Cluster Comput
DOI 10.1007/s10586-014-0384-x

A comparative study on resource allocation and energy efficient job scheduling strategies in large-scale parallel computing systems

Aftab Ahmed Chandio · Kashif Bilal · Nikos Tziritas · Zhibin Yu · Qingshan Jiang · Samee U. Khan · Cheng-Zhong Xu

Received: 11 November 2013 / Revised: 17 March 2014 / Accepted: 18 May 2014

Abstract In the large-scale parallel computing environment, resource allocation and energy efficient techniques are required to deliver the quality of services (QoS) and to reduce the operational cost of the system. Because the cost of the energy consumption in the environment is a dominant part of the owner's and user's budget. However, when considering energy efficiency, resource allocation strategies become more difficult, and QoS (i.e., queue time and response time) may violate. This paper therefore is a comparative study on job scheduling in large-scale parallel systems to: (a) minimize the queue time, response time, and energy consumption and (b) maximize the overall system utilization. We compare thirteen job scheduling policies to analyze their behavior. A set of job scheduling policies includes (a) priority-based, (b) first fit, (c) backfilling, and (d) window-based policies. All of the policies are extensively simulated and compared. For the simulation, a real data center workload comprised of 22385 jobs is used. Based on results of their performance, we incorporate energy efficiency in three policies i.e., (1) best result producer, (2) average result producer, and (3) worst result producer. We analyze the (a) queue time, (b) response time, (c) slowdown ratio, and (d) energy consumption to evaluate the policies. Moreover, we present a comprehensive workload characterization for optimizing system's performance and for scheduler design. Major workload characteristics including (a) Narrow, (b) Wide, (c) Short, and (d) Long jobs are characterized for detailed analysis of the schedulers' performance. This study highlights the strengths and weakness of various job scheduling polices and helps to choose an appropriate job scheduling policy in a given scenario.

Keywords Parallel computing systems · Job scheduling · Workload characterization · Data center · Energy efficiency

A. A. Chandio · N. Tziritas · Z. Yu · Q. Jiang · S. U. Khan · C.-Z. Xu
Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, People's Republic of China
e-mail: aftabac@siat.ac.cn

A. A. Chandio
Graduate University of Chinese Academy of Sciences, Beijing, People's Republic of China

A. A. Chandio
Institute of Mathematics and Computer Science University of Sindh, Jamshoro, Pakistan

N. Tziritas
e-mail: nikolaos@siat.ac.cn

Z. Yu
e-mail: zb.yu@siat.ac.cn

Q. Jiang
e-mail: qs.jiang@siat.ac.cn

C.-Z. Xu
e-mail: cz.xu@siat.ac.cn

K. Bilal · S. U. Khan (✉)
Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA
e-mail: samee.khan@ndsu.edu

K. Bilal
e-mail: kashif.bilal@ndsu.edu

C.-Z. Xu
Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI, USA

Published online: 15 June 2014

1 Introduction

Scientific organizations are gradually adopting high performance computing for solving large problems, which increases computational and storage needs. In the last decade, various scientific organizations spent gigantic budget to carry out research projects using supercomputers [1]. Because of the fact that supercomputers are unaffordable for various organizations, therefore, the organizations were forced to choose low-budget solutions. Consequently, cloud environment emerged as an alternate to provide the facility of large-scale parallel computations. Currently, many cloud resource providers (RPs) offer thousands of computational nodes and a variety of services to facilitate end-users.

In large-scale parallel computing environments, the end-users submit their requests unaware of the resource allocation strategy. These requests are usually complex jobs, which may be computation-intensive (i.e., job demands more CPU time), data-intensive (i.e., job demands more storage space and communication cost), or mixed (i.e., computation-intensive and data-intensive) [51]. Moreover, these requests may require different levels of quality of service (QoS), including job turnaround and queue time. Furthermore, the large-scale parallel computing environments consist of (a) a mixture of applications and (b) a pool of finite resources.

Because of the above identified factors, RPs pay considerable attention to resource management to deliver the required QoS and enhance system utilization [2]. Several researchers focus on resource management to optimize the system performance considering various QoS constraints. The job scheduling is one of the major components of a resource management system. A scheduling process involves assigning resources to jobs such that no other jobs access the resources at the same time interval [10]. However, due to dynamic nature of the workload, the scheduling problem is hard to solve. The scheduling policy should behave equally well considering the resource heterogeneity and the workload variability. Moreover, the scheduling and resource allocation process is more difficult to design when energy efficiency is also considered [28] [51]. Because of environmental aspects and energy prices, energy efficiency is one of the major requirements of cloud computing. However, energy efficient resource scheduling must not violate the QoS and SLA requirements. In recent years, RPs, such as IBM, Google, and Microsoft have deployed data centers for scientific research, hosting, and storage services for the Internet applications. These data centers are comprised of hundreds of thousands of servers and storage resources. Various components within a data center consume different amount of electricity for their operations, as shown in Table 1 [30]. The table shows the breakdown of peak power consumed by the major components of a single server. It can be observed that the CPU is the largest power consumer within a server.

Table 1 Peak power of the components of a typical server

Component	Peak power (W)	Count	Total (W)
CPU	40	2	80
Memory	9	4	36
Disk	12	1	12
PCI slots	25	2	50
Motherboard	25	1	25
Fan	10	1	10
System Total			213

Considering the aforementioned issues, the contribution of this paper is three-fold: (a) comprehensive characterization of real-world data center workloads, (b) comparison and analysis of a set of job scheduling policies to evaluate system's performance, and (c) the design of job scheduling policies for energy efficiency. The analysis of job scheduling policies can help to select an appropriate job scheduling policy for a given scenario. Additionally, this paper presents an analysis of how workload characteristics affect job scheduling performance. Collecting log files from a real world system is a common approach to estimate the future workloads [11]. Therefore, we use the workloads of a real data center for the experimental evaluation. As the workload characterization is a major factor in evaluation of the system performance [12], we present a comprehensive characterization of the workload. The study of workload characterization motivates to interpret the difference between jobs' computation time, and identifies the similar and repeatable workload trends.

First of all, we examine thirteen job scheduling policies: five priority-based policies, one tuning policy, one window-based policy, and two backfilling techniques. The priority-based scheduling policies are: (a) first come first serve (FCFS), (b) smallest job first (SJF), (c) largest job first (LJF), (d) minimum estimated execution time (MinET), and (e) maximum estimated execution time (MaxET). The aforementioned priority-based scheduling policies are tuned by applying the first fit (FF) technique. Moreover, we use a window-based scheduling policy called Window-K. Furthermore, we consider two backfilling techniques namely: (a) aggressive backfilling and (b) K-reserved based technique. Both backfilling techniques work in conjunction with the FCFS policy.

We simulate all of these scheduling policies with real data center workload. In all of the studied scheduling policies, the job parameters, such as number of jobs, tasks in each job, submission and execution time of each job remain the same to conduct a fair comparison. The aforementioned policies are analyzed using numerous results wherein we split the workload to create multiple datasets. This assumption allows determining, which job scheduling policy produces better results under different datasets. We use four class-based job observations for detailed analysis and comparison of scheduler performance. We performed detailed analysis

and observed interesting findings, such as: (a) MinET and SJF when combined with FF technique exhibit better performance compared to other policies, and (b) a large number of small jobs in the workload can stop the MaxET policy for producing at least same results compared to MinET policy with certain job characteristics. Our analysis reveals the results of all policies into three performance classes, i.e., best class, average class, and worst class.

Finally, after examining the results of the aforementioned policies, based on results of their performance, we incorporate energy efficiency in three policies i.e., (a) best result producer, (b) average result producer, and (c) worst result producer. The selected policies are extended with energy efficiency technique (i.e., DVFS) to further examine the behavior of workload. The proposed energy efficient policies, (a) SJF-energy efficient (SJF-EE), (b) LJF-energy efficient (LJF-EE), and (c) LJF-FF-energy efficient (LJFFF-EE), are extended based on our previous work [29]. Selection of the policies from best, average, and worse policies helps to analyze the energy efficiency strategy. We input the same workload used in the other job scheduling policies.

The rest of the paper is organized as follows. Section 2 states the related work followed by the job scheduling strategy and energy efficient strategy in Sect. 3. The comprehensive characterization and analysis of data center workload is explained in Sect. 4. While in Sect. 5, we present the simulation and experimental results of various job scheduling policies with their discussion. We extend the job scheduling policies to implement the energy efficiency technique and their result discussions are explained in the Sect. 6 . Finally, Sect. 7 concludes the paper and highlights future research directions.

2 Related work

In the large-scale parallel computing environment, RP offers dynamic and geographically distributed access to computational and storage resources. Moreover, RP aims to efficiently utilize the finite resources to a vast number of users, and to maintain the different QoS levels [2]. The resource management problem can be handled by selecting an appropriate job scheduling technique for performance optimization. A vast body of research such as [3–9] has focused on resource management through scheduling techniques to address the problem of resource allocation under different QoS constraints. For instance, the Wei et al. in [3] proposed a metric-aware scheduling policy, where the scheduler balances the competing scheduling objectives represented by different performance metrics, i.e., fairness, job waiting time and system utilization. Khan et al. used a self-adaptive weighted sum technique in [4] and [5], game theoretical methodologies in [6], and goal programming approach in [7] to optimize the system performance for grid resource allocation under different QoS constraints. Braun et al. [8] studied eleven static heuristics for mapping independent tasks on heterogeneous distributed computing system. The authors analyzed and implemented a collection of task mapping policies under a single set of common assumptions. The author in [9] compared the performance of six online scheduling algorithms for batch jobs, keeping in consideration the three objective functions including average flow time, makespan, and maximum wait-time. This paper addresses a similar problem of system performance through a comparative study of job scheduling strategies.

Additionally, these environments respond to a large number of users with *pay-per-use* and *pay-as-you-go* methods, and execute several jobs in parallel. These jobs require long execution times and are considered computation-intensive. The expected workloads for such systems comprise of a mixture of applications that demand different resources, which result in highly variable workloads [13]. Chapin et al. [21] described parallel workload models in detail, and explained standard workload format for large-scale parallel computing environments. The authors used publically available parallel workloads from [22] that consist of various real world workloads obtained from several large-scale parallel computers. These workloads have been characterized and analyzed in [24] and [25]. The authors observed similarities and differences in the workload characteristics, such as different arrival patterns in peak or non-peak intervals and "*power-of-two*" number of processor requirements for job execution. The workload characterization is considered a useful approach for system design. Therefore, we examine and characterize the workloads in different perspectives to find out similarities and differences that can be used as a tool for system's performance optimization.

In the state-of-the-art, various authors used publically available workloads to analyze scheduler performance. Most of the authors make certain assumptions about the nature of jobs to present a specific real system for their experiments. In contrast, we have used the log files collected from a real data center for an evaluation that is closer to real scenarios. Moreover, we study a set of scheduling policies to analyze and compare simulation results, highlighting their performance.

3 Job scheduling strategy

The scheduler is a major component for resources management of large-scale parallel environments. A policy in a scheduler is used to assign jobs to resources at specific time intervals such that the capacity of resources should meet job needs [10]. Suppose m denotes the total number of machines, $M_i (i = 1, \ldots, m)$ to process n jobs$J_j (j = 1, \ldots, n)$. A job J_j is a program submitted by a user at a specific time

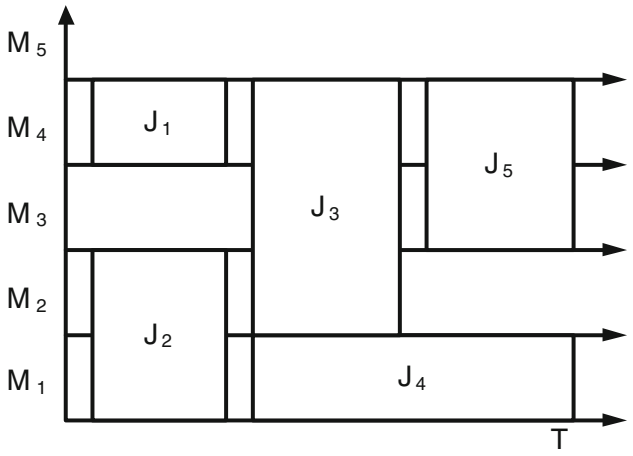

Fig. 1 A Gantt chart for job scheduling process

interval (submit-time). Each job contains one or more tasks $J_j = T_{jk}(k = 1, \ldots, o)$, with each of these tasks being executed on a separate CPU for a given time period. A complete scheduling process schedules the job, and allocates one or more time intervals of one or more machines as shown in Fig. 1. The corresponding scheduling policy problem is to find an optimal schedule process subject to various constraints, such as, (a) minimize the queue time, response time, makespan, and energy consumption, and (b) maximize the overall system utilization.

Job scheduling policies can be static or dynamic. In static scheduling, the set of jobs are known a-priori, while the dynamic one performs scheduling at job arrival. Because of jobs' arrival rate and the status of some nodes (off-line or online) may change without any prior knowledge, therefore, the dynamic scheduling method is required [15].

The scheduling process is categorized into two groups: (a) batch mode and (b) online mode scheduling. In an online mode, the job is scheduled on nodes immediately upon arrival, while the batch mode schedulers collect the jobs in a queue until a specified condition is met. A set of jobs considered for scheduling includes newly arrived jobs and the jobs that were unscheduled in the earlier scheduling events, called meta-tasks [15]. The meta-tasks are examined by the corresponding scheduling policy at prescheduled times called scheduling events. The scheduler events can be defined through regular time interval such as every 10 seconds [15]. The batch scheduling method is successfully applied in large-scale parallel environments, such as banking system, health system, virtual campuses, and bioinformatics applications [52]. However, the batch scheduling method and the independent nature of jobs is hard to solve [15]. The set of schedulers examined in this work are based on either dynamic or static batch scheduling policies. In case of batch scheduling policy, the jobs are grouped in batches and executed irrespective of the dynamic environment.

Similar to the aforementioned scheduler properties, the scheduling process can be further considered as a family of problems with respect to different job models. These job models directly affect the scheduling policies, which are inspired by the way the systems are managed and how the parallel applications are written [16]. In such a model, job flexibility is an advanced partitioning method supported by the application (i.e., rigid, moldable, evolving, and malleable job flexibility) [16]. There is a difference between rigid and moldable/evolving/malleable jobs. In case of a rigid job, the number of CPUs that are assigned to a job does not change throughout the execution. Alternatively, in case of a moldable/evolving/malleable job, the number of CPUs assigned to the job is subject to change throughout the execution. Another model for schedulers is to support different level of preemption, such as preemptive and non-preemptive [16]. In preemption level, the tasks or entire job can be preempted and migrated during the job execution (i.e., Gang scheduling). While in the non-preemptive scheduling, the processors are dedicated to the job throughout their execution after allocation. Preemption method may have great advantage in terms of system performance improvement, but this method may incur extra overheads, such as the cost of memory/communication due to migration and preemption [16]. Therefore, we consider non-preemptive scheduling process in our experiments.

3.1 Energy efficient strategy

Due to the primary focus being high throughput and better performance of the data centers and computing farms in cloud environment, the energy efficiency is a rare consideration at the design time [34]. US Environment Protection Agency reported that the data centers consumed 1.5 % of total sales of electricity in USA for the year 2006, which was 61 billion kilowatt-hour (kWh) [31]. The energy consumption within data centers was estimated to be doubled in 2011 (i.e., more than 100 billion kWh). According to [32], the worldwide power consumption within data centers was doubled from 2000 to 2005. The author estimated that around 80 % of the growth in power consumption can be attributable to servers, while 10 % to communication and remaining 10 % to storage resources.

In recent report [33] by Koomey in the year 2011, the rapid rates of growth in data center electricity, which was prevailed for the year 2000 to 2005, is estimated slowed significantly from the year 2005 to 2010 instead of doubled [33]. Authors [32–35] identified the factors such as: (a) to apply the dynamic voltage and frequency scaling (DVFS), (b) to apply dynamic power management (DPM) or device

reduction, (c) to improve the server, storage and cooling efficiency, (d) multi-core processor designs and (e) virtualization. All of the above are the major contributed techniques to reduce the energy consumption without much overhead [32], [34], [35]. The next sections briefly explain the DVFS and the DPM techniques for power efficiency.

3.1.1 DPM

DPM is the most effective and an aggressive technique for power saving in which the devices can be powered on/off dynamically [51]. Benini et al. [36] introduced the DPM methodology for dynamically reconfigure the electronic devices. The DPM technique can be applied at various power-hungry components, such as CPU, storage disk, memory, servers, and network devices. All these devices consume substantial power, even with low or zero loads. It has been reported in [37] that an idle machine consumes about 2/3 of the peak load on the server. In addition, the average load of the data center resources is only around 30 % of the resources [38]. These facts allow placing around 70 % of the resources to sleep mode in order to eliminate the idle power consumption most of the time [39].

DPM can be applied to place the processor cores in sleep mode [40], turn off the banks of memory [41], and transitioning network devices to sleep [39,42,50]. Due to the fact that the peak performance of the system components only rises during few time intervals, the components are not always be required in active state. For instance, virtual machine workload consolidation techniques are used to consolidate the workload on least number of nodes so that rest of the nodes can be powered off. Typical energy efficient scheduling policies consider: (a) to consolidate the workload on least set of the computing resources and (b) to increase the number of computing resources that can be transitioned to sleeping mode [43]. DPM technique may result in some overheads, such as slept devices take considerable time to wake up, which consumes more energy and increase delay.

3.1.2 DVFS

DVFS for energy efficiency has become a popular research issue in the last decade [44–47]. The basic idea behind the DVFS technique is to adjust the clock frequency of the CPU through appropriate supply voltage to reduce the CPU energy consumption. DVFS technique is an example of energy proportional techniques [50]. Energy proportional means to consume energy according to workload, i.e., less workload should result in lower energy consumption. On the other extreme, a shrinking fraction of total server power is only up to 25 % in current systems [48].

DVFS can be applied at two levels [45]: (a) behavioral and (b) system level. In behavioral level, unique supply voltage for the machine is determined during design which can be constant at runtime. While at the system level, the supply voltage of the machine can vary at runtime. The dynamic voltage machines offer better potential and more flexibility for reducing the energy consumption. This paper solves the problem of energy efficient job scheduling strategy in DVFS method at the behavioral level.

4 Workload analysis and characterization

The system performance is evaluated considering the characteristics of hardware and software components, as well as the workload it processes [12]. Workload characterization helps in understanding overall behavior of the system highlighting the job arrival rate, job size, and job length. Major challenges include: (a) how to manage the system for different loads, (b) how to utilize the resources efficiently, (c) how to meet user demands, and (d) how to minimize the total cost of ownership. Aforementioned questions mandate the RPs to select appropriate resource management techniques, such as job scheduling policies (see Sect. 3 for details).

4.1 Dataset information

We characterized a real data center workload from the Center for Computational Research of State University of New York at Buffalo to evaluate the system performance. The data center is a collection of multiple computational resources clustered together using communication infrastructure, which fall into two categories: (a) homogeneous and (b) heterogeneous resources. The resources in homogeneous systems are similar in terms of size and capacity, in which a job executes in similar capacity, whereas the resources in heterogeneous system are organized with different specification.

The workloads were collected during 30 days' time period from February 20, 2009 to March 22, 2009. A total of 22,385 jobs were executed on more than 1,000 dual processor nodes [27,28]. A complete specification of the data center is presented in Table 2

The total offered load which is the amount of queued workload over time is shown in Fig. 2. The offered load exceeds

Table 2 Full specification of data center

Time duration	20 February 2009 to 22 March 2009
Total jobs ran out on DC	22,385
Total distinct nodes	1,045
Processor name	1056 Dell PowerEdge SC1425 nodes
Processor speed	3.0 GHz or 3.2 GHz
Peak performance	13 TFlop/s

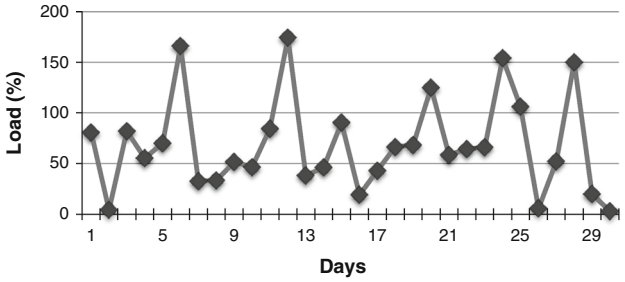

Fig. 2 Total offered load in a month per day

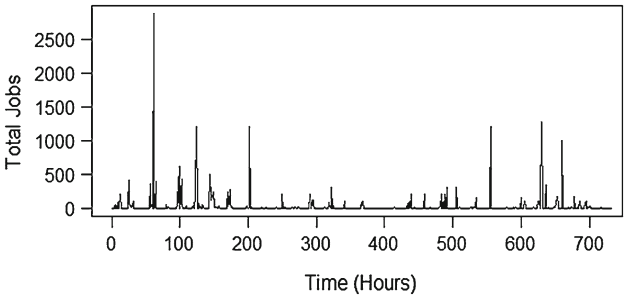

Fig. 3 Jobs arriving per hour

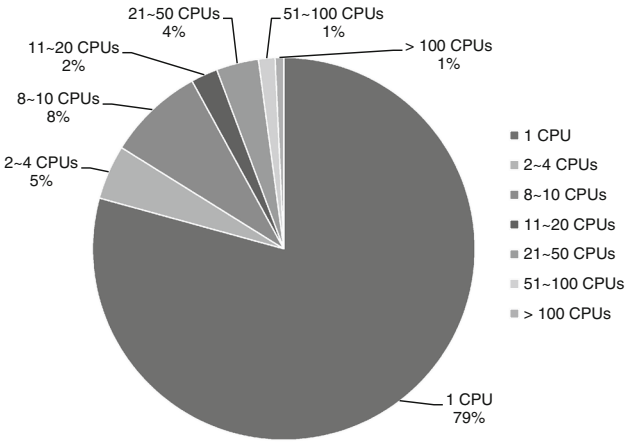

Fig. 4 Jobs breakdown according to number of CPUs

Table 3 Breakdown distribution for job length

Job length	No. of jobs	% of jobs
<1 h (Short)	10,428	46.58
>1 h (Long)	11,957	53.42
Total	22,385	100

100 % in many days, which implies that enough resources are unavailable to complete the tasks given to the system at particular times.

4.2 Job characteristics and classification

A job is generated by a user and submitted to the system. The system in turn, according to its scheduling policy, allocates a number of processors meeting the demands of the job in question. In this section, we characterize jobs according to different perspectives, such as job arrival rate and job size. Figure 3 shows the total number of jobs arrived per hour in 30-day cycle. From this figure we can make two observations: (a) there are fluctuations in the job arrival rate per hour, (b) the system experiences high job arrival rates in specific time intervals, and (c) job arrival rate does not follow a uniform distribution at hourly cycle.

In terms of the job size we make the following observations. The job size can be well explained in a 2D chart, with y axis representing the number of processors while x axis representing the execution time (see Sect. 3) [14]. Therefore, we distribute the job size according to: (i) job's width representing the number of CPUs required by the job in question and (ii) job's length representing the execution time of the respective job. The above perspectives are further classified into four categories: (a) Narrow, (b) Wide, (c) Short, and (d) Long. Specifically, regarding (i) a job requests either a single CPU (Narrow category) or an even number of CPUs (Wide category). On the other extreme (job's length), a job is executed within either an hour (Short category) or more than an hour (Long category). The above categories are classified based on the aforementioned workloads. Figure 4 shows the job distribution according to their width (CPU requirement). Our analysis revealed that the jobs demand either single CPU (i.e., Narrow jobs) or even number of CPUs (i.e., Wide jobs). It is worth mentioning that the number of Narrow jobs is dominant, i.e., 79 % of the total jobs, whereas, 21 % of the total jobs are in Wide category.

Our next observation is that the job length exhibits the time length of the job being executed. We observed in Table 3 that almost 50 % of the total jobs belong to Short category. Consequently, the rest of them belong to Long category.

To understand in-depth analysis of workload, Fig. 5 presents the classification of the jobs in (a) Short and (b) Long categories for in-depth analysis. It can be observed that most of the Short jobs (around 86 %) are executed within 18 min, while around 66.6 % of the Long jobs are executed within 12 h.

We also analyzed the job-size to address the correlation between job width and job length, as shown in Table 4. The correlation table reveals that the Narrow jobs dominate all of the categories. Moreover, Short jobs with execution time between 11 and 20 min and Long jobs with execution time more than 11 h are also prevailing.

In the section workload characterization, our analysis revealed important job characteristics, such as that jobs arrival rate does not follow any trend and possesses heterogeneity in job size and resource requirements. Such

heterogeneity in workloads dictates to analyze the effect of various scheduling policies in such scenarios.

5 Resource allocation job scheduling policies

Considering the scheduler properties discussed in Sect. 3, we simulate thirteen resource allocation job scheduling policies. All of these policies are briefly described as follows. The FCFS is a simple and static job scheduling policy, where a job is served on arrival basis. In this policy, a job can create long delay for the next jobs when the ready processor does not meet the requirements of the job in question [17]. The LJF and SJF scheduling update the batch of jobs (i.e., meta-task) in decreasing and increasing order in terms of job's size (i.e., job width), respectively. Alternatively, the MinET and MaxET update the batch of jobs in decreasing and increasing order in terms of job's length, respectively [15]. The FF is an additional technique to enhance the capability of the above five policies. The FF policy finds a job in shared ordered queue list that can be fit to the first available idle resources to increase resource utilization.

A backfilling technique [17], [18] makes resource reservations for jobs in the queue, and backfills these jobs under the constraint that next jobs (i.e., Short jobs) may not violate the time reserved for previous jobs. There are two basic backfilling techniques: (a) aggressive and (b) conservative. The aggressive technique makes a reservation only for the first job in the queue, while the conservative technique makes reservations for all of the jobs contained in the shared queue. The aggressive backfilling (named EAZY) was developed for IBM SP1 parallel supercomputer, which is based on FCFS scheduling policy [17], [18]. In our work, we use the aggressive technique because it outperforms the conservative backfilling technique [18]. In the K-reserved based policy, queue list has a counter containing K number of times that it has been overtaken by subsequent jobs [19]. The K-reserved based policy works similar to the aggressive backfilling technique with the difference being that in the K-reserved based policy, a job is considered to backfill only K numbers of times, while the aggressive technique does not have any limitation in terms of the times that job is considered to backfill. The Window-K policy enhances the FCFS policy for a window of K consecutive jobs [19]. The window starts with the oldest waiting job, and it contains up to K number jobs arrived successively. We set the value of $K = 5$ for the K-reserved based policy and $K = 10$ for the Window-K policy as suggested by the authors in [19].

5.1 Experimental setup

This section presents the simulation details for the set of scheduling policies. All of the policies are used to schedule the aforementioned workload to figure out the best job

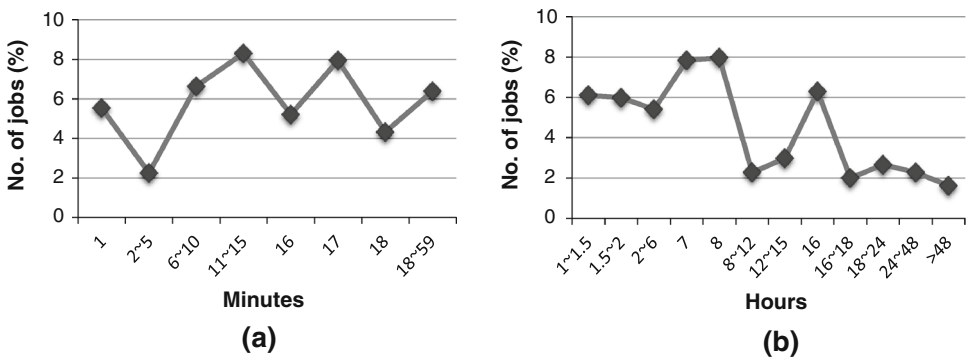

Fig. 5 Breakdown distributions of the **a** Short jobs and **b** Long jobs

Table 4 Percentage breakdown for correlation between jobs width and length

Job size		Job width			
		1 CPU	2–24 CPUs	>32 CPUs	Total jobs
Job length	<1 min.	4.83	0.58	0.13	5.55
	2–10 min.	7.60	0.90	0.38	8.88
	11–20 min.	20.72	3.40	2.97	27.09
	21–60 min.	3.73	1.07	0.27	5.07
	2–4 h	13.23	0.67	0.68	14.57
	5–8 h	18.43	0.15	0.16	18.74
	>9 h	10.73	8.27	1.10	20.11
Total Jobs		79.27	15.03	5.70	100

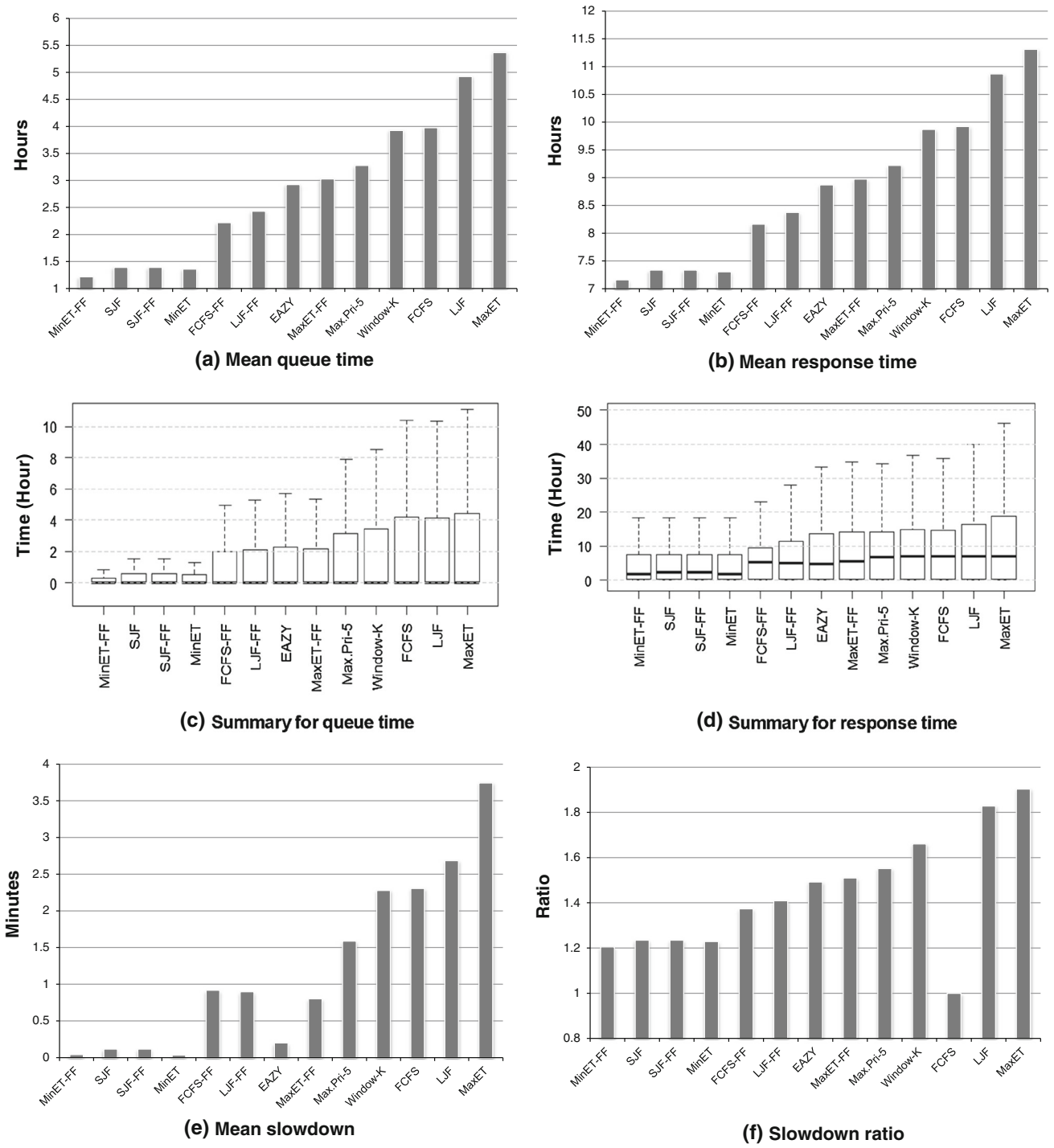

Fig. 6 Comparison results of all policies with overall workloads: **a** Mean queue-time, **b** mean response-time, **c** box-plot for queue-time of all jobs, **d** box-plot for response-time of all jobs, **e** mean slowdown, and **f** slowdown ratio

scheduling policies for optimizing the energy efficiency along with the system performance. For the workloads simulation, we developed a custom Java based discrete event simulator. The Java environment setup allows database connectivity [20], where we stored a dataset described in the Subsect. 4.1. In the event-based setup, the scheduler policies check the queue periodically (i.e., every 10 s [15]) and schedule the jobs accordingly.

Table 5 Observation table for entire workloads (job's breakdowns for correlation between job's width and job's length)

Job size		Short (S) ≤1H	Long (L) >1H	Total
Narrow (N)	1 CPU	36.88	42.39	79.27
Wide (W)	>1 CPU	9.70	11.03	20.73
Total		46.58	53.42	100.00

5.2 Simulation results

In this section, we analyze all of the scheduling policies considered in this work. We consider four metrics to evaluate the performance of considered scheduling policies (a) mean queue-time, (b) mean response-time, (c) mean slowdown, and (d) slowdown ratio. It is worthy to note that the service providers are mainly concerned about the mean response-time and mean queue-time, while the customers are concerned with the mean slowdown and the slowdown ratio. The job's queue-time represents the time elapsed from the arrival time of the respective job until the assignment of the corresponding job to the assigned nodes. The job's response-time represents the time elapsed after the arrival of the respective job until its finish [23]. Equations 1 and 2 are used to calculate the mean response-time and queue-time of the entire workload, respectively.

$$\text{Mean Queue Time} = \frac{\sum \text{Time}\,(\text{start}_{\text{time}} - \text{submit}_{\text{time}})}{\text{Total number of jobs}}, \quad (1)$$

$$\text{Mean Response Time} = \frac{\sum \text{Time}(\text{end}_{\text{time}} - \text{submit}_{\text{time}})}{\text{Total number of jobs}}. \quad (2)$$

In the sequel, we give the definition of the rest of the metrics. Mean slowdown is the normalized time of each job (i.e., job completion time divided by job running time). The slowdown ratio exhibits the normalized time of mean response-time [26] derived in Eq. 3.

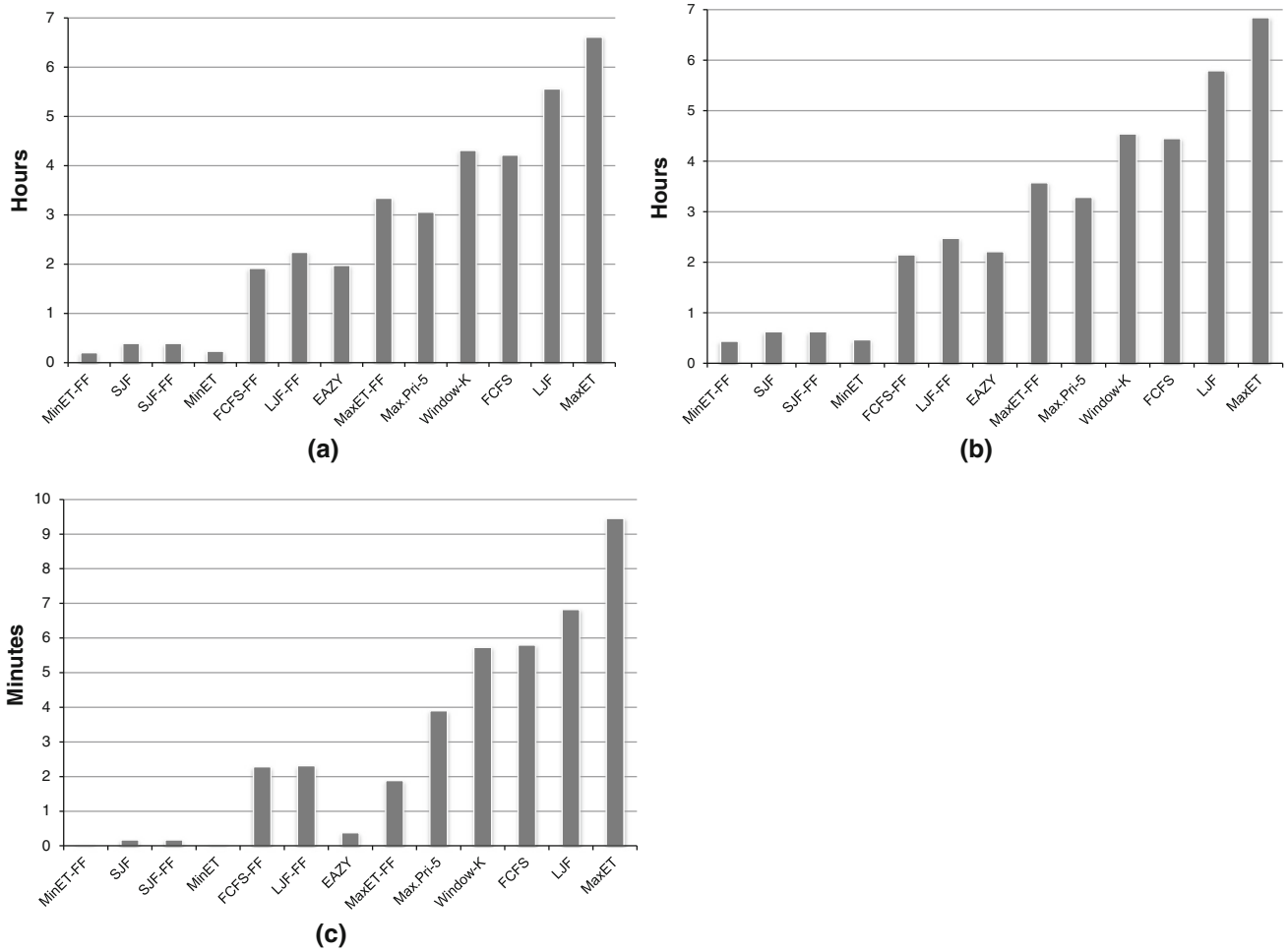

Fig. 7 Comparison results of all policies with overall workloads in first observation: **a** mean queue-time, **b** mean response-time, and **c** mean slowdown

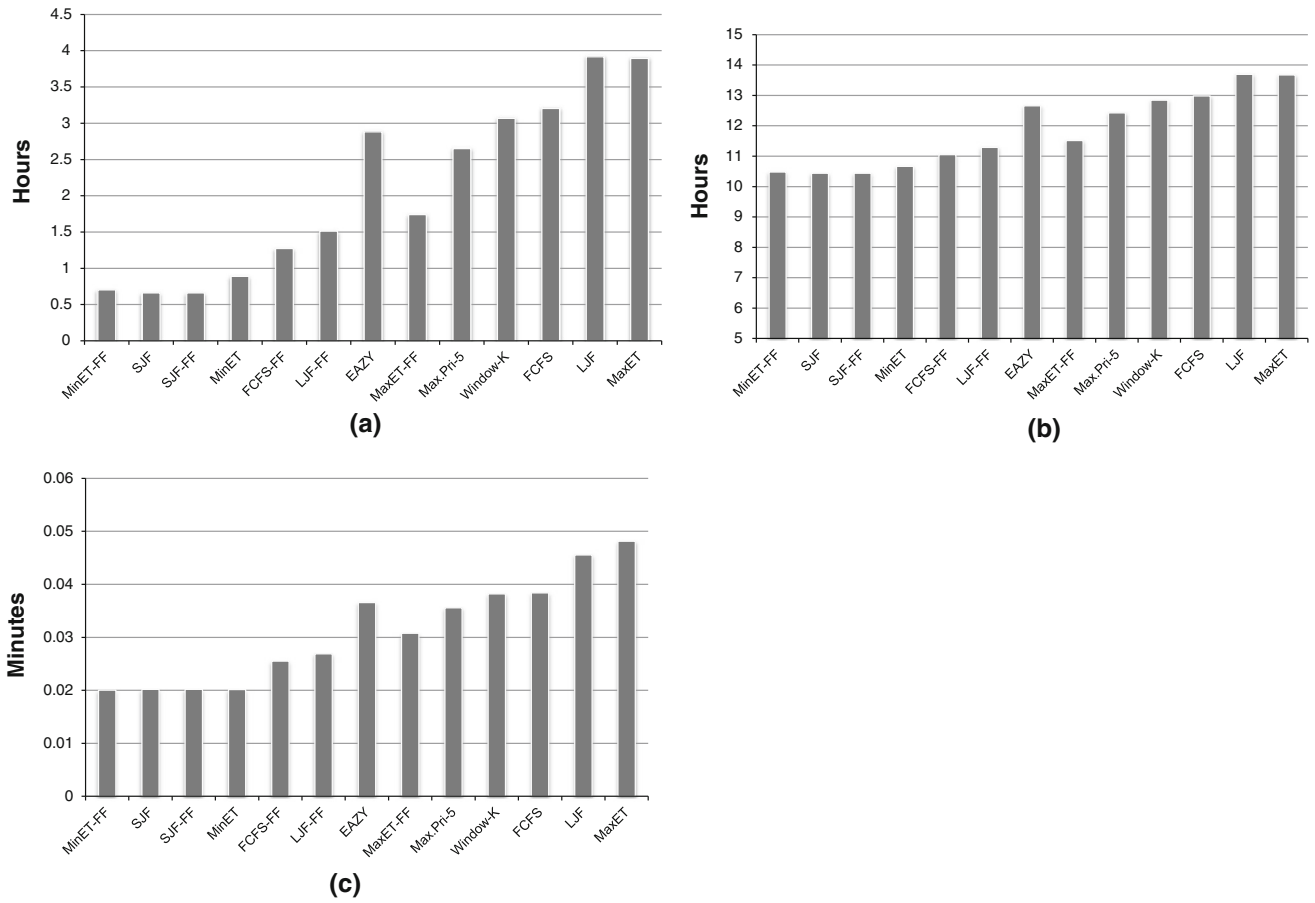

Fig. 8 Comparison results of all policies with overall workloads in second observation: **a** mean queue-time, **b** mean response-time, and **c** mean slowdown

$$\text{Slowdown ratio} = \frac{\text{Time (mean\_response)}}{\text{Time (mean\_execution)}}, \tag{3}$$

where Time (mean_execution) and Time (mean_response) (see Eq. 2) indicate the mean running time and response-time of the entire workload. For example, if the mean response-time of a set of jobs is 10 time unit and the mean execution-time of the jobs on nodes is 5 time unit, then the slowdown ratio will be 2 time unit. The slowdown ratio is important for measuring the performance of the scheduling policy for the entire workload [23]. Figure 6 presents the results of the aforementioned performance metrics for the entire workload.

We further compare the performance metrics in terms of job size correlations. To see how job characteristics affect the scheduling accuracy, we create four job observations based on the aforementioned job categories. Table 5 presents the classification as Short and Narrow (SN), Long and Narrow (LN), Short and Wide (SW), and Long and Wide (LW) jobs in Figs. 7, 8, 9, and 10, respectively.

$$\text{Job}_{\text{width}} = 1\,\text{CPU AND Job}_{\text{length}} \leq 1\,\text{Hour} \tag{Ob.1}$$

$$\text{Job}_{\text{width}} = 1\,\text{CPU AND Job}_{\text{length}} > 1\,\text{Hour} \tag{Ob.2}$$

$$\text{Job}_{\text{width}} > 1\,\text{CPU AND Job}_{\text{length}} \leq 1\,\text{Hour} \tag{Ob.3}$$

$$\text{Job}_{\text{width}} > 1\,\text{CPU AND Job}_{\text{length}} > 1\,\text{Hour} \tag{Ob.4}$$

5.3 Discussion

This section discusses the results and conclusions of the above sections. We explored the job characteristics such as (a) the maximum percent of total jobs requesting single CPU for execution and (b) the percentage of jobs requesting even number of CPUs. Another remark in terms of job running time is that almost half of the total jobs execute within an hour (Short jobs), while the rest of them require more than an hour (Long jobs). Moreover, jobs' arrival rate does not follow a uniform distribution. The above findings exhibit the workload heterogeneity that may affect various services offered by the system under consideration. Therefore, it is essential to compare and analyze different scheduling policies.

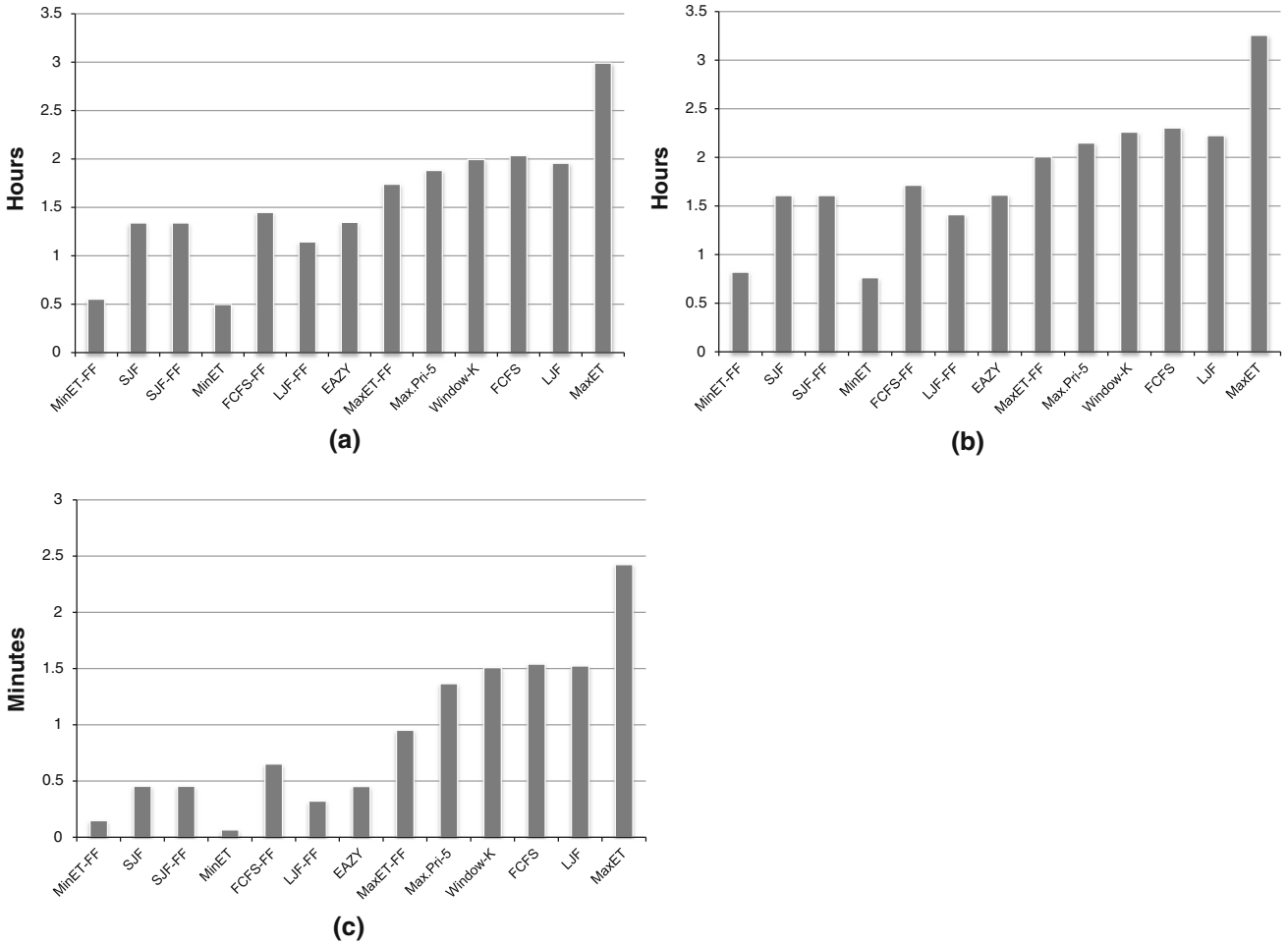

Fig. 9 Comparison results of all policies with overall workloads in third observation: **a** mean queue-time, **b** mean response-time, and **c** mean slowdown

Various job scheduling policies are studied in this paper for large-scale parallel computing systems. Some job scheduling policies produce results with overheads. However, each policy possess various characteristics, such as FCFS produces better results with respect to fairness, but does not support resource fragmentation. If a job demands a large number of CPUs for execution and at that time period the system cannot serve the job due to unavailable CPUs, then the job waits in the queue and prevents the next job from being executed. The above happens even in the case that the requirements of the next job are met by the system. Consequently, the aforementioned case increases the job queue time as well as the response time.

Our analysis in all of the observations in previous section reveals that the results of FCFS policy are not satisfactory in terms of job queue time and response time metrics. A solution of the processors fragmentation problem in FCFS is introduced in backfilling technique (i.e., conservative and aggressive [17], [18]) with addition to maintain the fairness situation. However, taking into account the introduced job characteristics, other job scheduling policies may become superior to FCFS. For instance, with respect to the job size (i.e., job's width), the LJF results in better solutions for Wide jobs, while SJF exhibits better results in terms of Narrow jobs. Similarly, according to job running time (i.e., job's length), MaxET and MinET policies are well suited for Long jobs and Short jobs, respectively.

The scheduling policies are evaluated under three different classes, as shown in Table 6. These classes distinguish the scheduling policies into three different sets of policies, such as Class-I, Class-II, and Class-III. The sets of the scheduling policies in each class are explained as follows. Class-I includes four policies: MinET, SJF, as well as these policies combined with FF technique. In Class-II, FF technique is combined with FCFS, LJF, and MaxET. In the same class we also include the aggressive backfilling (EAZY) and K-reserved based (Max_Pri) policies. Finally, Class-III consists of four policies: FCFS, LJF, MaxET, and Window-K policies.

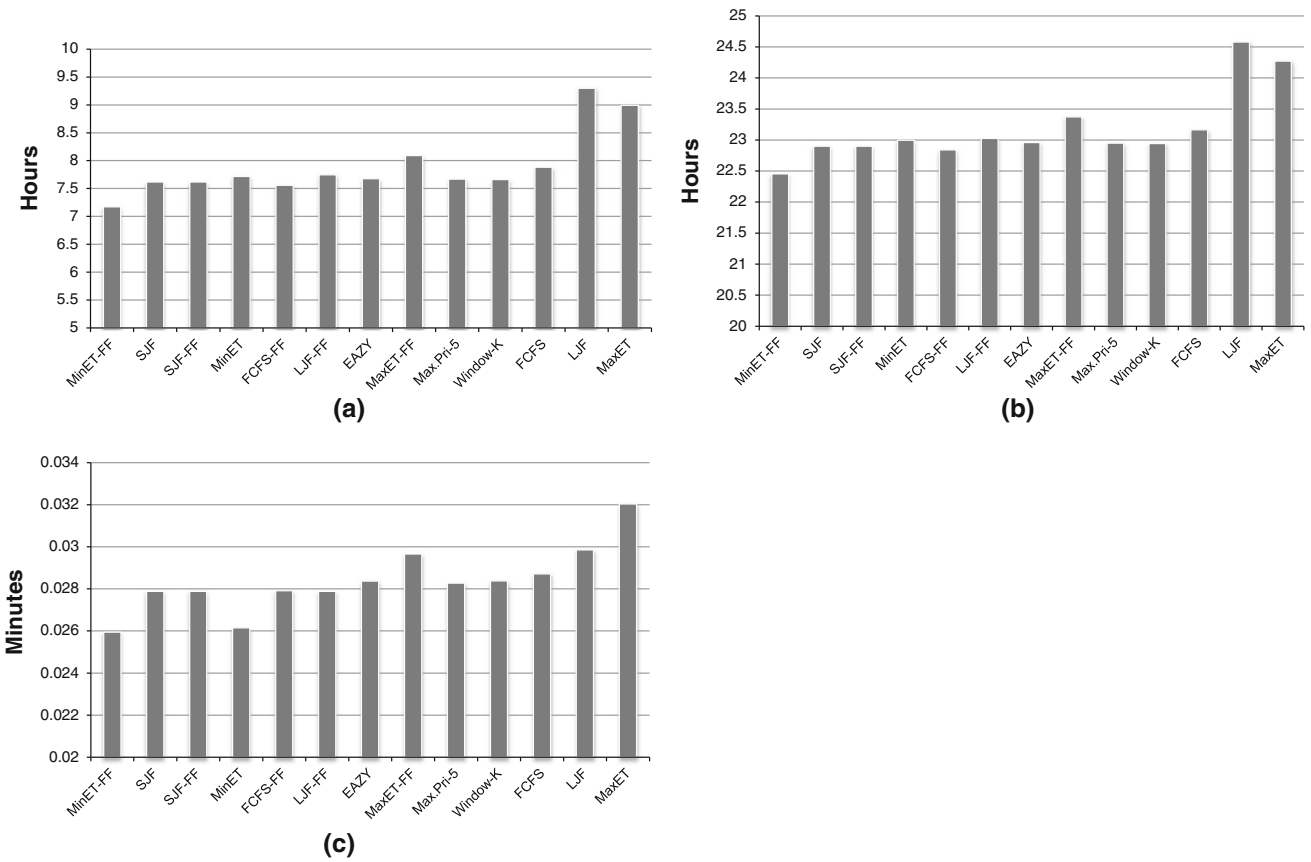

Fig. 10 Comparison results of all policies with overall workloads in fourth observation: **a** mean queue-time, **b** mean response-time, and **c** mean slowdown

We found that the policies in Class-I produce better results as compared to the policies in Class-II and Class-III. The policies in Class-II are superior to the policies in Class-III. The reasons are explained in the following paragraph.

The major reason that SJF policy is superior to the rest policies in all of the figures is due to the fact of the large number of Narrow jobs. Alternatively, for the MinET and MaxET policies, we have already highlighted that (a) MinET and MaxET take into account user's estimated running time, and (b) half of the jobs are Long and the rest are Short. Because of the aforementioned factors, the MinET and MaxET may produce almost similar results. However, the results shown in Fig. 5 reveal that the aforementioned does not hold. This is explained by the fact that the number of Narrow jobs is quite larger than that of Wide jobs. Because MinET and MinET-FF take into account the above fact, it results in better solutions against MaxET and MaxET-FF. The Figs. 7 and 8 present the results for Narrow jobs respectively. The number of jobs is 36 % in Fig. 7 and 42 % in Fig. 8 of the total jobs in the overall workload. In Fig. 7, the jobs belong to SN category, while in Fig. 8 the jobs belong to LN category. In both of the figures, because the policies in the Class-I schedule the jobs according to SJF and minimum execution time, the policies of Class-I dominate the policies belonging to other classes. The results for Wide jobs (i.e., SW and LW) are shown in Figs. 9 and 10, and their job percentage is 10 and 11 %, respectively. In Fig. 9, the scheduling policies of both Class-I and Class-II exhibit better results as compared to the policies in Class-III. Figure 10 depicts that the results are almost same for all of the scheduling policies.

6 Energy efficient job scheduling policies

In the previous section, we evaluated several resource allocation policies and categorized their results based on their performance. As growing rate of the power consumption in large-scale parallel systems is dominant part of the user's and owner's budget, an energy efficient scheduling policy is needed together with a resource allocation policy [53]. In this section, we incorporate energy efficiency in three policies selected from each class (discussed in Subsect. 5.3). In the following sections, we discuss power model, scheduling criteria, and problem statement, and then we explain our

Table 6 Performance of job scheduling policies

Policies	Performance	Rank
MinET, MinET-FF, SJF, SJF-FF	Best	1st Class
FCFS-FF, LJF-FF, MaxET-FF, EAZY, Max_Pri	Average	2nd Class
FCFS, LJF, MaxET, Window-K	Worst	3rd Class

proposed energy efficient policies with their results and discussion.

6.1 Energy model

The energy model for DVFS that is based on the power consumption model in CMOS (complimentary metal-oxide semiconductor) logic circuit [49] has been used in this paper. The machine M_i consumes the power capacitive P_{ijk} for computing the task T_{jk}. The power is calculated in the following way.

$$P_{ijk} = A.C.v^2.f, \tag{1}$$

where A represents the number of switches per clock cycle, C represents total capacitance load, v represents supply voltage, and f represents machine frequency. The value of switches per clock cycle and total capacitance load are determined at the design time [49]. While the combined reduction of the clock frequency and supply voltage lies in the DPM technique called DVFS.

Brown et al. [31] assumed that frequency of each machine is proportionate to its processing speed. The energy consumed by the machine can be reduced by decreasing the frequency and supply voltage. We assume that each machine in the data center is equipped with DVFS module [31] and belonged to one of the specific machine class. Table 7 shows the configuration of different classes that are created in such a way that the total computation size must meet with original setup. For instance in original setup, total number of CPUs was 2,090 with 3.0 GHz speed (i.e., total computational size was 6,270.0 GHz). The parameters for DVFS levels are presented in Table 8. The table shows the values of the frequency and voltage parameters of twelve DVFS levels of three machine classes [53]. Each class of the machines is comprised different computing capacity (i.e., number of computational cycles per second).

When lowering the voltage, the operation frequency of the machine decreases and the computational time of the task executed on the machine increases. We apply DVFS at the behavioral level where the supply voltage of processing unit is constant for a task during execution but it may be different for other task.

Table 7 CPU classes

Machine	Total	Speed in MHz	Computation Size
Class-I	1,175	3,000	3525000
Class-II	520	3,600	1872000
Class-III	220	4,000	880000
Total	1,915	10,600	6277000

Table 8 DVFS module

Level	Freq.	Class-I (V)	Class-II (V)	Class-III (V)
0	1.0	1.3	1.9	2.5
1	0.9	1.2	1.7	2.3
2	0.8	1.1	1.5	2.1
3	0.7	1.0	1.3	1.9

6.2 Energy efficient scheduling criteria

Each job is a collection of tasks, which comprised deadline and workload. All tasks in a job have same value of deadlines and workloads. The scheduler randomly generates the job's deadline for each job in meta-task event. The randomly generated value of the job deadline must range between the running time of the task on minimum power level of a lowest class machine and the running time of the task on maximum power level of a highest class machine. A load wl of the task T_{jk} expressed in Millions-of-Instructions execute on a machine M_i with computing capacity cc expressed in Millions-of-Instruction-Per-Seconds. We use job running time (i.e., used for previous setup) as job's computational cycles. The overall workload WL and computing capacity CC is denoted by: $WL = [wl_1 \ldots, wl_n]$ and $CC = [cc_1 \ldots, cc_n]$. Estimation Time for Completion (ETC) of the task T_{jk} for executing on the machine M_i is created by workload wl_{jk} divided by computing capacity cc_i. ETC is denoted by $ETC_{ijk} (i \in m, j \in n, k \in o)$ and calculated in the following way.

$$ETC_{ijk} = \frac{wl_{jk}}{cc_i} \tag{2}$$

For a scheduled S task-machine pair (t_{jk}, M_i), the task t_{jk} completion time and energy consumption on the machine M_i is minimized and not violate the task's deadlines. Suppose we are given processing units and a set of jobs along with a set of tasks. Each job must be placed to processing units such that the job's constraints (i.e., number of tasks and deadline) are fulfilled. The running time of job on processing unit must be less than deadline. The placement of each job and its task must be placed as much as possible on minimum level of lowest class machine as energy efficiency must be achieved.

6.3 Problem statement

This paper addresses the following main scheduling objectives as follows.

- *Job placement must meet job's requirement (i.e., each job needs number of CPUs to execute).*
- *Minimize total energy consumed by processing unit and makspan (finishing time of latest job).*

The mathematical equation for energy consumption objective is given by:

$$minimize \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} \tag{3}$$

where p is the power, i represents a machine, and j represents a task.

6.4 Proposed energy efficient policies (SJF-EE, LJF-EE, and LJFFF-FF)

From the Subsect. 5.3, we found the performance of each job scheduling policy evaluated in different classes. A scheduling policy from each class has been selected to modify with DVFS technique to analyze the behavior of job scheduling policy under the specific workload characteristics. The selection of the policy from best, average, and worse policies will help to analyze the energy efficiency strategy. For the energy efficient policies, we use the same workload described in Sect. 4.1, which were simulated for resource allocation policies (i.e., without DVFS technique). The modified policies that incorporate energy efficiency are: (1) best result producer SJF, (2) average result producer LJF-FF, and (3) worst result producer LJF. Specifically their names when combined with energy efficient policies become: (a) SJF energy efficient, SJF-EE, (b) LJF energy efficient, LJF-EE, and (c) LJF-FF energy efficient, LJFFF-EE.

Algorithm 1 shows the pseudo-code for the SJF-EE. To realize the dynamic and real environment in our setup, the jobs submitted to the system are queued, and the scheduling policy may check the queue periodically (i.e., 10 s time interval). In each time interval, the SJF-EE policy sorts the machines according to their minimum ready time (Line 3), and then Line 4 sorts the jobs in decreasing order of their CPU demand (i.e., number of tasks). For the job placement on the machine, each job in the queue is examined such that (a) number of tasks in a job must be less than or equal to the number of ready machines in any machine' class (Line 6), (b) to place the job on lowest machine class (Line 7), and (c) job estimation time for completion must be less than or equal to job's deadlines (Line 9). Consequently, energy consumption of the system will be minimized with little running time overhead.

Algorithm 1. Smallest Job First Energy Efficient (SJF-EE)

Input: Processing units, initialized to its power using $DVFS = \{dvfs_1, dvfs_2, ..., dvfs_n\}$ and Jobs J_i $(i = 1, \ldots, n)$
Output: A job with its task mapped on the machines with possible minimum energy and makespan cost.

```
1. Set loop_interval = start_time
2. while batch_interval do
3.  Sort the ready machines M_j (j = 1, ..., m) in decreasing order of their power consumption (i.e., lowest
    class then minimum level)
4.  Sort the submitted jobs J_i (i = 1, ..., n) in decreasing order of their CPU demand, number of
    tasks T_ik (k = 1, ..., p)
5.  for each J_i do
6.   if number of ready machines m in any class >= number of task p in J_i then
7.    for each power_level do
8.     Calculate job ETC estimation time for completion
9.     if ETC_ij ≤ J_i.deadlines then
10.     Place the job on the machine
11.     break
12.    end if
13.   end for loop
14.  end if
15. end for loop
16. loop_interval = loop_interval + 10 seconds
17. end while
```

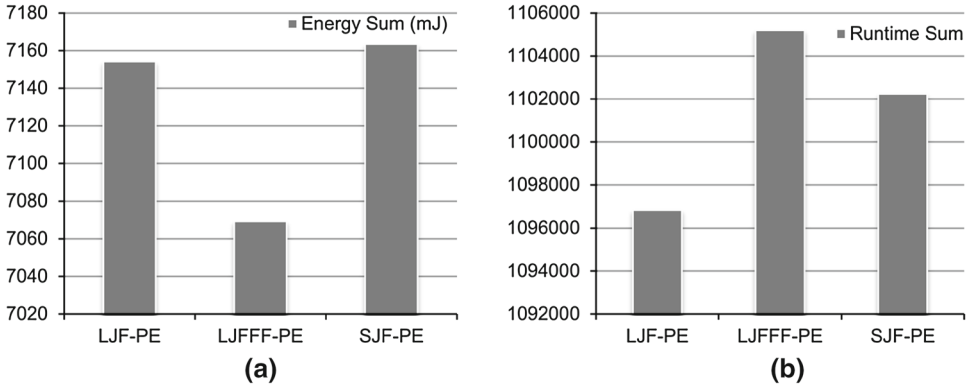

Fig. 11 Comparison results of energy efficient policies with overall workloads: **a** energy sum and **b** runtime sum

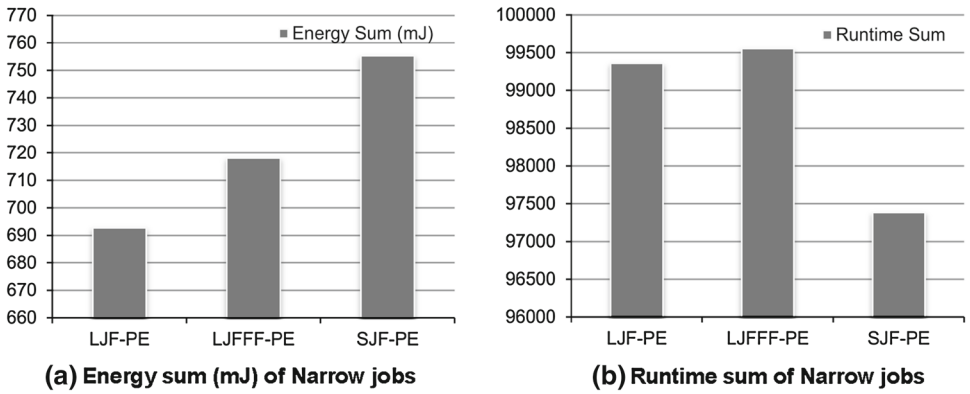

Fig. 12 Comparison results of energy efficient policies with one CPU workloads: **a** energy sum and **b** runtime sum

Other modified scheduling policies for energy efficiency, LJF-EE and LJFFF-EE, perform in the same way as SJF-EE. Their difference is that LJF-EE sorts the jobs in increasing order of their CPU demand, and LJFFF-EE finds the jobs that can be fit on first idle machines to increase the resource utilization (described in the Sect. 5).

6.5 Results and discussion of energy efficient policies

In this section, we present the result produced by the proposed energy efficient scheduling policies. We consider two metrics to evaluate the performance of energy efficient scheduling policies: (a) total energy consumption of the system and (b) total running time (i.e., makespan) of the entire workload, as shown in Fig. 11a, b, respectively. In the figures, the results reveal that less energy has been consumed by LJFFF-PE against others but it is also observed that the overall running time (makespan) has been increased. While the SJF-PE, modified policy of SJF ranked in first Class in the previous experiment (see Sect. 5.3), does not produce satisfactory results in terms of both total energy consumption of the system and total runtime of the entire workload. On the other extreme, the extended policy of LJF, LJF-PE consumes less energy than SJF-PE and also minimizes overall running time as compared to other policies. The reason behind the fact that LJFFF-PE provides best performance in aspect of low energy consumption is that in each scheduling batch interval Wide jobs are taking chance to execute on the low performance machines. While the reason behind the fact of LJFFF-PE increasing overall running time is that the scheduler tries to place the jobs on low level of the power of the machine, which forces to increase the job execution time. The aforementioned reasons are also considerable in SJF-PE producing unsatisfactory results, which increases energy consumption of the systems and overall running time of the workload.

After finding results for entire workload, we further analyze the workload for Narrow and Wide jobs (i.e., discussed in the Sect. 4.2). The results are shown in Figs. 12 and 13 for Narrow jobs and Wide jobs, respectively. As the primary goal of energy efficient technique in job scheduling policy is to minimize the power consumption with reasonable overhead of running time, here LJF-PE produces better results against other policies. In both scenarios, Narrow and Wide jobs, LJF-PE consumed less amount of energy with little overhead of running time, shown in both figures. While in SJF-PE in both scenarios, SJF-PE did not produce better results against other policies. From the above evaluation of the energy efficient policies, we found that LJF-PE provides affordable results in aspect of both low energy consumption of the system and minimum running time of entire workload.

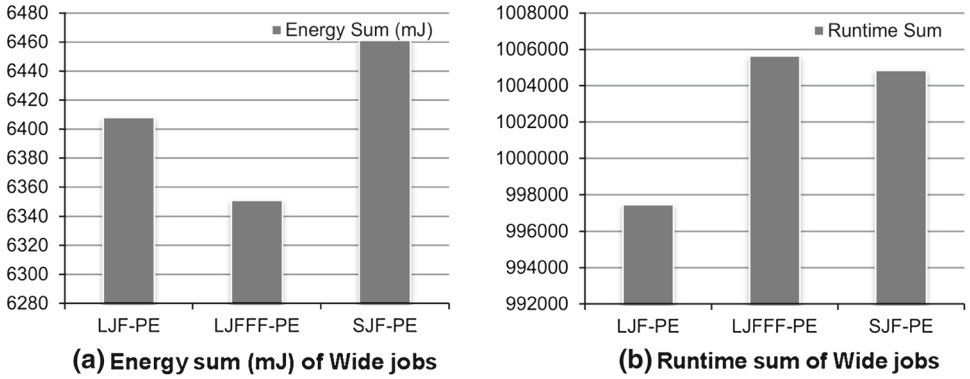

Fig. 13 Comparison results of energy efficient policies with more than one CPU workloads: **a** energy sum and **b** runtime sum

In the above sections, we thoroughly discussed the characterization and analysis of the real data center workload and several resource allocation job scheduling policies, and then we modified the selected policies for energy efficiency. We investigated that most of the job scheduling policies are affected significantly by certain workload characteristics. For instance: (a) Due to the large number of Narrow jobs, MinET and SJF when combined with FF technique exhibit better performance compared to other policies. (b) A large number of small jobs in the workload can stop the MaxET policy for producing at least same results compared to MinET policy with certain job characteristics. (c) While the Wide jobs taking chance to execute on the low performance machine with low level of the power make LJF-PE best performance in terms of energy consumption.

7 Conclusions

This paper is a comparative study on resource allocation and energy efficient job scheduling strategies in large-scale parallel computing environment. Because of the growing rate of power and cooling in the aforementioned environments is dominant part of the user's and owner's budget, resource allocation job scheduling techniques must consider the energy efficiency while maintaining desired QoS constraints. However, considering energy efficiency together QoS such as queue time and response time, a job scheduling problem is more difficult to design. In this paper, we have studied a total of thirteen job scheduling policies to analyze and compare their results. The set of job scheduling policies considered in this work includes (a) priority-based, (b) FF, (c) backfilling, and (d) window-based techniques. We used three metrics (queue time, response time, and slowdown ratio) to evaluate the performance of the job scheduling policies. All of the policies were extensively simulated and compared using a real data center workload exhibited a wide range of job heterogeneity. Based on the results of their performance, we then incorporate energy efficiency in three job scheduling policies. We found that most of job scheduling policies are affected significantly by certain workload characteristics, such are: (a) Due to the large number of Narrow jobs, MinET and SJF when combined with FF technique exhibit better performance compared to other policies. (b) A large number of Small jobs in the workload avoid the MaxET policy to produce at least same results compared to MinET policy with certain job characteristics. (c) The Wide jobs taking chance to execute on the low performance machine with low level of the power, which helps LJF-PE to produce best results in terms of energy consumption.

Our analysis revealed that a single policy is not sufficient for resource management in parallel computing environments. Such environments need to implement dynamic and adaptive scheduling policies.

Acknowledgments The shorter version [29] of the paper has been published in the proceedings of the 11th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA) co-located with TrustCom and IUCC, Melbourne, Australia on July 2013. Aftab Ahmed Chandio's work was partly supported for his PhD studies in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China. Nikos Tziritas's work is partly supported by Chinese Academy of Sciences. Samee U. Khan's work was partly supported by the Young International Scientist Fellowship of the Chinese Academy of Sciences, (Grant No. 2011Y2GA01).

References

1. Iosup, A., Ostermann, S., Yigitbasi, M.N., Prodan, R., Fahringer, T., Epema, D.H.J.: Performance analysis of cloud computing services many-task s scientific computing. IEEE Trans. Parallel Distrib. Syst. **22**(6), 931–945 (2011)
2. Xu, C.Z., Rao, J., Bu, X.: URL: a unified reinforcement learning approach for autonomic cloud management. J. Parallel Distrib. Comput. **72**(2), 95–105 (2012)
3. Wei, T., Dongxu, R., Zhiling, L., Narayan, D.: Adaptive metric-aware job scheduling for production supercomputers. In: 41st International Conference on Parallel Processing Workshops, (ICPPW'12), 2012, pp. 107–115
4. Khan, S.U.: A self-adaptive weighted sum technique for the joint optimization of performance and power consumption in data centers. In: 22nd International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS), Louisville, KY, USA, 2009, pp. 13–18

5. Khan, S.U., Ardil, C.: A weighted sum technique for the joint optimization of performance and power consumption in data centers. Int. J. Electr. Comput. Syst. Eng. **3**(1), 35–40 (2009)
6. Khan, S.U., Ahmad, I.: Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation. In: 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece, 25–29 April 2006, p. 10
7. Khan, S.U.: A goal programming approach for the joint optimization of energy consumption and response time in computational grids. In: 28th IEEE International Performance Computing and Communications Conference (IPCCC), Phoenix, AZ, USA, 2009, pp. 410–417
8. Braun, T.D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
9. Arndt, O., et al.: A comparative study of online scheduling algorithms for networks of workstations. Cluster Comput. **3**(2), 95–112 (2000)
10. Baraglia, R., et al.: A multi-criteria job scheduling framework for large computing farms. J. Comput. Syst. Sci. **79**, 230–244 (2013)
11. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with the Parallel Workloads Archive. Technical Report 2012–6, School of Computer Science and Engineering, The Hebrew University of Jerusalem, 2012
12. Feitelson, D.G., Workload modeling for performance evaluation. Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science, vol. 2459, pp. 114–141 (2002)
13. Steven, H., David, S., O', Donnell, T. Analysis of the Early Workload on the Cornell Theory Center IBM Sp2. In: ACM SIGMETRICS Conference on Measurement and Modeling of Computer, System, 1996; Poster
14. Srinivasan, S., Kettimuthu, R., Subramani, V., Sadayappan, P.: Selective Reservation Strategies for Backfill Job Scheduling, Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science, vol. 2537, pp. 55–71 (2002)
15. Maheswarana, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput. **59**(2), 107–131 (1999)
16. Feitelson, D.G., et al.: Theory and Practice in Parallel Job Scheduling. Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science, vol. 1291, pp. 1–34 (1997)
17. Litka, D.A.: The ANLIIBM SP Scheduling System. Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, vol. 945, pp. 295–303 (1995)
18. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
19. Ababneh, I., Bani-Mohammad, S.: A new window-based job scheduling scheme for 2D mesh multicomputers. Simul. Model. Pract. Theory **19**(1), 482–493 (2011)
20. Chandio, A.A., Zhu, D., Sodhro, A.H.: Integration of Inter-Connectivity of Information System (i3) using Web Services. International MultiConference of Engineers and Computer Scientists (IMECS). Lecture Notes in Engineering and Computer Science, vol. 2195, pp. 651–655 (2012)
21. Chapin, S., Cirne, W., Feitelson, D.G., Jones, P., Leutenegger, S., Schwiegelshohn, U., Smith, W., Talby, D.: Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science, vol. 1659, pp. 66–89 (1999)
22. Parallel Workload Archive. http://www.cs.huji.ac.il/labs/parallel/workload/
23. Lo, M., Mache, J., Windisch, K.J.: A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling. Workshop on Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science, vol. 1459, pp. 25–46 (1998)
24. Lublin, U., Feitelson, D.: The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J. Parallel Distrib. Comput. **63**(11), 1105–1122 (2003)
25. Chiang, S.-H., Vernon, M.K.: Characteristics of a large shared memory production workload. Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer-Science, vol. 2221, pp. 159–187 (2001)
26. Downey, A.B.: A parallel workload model and its implications for processor allocation. In: 6th IEEE International Symposium on High Performance Distributed Computing, pp. 112–123 (1997)
27. Chandio, A.A., Yu, Z., Syed, F.S., Korejo, I.A.: A Case Study on Job Scheduling Policy for Workload Characterization and Power Efficiency, Sindh University Research Journal (Science Series). **45**(A-1), 23–28 (2013)
28. Wang, L., Khan, S.U., Dayal, J.: Thermal aware workload placement with task-temperature profiles in a data center. J. Supercomput. **61**(3), 780–803 (2012)
29. Chandio, A.A., et al.: A Comparative Study of Scheduling Strategies in Large-scale Parallel Computational Systems. In: 11th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA) co-located with TrustCom and IUCC, Melbourne, Australia, July 2013, pp. 949–957 (2013). doi:10.1109/TrustCom.2013.116
30. Fan, X., Weber, W.-D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. SIGARCH Comput. Archit. News **35**(2), 13–23 (2007)
31. Brown, R., et al.: Report to Congress on Server and Data Center Energy Efficiency. Public Law 109–431, 2008
32. Koomey, J.G.: Worldwide electricity used in data centers. Environ. Res. Lett. **3**(3), 034008 (2008)
33. Koomey, J.: Growth in Data Center Electricity use 2005 to 2010. Analytics Press, Oakland (2011)
34. Shuja, J., et al.: Energy-efficient data centers. Computing **94**(12), 973–994 (2012)
35. Masanet, E.R., et al.: Estimating the energy use and efficiency potential of U.S. data centers. Proc. IEEE **99**(8), 1440–1453 (2011)
36. Benini, L., Bogliolo, A., De Micheli, G.: A survey of design techniques for system-level dynamic power management. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **8**(3), 299–316 (2000)
37. Chen, G., et al.: Energy-aware server provisioning and load dispatching for connection-intensive internet services. In: 5th USENIX Symposium on Networked Systems Design and Implementation, USENIX Association: San Francisco, California, 2008, pp. 337–350
38. Liu, J., et al.: Challenges Towards Elastic Power Management in Internet Data Centers. In: 29th IEEE International Conference on Distributed Computing Systems Workshops, 2009, IEEE Computer Society, pp. 65–72
39. Kliazovich, D., Bouvry, P., Khan, S.U.: DENS: Data Center Energy-Efficient Network-Aware Scheduling. in Green Computing and Communications (GreenCom). In: IEEE/ACM International Conference on Green Computing and Communications & IEEE/ACM International Conference on Cyber, Physical and Social Computing, Hangzhou, China, pp. 69–75 (2010)
40. Meisner, D., Wenisch, T.F.: DreamWeaver: architectural support for deep sleep. SIGARCH Comput. Archit. News **40**(1), 313–324 (2012)
41. Isard, M., et al.: Dryad: distributed data-parallel programs from sequential building blocks. SIGOPS Oper. Syst. Rev. **41**(3), 59–72 (2007)

42. Nedevschi, S., et al.: Reducing network energy consumption via sleeping and rate-adaptation. In: 5th USENIX Symposium on Networked Systems Design and Implementation, USENIX Association: San Francisco, California, 2008, pp. 323–336
43. Bo, L., et al.: EnaCloud: An Energy-Saving Application Live Placement Approach forCloud Computing Environments. In: IEEE International Conference on Cloud Computing (CLOUD '09), Bangalore, India, pp. 17–24 (2009)
44. Weiser, M., et al.: Scheduling for reduced CPU energy. In: 1st USENIX conference on Operating Systems Design and Implementation. USENIX Association: Monterey, California, 1994, p. 2
45. Gruian, F., Kuchcinski, K.: LEneS: task scheduling for low-energy systems using variable supply voltage processors. In: Asia and South Pacific Design Automation Conference (ASP-DAC), Yokohama, Japan, pp. 449–455 (2001)
46. Horvath, T., et al.: Dynamic voltage scaling in multitier web servers with end-to-end delay control. IEEE Trans. Comput. **56**(4), 444–458 (2007)
47. Zhong, X., Xu, C.-Z.: System-wide energy minimization for real-time tasks: lower bound and approximation. ACM Trans. Embed. Comput. Syst. **7**(3), 1–24 (2008)
48. Meisner, D., Gold, B.T., Wenisch, T.F.: PowerNap: Eliminating Server Idle Power. In: Architectural Support for Programming Languages and Operating Systems (ASPLOS '09), Washington, DC, 2009, pp. 205–216
49. Kołodziej, J., et al.: Security, energy, and performance-aware resource allocation mechanisms for computational grids. Future Gener. Comp. Sy. **31**, 77–92 (2014). doi:10.1016/j.future.2012.09.009
50. Bilal, K., et al.: A survey on green communications using adaptive link rate. Cluster Comput. **16**(3), 575–589 (2013)
51. Kliazovich, D., Bouvry, P., Khan, S.U.: DENS: data center energy-efficient network-aware scheduling. Cluster Comput. **16**(1), 65–75 (2013)
52. Kołodziej, J., et al.: Hierarchical genetic-based grid scheduling with energy optimization. Cluster Comput. **16**(3), 591–609 (2013)
53. Kołodziej, J., Khan, S.U., Wang, L., Zomaya, A.Y.: Energy efficient genetic-based schedulers in computational grids. Concurr. Comput. (2012). doi:10.1002/cpe.2839

Aftab Ahmed Chandio is a doctoral student at Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong, China. He is also a lecturer of Institute of Mathematics and Computer Science at the University of Sindh, Jamshoro, Sindh, Pakistan. His research interests include cloud computing, parallel and distributed systems, scheduling, workload characterization, and map-matching strategies for GPS trajectories.

Kashif Bilal is a doctoral student at North Dakota State University, Fargo, ND, USA. His research interests include cloud computing, data center networks, green computing, and distributed systems. He is a student member of IEEE.

Nikos Tziritas received a Ph.D. degree from the University of Thessaly, Greece, in 2011. He is currently a postdoctoral researcher in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong, China. His research has been on scheduling, load-balancing and replication in CDNs, as well as energy optimization and resource management in WSNs and cloud computing systems.

Zhibin Yu received his PhD degree in computer science from Huazhong University of Science and Technology China in 2008. He spent one year as a visiting scholar in the Laboratory of Computer Architecture, University of Texas at Austin. He is currently an associate professor at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong, China. His research interests include microarchitecture simulation, computer architecture, workload characterization and generation, performance evaluation, multicore architecture, and virtualization technologies. In 2005, he won first prize in the HUST Young Lecturers Teaching Contest, and in 2003, he won second prize in the teaching quality assessment of HUST. He is a member of IEEE and ACM.

Qingshan Jiang received the PhD degree in mathematics from Chiba University, Japan, in 1996 and the PhD degree in computer science from University of Sherbrooke, Canada, in 2002. He is a professor at Shenzhen Key Laboratory for High Performance Data Mining, Shenzhen Institutes of Advanced Technology of Chinese Academy of Sciences, Shenzhen, Guangdong, China. He has been a professor at Xiamen University since 2003. In 1999-2000, he worked as a Postdoc fellow at the Fields Institute for Research in Mathematical Sciences, University of Toronto, Canada. His research interests include data mining, cloud computing security, pattern recognition, statistical analysis, and fuzzy logic.

Samee U. Khan received a Ph.D. degree from the University of Texas, Arlington, TX, USA, in August 2007. Currently, he is assistant professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Prof. Khan's research interests include optimization, robustness, and security of: cloud, grid, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. His work appears in over 225 publications He is a senior member of IEEE, and a fellow of IET. For more information, please visit: http://sameekhan.org/.

Cheng-Zhong Xu received his Ph.D. degree from the University of Hong Kong in 1993. He is currently a tenured professor of Wayne State University and the Director of the Institute of Advanced Computing and Data Engineering of Shenzhen Institutes of Advanced Technology of Chinese Academy of Sciences, Shenzhen, Guangdong, China. His research interest is in parallel and distributed systems and cloud computing. He has published more than 200 papers in journals and conferences. He was the Best Paper Nominee of 2013 IEEE High Performance Computer Architecture (HPCA), and the Best Paper Nominee of 2013 ACM High Performance Distributed Computing (HPDC). He serves on a number of journal editorial boards, including IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Cloud Computing, Journal of Parallel and Distributed Computing and China Science Information Sciences. He was a recipient of the Faculty Research Award, Career Development Chair Award, and the President's Award for Excellence in Teaching of WSU. He was also a recipient of the "Outstanding Overseas Scholar" award of NSFC. For more information, visit http://www.ece.eng.wayne.edu/~czxu.