it can be observed that the difference on the mean response time between buffering schemes is due to the discrepancy of the mean

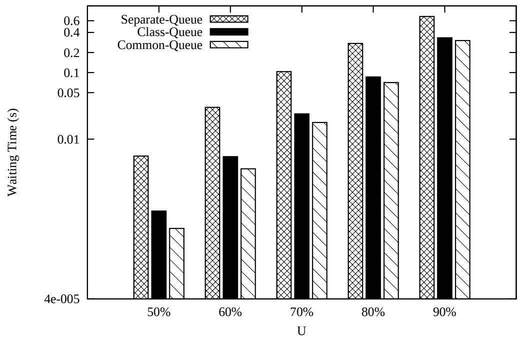


(a) EE policy applied

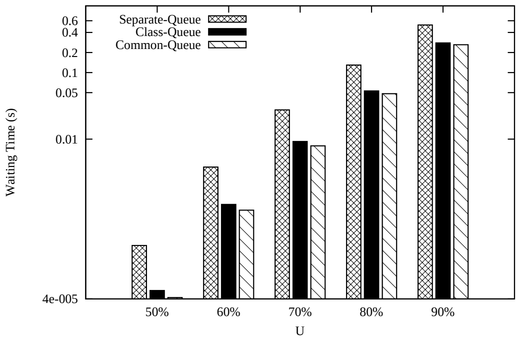


(b) HP policy applied

Fig. 5. Mean response time vs. system load



(a) EE policy applied

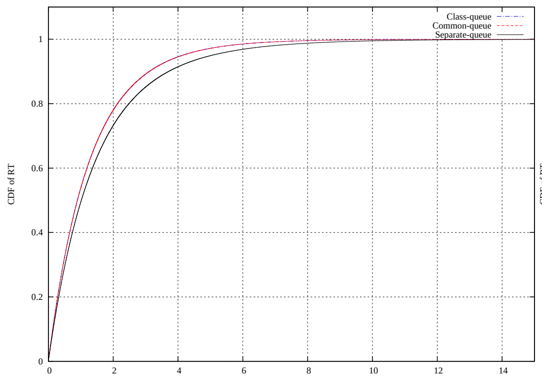


(b) HP policy applied

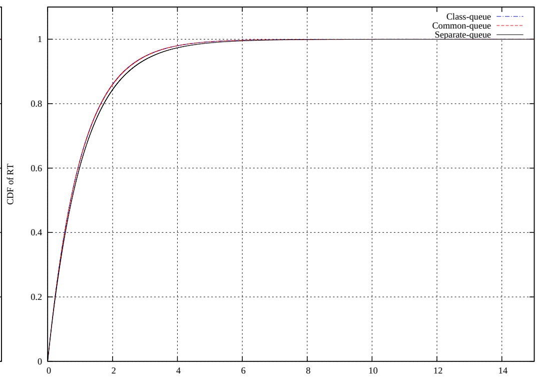
Fig. 6. Mean waiting time vs. system load

waiting time. For the EE policy, at $U = 50\%$, the mean waiting time when Common Queue is applied is 0.00046 s, while the mean waiting time when the Separate Queue is applied is 0.00557 s. The difference is about 0.005 s, which is negligible compared to the mean service time of 1.58441 s. Therefore, no significant difference is observed concerning the mean response time at $U = 50\%$ in Figure 5. However, for the EE policy and $U = 90\%$, the mean waiting time when Common Queue is applied is 0.302311 s, while the mean waiting time when the Separate Queue is applied is 0.693619 s. The difference is 0.3913 s, which is clearly observable in Figs. 5 and 6.

To quantitatively compare the capability of policies and buffering schemes to fulfill deadlines (i.e. the probability that jobs should be executed within a certain deadline) we plot the cumulative distribution function (CDF) of response times in Figure 7 and Figure 8, regarding to 50% and 90% of system load with both policies applied. It is observed that the Class Queue and Common Queue buffering schemes have the same performance concerning the capability to meet deadlines, and they outperforms the Separate Queue schemes in a certain ranges of deadlines. At the medium load, HP policy performs much better because of prior to high performance processors. In the HP policy, the execution of jobs ends up to 1 at limitation value of 6 seconds, while it is

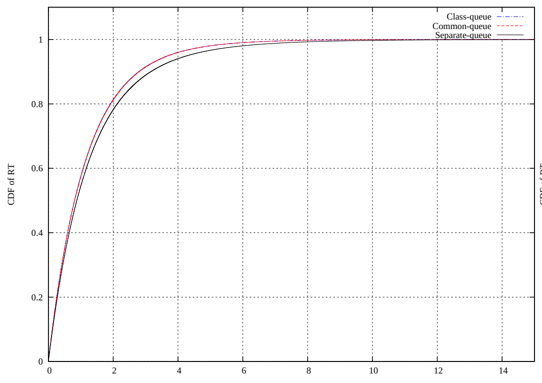


(a) EE policy applied

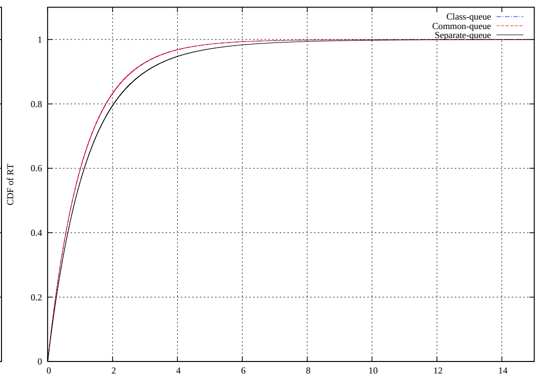


(b) HP policy applied

Fig. 7. CDF of response times at U = 50%



(a) EE policy applied

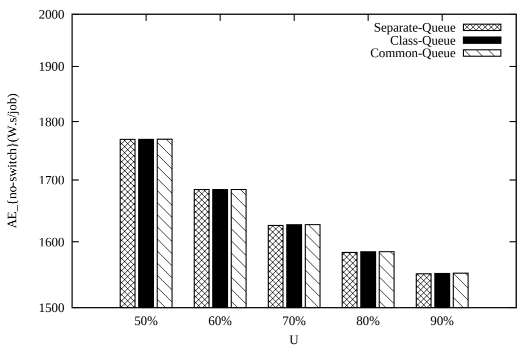


(b) HP policy applied

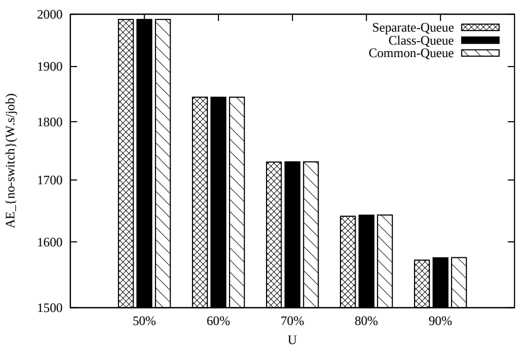
Fig. 8. CDF of response times at U = 90%

the value of 10 seconds in the EE policy. However, at 90% system load, when processors in three sites of every model are utilized quite equally, there is no noticeable difference between the two policies.

3.2.2 Energy Consumption

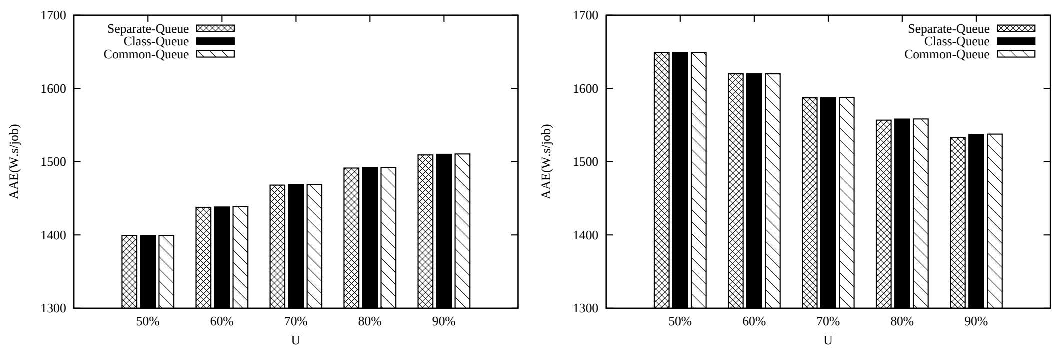


(a) EE policy applied



(b) HP policy applied

Fig. 9. $AE_{no-switch}$ vs. system load



(a) EE policy applied (b) HP policy applied

Fig. 10. $AE_{switch-off}$ vs. system load

In Figures 9 and 10, the energy consumption per job is compared when idle servers are not switched off and are switched off, respectively. It can be observed that the energy consumption is independent of the scheduling schemes.

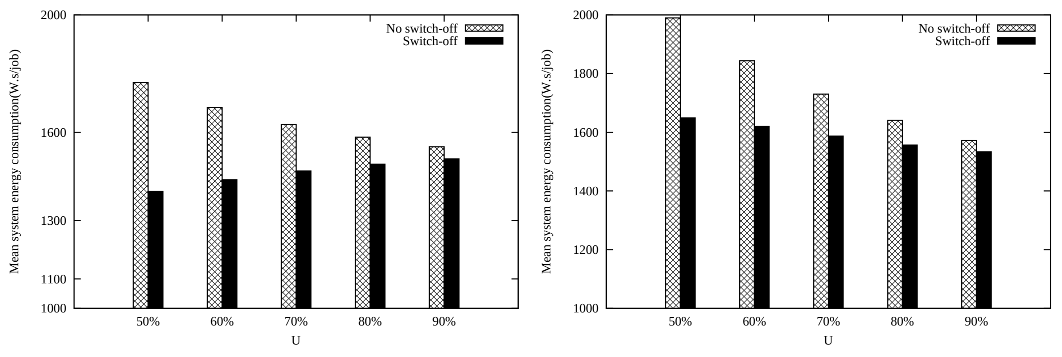
When idle servers are not switched off (Figure 9), the average operating energies of both policies have the greatest difference noticeable at medium system load, where EE consumes 1769.27 $W.s$/job and HP 1989.82 $W.s$/job. The energy consumption per job converges to 1550 $W.s$/job as the load increases.

When idle servers are switched off (Figure 10), the HP and EE policies show opposing tendencies: surprisingly, with HP priority, the energy consumption per job decreases as system load grows, while $AE_{switched-off}$ increases with EE priority. The phenomenon can be explained by the trade-off between performance and energy saving. At medium system load, most job requests are executed on servers operating with lower energy cost, thus the energy dedicated to the execution of jobs is significantly smaller (1399 $W.s$/job vs. 1649 $W.s$/job). As demand for jobs grows, the use of high performance servers becomes inevitable, since the policies are based on shortest queue. In consequence, the difference in AE between SQEE and SQHP decreases at high system load (1510 $W.s$/job vs 1535 $W.s$/job at 90% of utilization).

The impact of saving energy by switching off servers is illustrated in Figure 11. It is observed that the saving decreased as the load is increased.

3.2.3 DVFS

To investigate the impact of DVFS, we create a scenario (applying Common Queue buffering scheme) where processors reduce their frequency and voltage compared to the full power (at %100 target load). In this scenario, the `ssj_ops` value (corresponds to 70% of the full capacity) and the power consumption of the servers applying DVFS when they execute jobs are as follows:

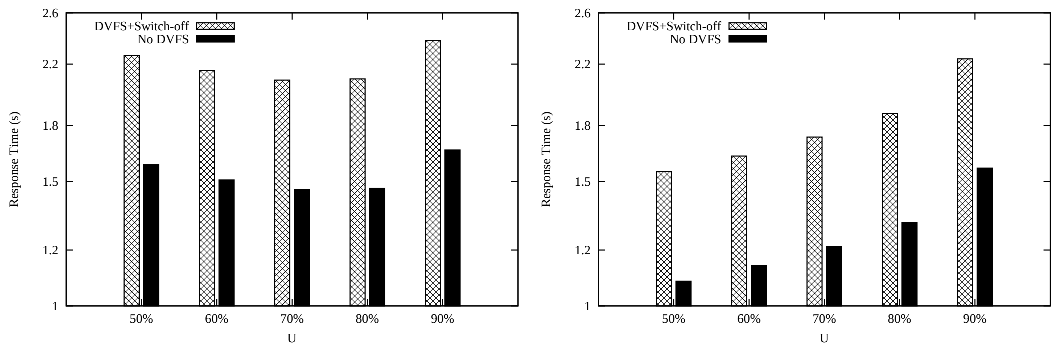


(a) EE policy applied (b) HP policy applied

Fig. 11. Mean energy consumption vs. system load

- Acer AW2000h-Aw170h F2 (Intel Xeon E5-2670): `ssj_ops` is 4517449 and the power is 1169W.
- Acer AW2000h-AW170h F2 (Intel Xeon E5-2660): `ssj_ops` is 3706521 and the power is 881W.
- PowerEdge R820 (Intel Xeon E5-4650L): `ssj_ops` is 1961157 and the power is 317W.

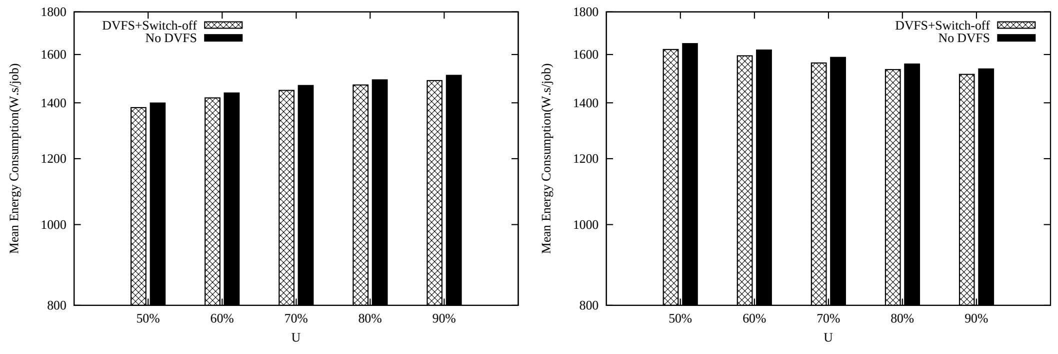
In Figures 12 and 13, we plot the average response time and the average energy consumption per job vs load for the DVFS scenario and the scenario (denoted as "no DVFS") with the full processing capacity. It is worth emphasizing that at the same load value the number of arriving jobs is less in the DVFS scenario than in the scenario at the full processing capacity (note that DVFS and "no DVFS" also switch off idle servers). The impact of DVFS is quite drastic: the increase in the average response time is much higher than the reduction in the average energy consumption per job.



(a) EE policy and Common Queue applied (b) HP policy and Common Queue applied

Fig. 12. Average response time vs. system load

In Figures 14, 15 and 16, we plot results when DVFS and "no DVFS" handle the same number of jobs by keeping the same arrival rate. As anticipated, the price of DVFS is the degradation in the quality of service compared to a case when processors function at their full processing capacity, which is

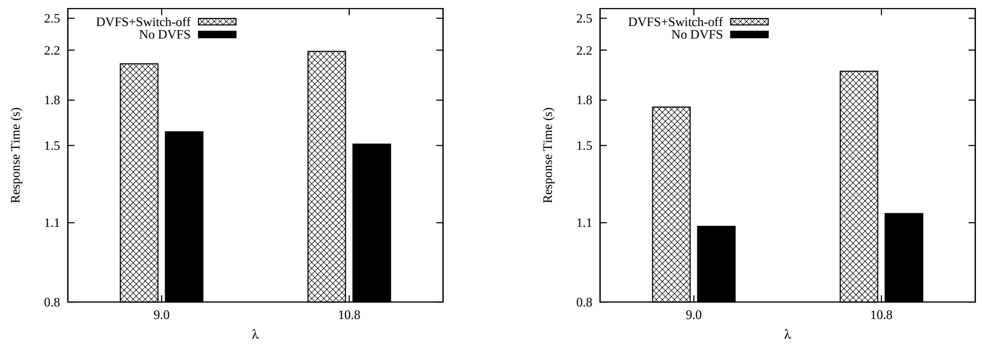


(a) EE policy and Common Queue applied

(b) HP policy and Common Queue applied

Fig. 13. Mean energy consumption vs. system load

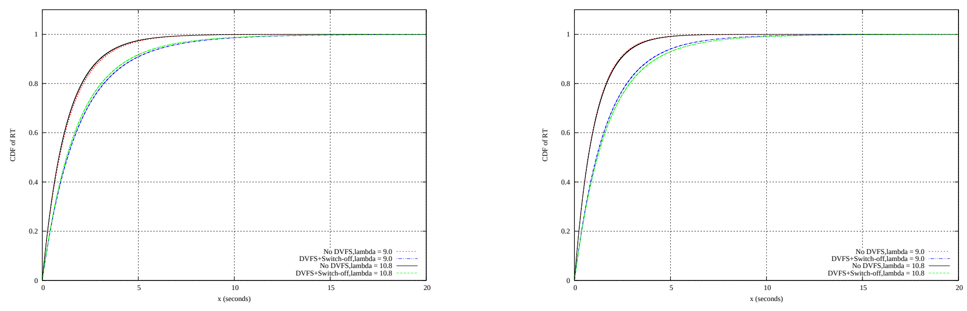
clearly observable in Figure 14. However, the energy consumption of DVFS is higher than the non-DVFS when EE policy and Common Queue are applied. The impact of DVFS on the energy consumption is only observed for the HP policy and Common Queue buffering (see Figure 16). The results illustrate that DVFS should be carefully tuned if one wants to apply DVFS to reduce the energy consumption of computational clusters.



(a) EE policy and Common Queue applied

(b) HP policy and Common Queue applied

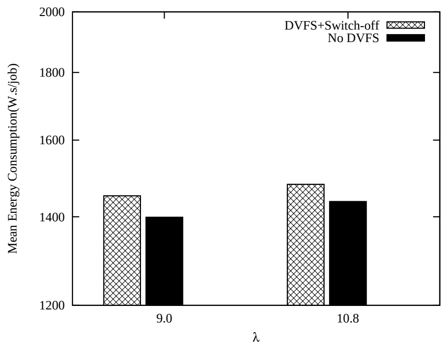
Fig. 14. Average response time vs. λ



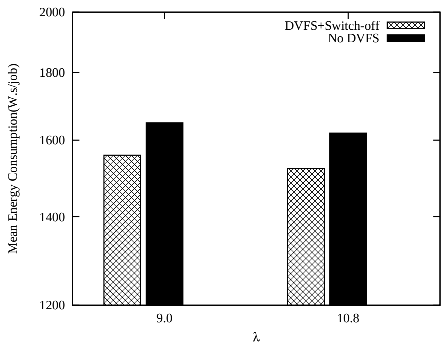
(a) EE policy and Common Queue applied

(b) HP policy and Common Queue applied

Fig. 15. CDF of response times



(a) EE policy and Common Queue applied



(b) HP policy and Common Queue applied

Fig. 16. Mean energy consumption vs. λ

4 Conclusion

In this paper we presented a generalized model for the performance evaluation of scheduling compute-intensive jobs with unknown service times in computational clusters. We gave an implementation of this model on three schemes (Separate Queue, Class Queue and Common Queue), differing in the way incoming jobs are buffered, and on two job-scheduling policies (SQEE, SQHP), prioritizing either energy efficiency or high performance. We defined a ranking methodology of physical servers, which is used to schedule jobs.

Numerical results show that the buffering schemes do not affect the energy consumption of the investigated clusters. However, they have a significant impact on the mean response time and mean waiting time of incoming jobs. Furthermore, the Common Queue and Class Queue schemes markedly outperform the Separate Queue scheme. Therefore, a good buffering scheme can result in improved overall cluster performance without increased power consumption and energy cost. Furthermore, Dynamic Voltage and Frequency Scaling should be carefully applied for the purpose of reducing the energy consumption of computational clusters.

Acknowledgement

The publication was supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund.

References

[1] I. Foster, “The anatomy of the grid: enabling scalable virtual organizations,” *The International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[2] B. Yagoubi and Y. Slimani, “Dynamic load balancing strategy for grid computing,” *Transactions on Engineering, Computing and Technology*, vol. 13, pp. 260–265, 2006.

[3] G. Terzopoulos and H. D. Karatza, “Performance evaluation of a real-time grid system using power-saving capable processors,” *The Journal of Supercomputing*, vol. 61, no. 3, pp. 1135–1153, 2012.

[4] A. Tchernykh, J. M. Ramírez, A. Avetisyan, N. Kuzjurin, D. Grushin, and S. Zhuk, “Two level job-scheduling strategies for a computational grid,” in *Proceedings of the 6th international conference on Parallel Processing and Applied Mathematics*, PPAM’05, (Berlin, Heidelberg), pp. 774–781, Springer-Verlag, 2006.

[5] S. Zikos and H. D. Karatza, “Resource allocation strategies in a 2-level hierarchical grid system,” *Simulation Symposium, Annual*, vol. 0, pp. 157–164, 2008.

[6] S. Zikos and H. D. Karatza, “A clairvoyant site allocation policy based on service demands of jobs in a computational grid,” *Simulation Modelling Practice and Theory*, vol. 19, no. 6, pp. 1465–1478, 2011.

[7] S. Zikos and H. D. Karatza, “The impact of service demand variability on resource allocation strategies in a grid system,” *ACM Trans. Model. Comput. Simul.*, vol. 20, pp. 19:1–19:29, Nov. 2010.

[8] Y. He, W. Hsu, and C. Leiserson, “Provably efficient online non-clairvoyant adaptive scheduling,” in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1 –10, march 2007.

[9] S. Zikos and H. D. Karatza, “Communication cost effective scheduling policies of nonclairvoyant jobs with load balancing in a grid,” *Journal of Systems and Software*, vol. 82, no. 12, pp. 2103–2116, 2009.

[10] T. V. Do and C. Rotter, “Comparison of scheduling schemes for on-demand iaas requests,” *Journal of Systems and Software*, vol. 85, no. 6, pp. 1400–1408, 2012.

[11] L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system-level dynamic power management,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, pp. 299 –316, june 2000.

[12] M. Weiser, B. Welch, A. Demers, and S. Shenker, “Scheduling for reduced CPU energy,” in *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI ’94, (Berkeley, CA, USA), USENIX Association, 1994.

[13] S. Zikos and H. D. Karatza, "Performance and energy aware cluster-level scheduling of compute-intensive jobs with unknown service times," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 239–250, 2011.

[14] K. Gkoutioudi and H. D. Karatza, "Multi-criteria job scheduling in grid using an accelerated genetic algorithm," *J. Grid Comput.*, vol. 10, no. 2, pp. 311–323, 2012.

[15] C. Hertel, *Implementing CIFS - The Common Internet File System.* Prentice Hall, 2003.

[16] B. Callaghan, B. Pawlowski, and P. Staubach, "NFS Version 3 Protocol Specification." RFC 1813 (Informational), June 1995.

[17] P. Fishwick, "Simulation toolkit." `http://www.cs.sunysb.edu/~algorith/implement/simpack/implement.shtml`.

[18] SPEC, "Acer AW2000h-Aw170h f2 (intel xeon e5-2670) machine." `http://www.spec.org/power_ssj2008/results/res2013q1/power_ssj2008-20121212-00590.html`, February 2013.

[19] SPEC, "Acer AW2000h-Aw170h f2(intel xeon e5-2660) machine." `http://www.spec.org/power_ssj2008/results/res2012q4/power_ssj2008-20120918-00546.html`, October 2012.

[20] SPEC, "PowerEdge r820 (intel xeon e5-4650l) machine." `http://www.spec.org/power_ssj2008/results/res2012q4/power_ssj2008-20121113-00586.html`, November 2012.

[21] Standard Performance Evaluation Corporation. `http://www.spec.org/`.

[22] SPEC, "ssj_ops." `http://www.spec.org/power/docs/SPECpower_ssj2008-Result_File_Fields.html#Ops`.