2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications

A Comparative Study of Job Scheduling Strategies in Large-scale Parallel Computational Systems

Aftab Ahmed Chandio[1,3], Cheng-Zhong Xu[1,2], Nikos Tziritas[1]
[1]Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences, Shenzhen, China
[2]Wayne State University, Detroit, USA
[3]University of Sindh, Jamshoro, Pakistan
{aftabac, cz.xu, nikolaos}@siat.ac.cn

Kashif Bilal and Samee U. Khan
Department of Electrical and Computer Engineering, North Dakota State University, Fargo, USA
{kashif.bilal, same.khan}@ndsu.edu

***Abstract*—With the advent of High Performance Computing (HPC) in the large-scale parallel computational environment, job scheduling and resource allocation techniques are required to deliver the Quality of Service (QoS) and resource management. Therefore, job scheduling on a large-scale parallel system has been studied to: (a) minimize the queue time and response time, and (b) maximize the overall system utilization. We compare and analyze thirteen job scheduling policies to analyze their behavior. The set of job scheduling policies include: (a) priority-based policies, (b) first fit, (c) backfilling techniques, and (d) window-based policies. All of the policies are extensively simulated and compared. A real data center workload comprised of 22385 jobs is used for simulation. We analyze the: (a) queue time, (b) response time, and (c) slowdown ratio to evaluate the policies. Moreover, we present a comprehensive workload characterization that can be used as a tool for optimizing system's performance and for scheduler design. We investigate four categories of the workload characteristics including: (a) Narrow, (b) Wide, (c) Short, and (d) Long 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- Large-scale Parallel Computational Systems; Job Scheduling; Workload Characterization; Data center;

I. INTRODUCTION

Scientific organizations are increasingly adopting high performance computing (HPC) for solving large problems, which increases the computational and storage needs. In the last decade, various scientific organizations have spent ample budget to carry out research projects using supercomputers [1]. Because of the fact that supercomputers are unaffordable for various organizations, they 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 variety of services to facilitate end-users.

In the large-scale parallel computational environments, the end-users submit their requests unaware of the resource allocation. Unusually, these requests are complex jobs, which may be computation-intensive (i.e., job demands more CPU time) or data-intensive (i.e., job demands more storage space and communication). Moreover, these requests may require different levels of Quality of Service (QoS), including job turnaround time and queue time. Furthermore, the large-scale parallel computational environments consist of **(a)** the mixture of applications and **(b)** a pool of finite resources meeting the demands of end-users.

For the above identified factors, RPs pay considerable attention to resource management to deliver the required QoS and enhance system utilization. Several researchers focus on resource management to optimize the system performance considering various QoS constraints. The job scheduling policy is one of the major components of resource management system for solving the above problems in such environments. 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, and an active research direction for resource management. The scheduling policy should behave equally well considering the resource heterogeneity and the workload variability.

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

978-0-7695-5022-0/13 $26.00 © 2013 IEEE
DOI 10.1109/TrustCom.2013.116

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 assumptions and job parameters (i.e., number of jobs, tasks in each job, estimated time of job execution, and submission time of each job) remain same to maintain a fair comparison. The aforementioned policies are analyzed by 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.

The rest of the paper is organized as follows. Section II states the related work, followed by the comprehensive characterization and analysis of data center workload in Section III. Section IV presents the simulation results of the real data center workload using several well-known job-scheduling policies. Performance analysis and comparison analysis is detailed in Section V. Finally, Section VI concludes the paper and highlights future research directions.

II. Related Work

In large-scale parallel computational 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 the different QoS constraints. For instance, author in [3] propose metric-aware scheduling, which enables the scheduler to balance competing scheduling goals represented by different metrics, such as job waiting time, fairness, and system utilization. Khan used a self-adaptive weighted sum technique in [4], [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. Tracy *et al.* [8] studied eleven static heuristics for mapping a class of independent task onto heterogeneous distributed computing system. The authors analyzed and implemented a collection of task mapping policies under one set of common assumptions. The author in [9] compared the performance of six online scheduling algorithms for batch jobs by simulating keeping in consideration the three objective functions including makespan, average flowtime, and maximum wait-time. This paper addresses the similar problem of system performance through a comparative study of job scheduling strategies.

Additionally, these environments respond to 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 large-scale parallel computational environments consist of a mixture of applications that demand different resources, which result in highly variable workloads [13]. Feitelson *et al.* [23] described parallel workload models in detail, and explained standard workload format for large-scale parallel computing environments. The author used publically available parallel workloads from [24] that consist of various real world workloads obtained from several large-scale parallel computers. These workloads have been characterized and analyzed in [26], [27], and observed similarities and differences in workload characteristics, such as different arrival patterns in peak or non-peak intervals and "power-of-two" number of processor requirement for job execution. 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 collected the log files 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.

III. Charaterization of Data center Workload

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, 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 (TCO). Aforementioned questions mandate the RPs to select appropriate resource management techniques, such as job scheduling policies (see Section IV.A).

A. *Workload's information*

We characterized a real data center workload from the Center for Computational Research (CCR) 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 20 February 2009 to 22 March 2009. A total of 22385 jobs were executed on more than 1000 dual processor nodes [29], [30]. A complete specification of the data center is presented in Table I.

TABLE I. FULL SPECIFICATION OF DATA CENTER

Time Duration	20 Feb. 2009 to 22 Mar. 2009
Total Jobs ran out on DC	22385
Total Distinct Nodes	1045
Processor name	1056 Dell PowerEdge SC1425 nodes
Processor Speed	3.0GHz or 3.2GHz
OS	x86 64 Linux
Peak performance	13 TFlop/s

B. *Job's information*

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. Fig. 1 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 Section IV.A) [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. Fig. 2 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% percent of the total jobs, whereas, 21% of the total jobs are in Wide category.

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

TABLE II. BREAKDOWN DISTRIBUTION FOR JOB LENGTH

Job length	No. of jobs	% of jobs
< 1 hour (Short)	10428	46.58
> 1 hour (Long)	11957	53.42
Total	22385	100

Table III and IV present the classification of the jobs in **(a)** Short and **(b)** Long categories. It can be observed that most of the Short jobs (around 86%) are executed within 18 minutes, while around 66.6% of the long jobs are executed within 12 hours.

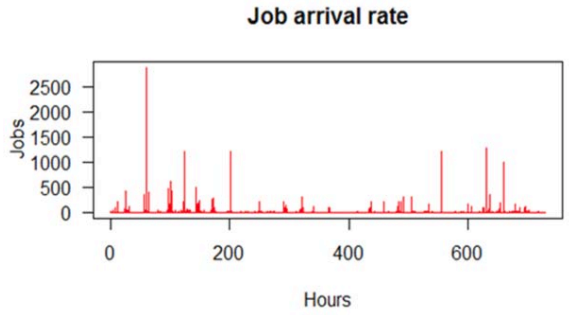


Fig. 1 Jobs arriving per hour

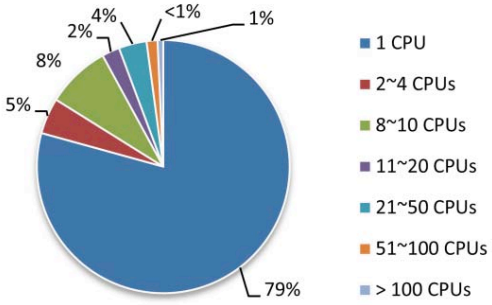


Fig. 2: Jobs breakdown according to number of CPUs

TABLE III. BREAKDOWN DISTRIBUTION OF THE SHORT JOBS

Job length	No. of jobs	% of jobs
1 Min.	1239	5.53
2~5 Min.	504	2.25
6~10 Min.	1485	6.63
11~15 Min.	1858	8.30
16 Min.	1166	5.21
17 Min.	1777	7.94
18 Min.	968	4.32
18~59 Min.	1431	6.39
Total	10428	46.58

TABLE IV. BREAKDOWN DISTRIBUTION OF THE LONG JOBS

Job length	No. of jobs	% of jobs
1~1.5 hours	1368	6.11
1.5~2 hours	1339	5.98
2~6 hours	1210	5.41
7 hours	1754	7.84
8 hours	1785	7.97
8~12 hours	511	2.28
12~15 hours	667	2.98
16 hours	1408	6.29
16~18 hours	447	2.00
18~24 hours	595	2.66
24~48 hours	511	2.28
>48 hours	362	1.62
Total	11957	53.42

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

TABLE V. 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 hours	13.23	0.67	0.68	**14.57**
	5~8 hours	18.43	0.15	0.16	**18.74**
	> 9 hours	10.73	8.27	1.10	**20.11**
Total Jobs		**79.27**	15.03	5.70	100

In the above workload characterization, the analysis revealed important job characteristics, such as that jobs arrival rate does not follow any trend and heterogeneity in both job size and requirements. Such heterogeneity in workloads dictates to analyze the effect of various scheduling policies in such scenarios.

IV. EXPERIMENTAL SETUP

This section presents the experimental setup regarding the set of scheduling policies. All of the policies are used to schedule the aforementioned workload to figure out the best job scheduling policies for optimizing the system performance.

A. Job scheduling policies

The scheduler is a major component for managing the resources of large-scale parallel environments. A policy in a scheduler is used to assign jobs to resources in specific time intervals such that the capacity of resources should meet jobs' needs [10]. Suppose m denotes the total number of machines, $M_j\ (j = 1, \dots, m)$ to process n jobs $J_i\ (i = 1, \dots, n)$. A job J_i is a program submitted by a user at a specific time interval (submit-time). Each job contains one or more tasks $J_i = t_j (j = 1, \dots, k)$, 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. 4. The corresponding scheduling policy problem is to find an optimal schedule process subject to various constraints.

Schedulers achieve the scheduling process by using either a static or a dynamic scheduling method. In the *static* case, the set of jobs are known a-priori, while the *dynamic* one performs scheduling at job arrival. Due to the fact that jobs' arrival rate and the status of some nodes (off-line or online) may change without any a-priori knowledge, the *dynamic* scheduling method is required [15].

Moreover, 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 its arrival, while the *batch* mode schedulers collect the jobs in a queue until a 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. The meta-tasks are examined by the corresponding scheduling policy at prescheduled times called scheduling events. The scheduler event 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 including banking system, health system, virtual campuses, and bio-informatics

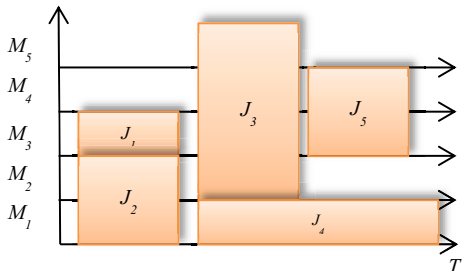


Fig. 4 A Gantt chart for Job Scheduling process

applications. However, the batch scheduling method and the independent nature of jobs is hard to be solved [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 different 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 application (i.e., rigid, moldable, evolving, and malleable job flexibility). There is a differentiation between rigid and moldable/evolving/malleable jobs. In case of a rigid job, the number of processors that are assigned to a job does not change throughout the execution. On the other hand, in case of a *moldable/evolving/malleable* job the number of processors assigned to a job are subject to changes throughout the execution. Another model for schedulers is to support different level of preemption, such as preemption and non-preemption. 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-preemption scheduling, the processors will be dedicated to the job throughout their execution after allocation. Preemption method may have great advantage in terms of system performance improvement, but it may incur extra overheads, such as the cost of memory due to migration and preemption [16]. Therefore, we consider non-preemptive scheduling process in our experiments.

Considering the above scheduler properties, we simulate thirteen different scheduling policies. All of these policies are described below.

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. Both LJF and SJF 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. On the other hand, both MinET and MaxET perform the same operation in decreasing and increasing order in terms of job's length, respectively [15]. FF is an additional technique to enhance the capability of the above five policies. FF policy finds a job in shared ordered queue list that can be fit on first idle resources such that the utilization of the resources is increased.

A backfilling technique [17], [18] makes resource reservations for jobs, and then it backfills the jobs only if next jobs (i.e., Short jobs) do 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 second one makes reservations for all of the jobs contained in the shared queue. The *aggressive* backfilling (known 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 its performance is superior to the conservative one [18].

In *K*-reserved based policy, a waiting request has a counter containing *K* number of times that it has been overtaken by subsequent requests [19]. The *K*-reserved based policy works similarly with the aggressive backfilling technique with the difference being that the *K*-reserved based policy backfills only *K* numbers of jobs, while the aggressive technique does not have any limitation in terms of the jobs that are backfilled. Window-*K* policy enhances the FCFS policy for a window of consecutive jobs. The window starts with the oldest waiting job and contains up to *K* jobs that have arrived successively [19]. As in [19], we set *K* to 5 for the *K*-reserved based policy and 10 for the Window-*K* policy.

B. Simulation setup

For the workloads simulation, we used a custom Java-based discrete event simulator. Java provides full advantage of Object-Oriented Programming (OOP) technique [20]. A typical object-oriented software system implementation is centered on dynamic creation, manipulation, storing, and releasing of objects [21], [22]. The Java environment setup also allows database connectivity, where we stored all the datasets. We simulated all of the policies several times and took to get accurate result for the experiments. We created an event-based setup, in which scheduler policies (Java programs) are invoked every 10 seconds. To approach the real setup, in each iteration the timestamps of the jobs belonging to the queue are updated according to the aforementioned interval of 10 seconds.

V. Analyzing and Comparing the Results

In this section, we analyze all of scheduling policies considered in this work. We evaluate their performance by measuring four metrics: **(a)** mean queue-time, **(b)** mean response-time, **(c)** mean slowdown, and **(d)** slowdown ratio. We must note that service providers are concerned with mean response-time and mean queue-time, while customers are concerned with mean slowdown and the slowdown ratio. 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 required nodes. Job's response-time represents the time elapsed after the arrival of the respective job until its execution [25]. Equation 1 and 2 can be used to calculate the mean response-time and queue-time of the entire workload.

$$Mean\ Queue\ Time = \frac{\sum Time\ (start_{time} - submit_{time})}{Total\ number\ of\ Jobs}, \quad (1)$$

$$Mean\ Response\ Time = \frac{\sum Time\ (end_{time} - submit_{time})}{Total\ number\ of\ Jobs}. \quad (2)$$

In the sequel, we give the definition of the rest metrics. *Mean slowdown* is the normalized time of each job (i.e., job completion time divided by job running time), and

slowdown ratio exhibits the normalized time of mean response-time [28] derived in Equation 3.

$$Slowdown\ Ratio = \frac{Time\ (mean\_response)}{Time\ (mean\_execution)}, \quad (3)$$

Time (mean_response) and *Time (mean_execution)* indicate the mean response-time and mean running time of the entire workload. For example, if the mean response-time of a set of jobs is 10 hours and their mean execution-time on nodes is 5 hours, then the slowdown ratio will be 2 hours. The slowdown ratio is important for measuring the performance of the scheduling policy for the entire workload [25].

Fig. 5 shows the results regarding the above performance metrics for the entire workload: **(a)** mean queue-time, **(b)** mean response-time, **(c)** queue time, and **(d)** response time for all jobs, **(e)** mean slowdown, and **(f)** slowdown ratio.

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 VI shows their classification, such as short and narrow (SN), long and narrow (LN), short and wide (SW), and long and wide (LW) jobs. The above four job observations are defined below with their results being shown in Fig. 6, Fig. 7, Fig. 8, and Fig. 9.

$$Job_{width} = 1\ CPU \quad AND \quad Job_{length} \leq 1\ Hour \quad (Ob.1)$$

$$Job_{width} = 1\ CPU \quad AND \quad Job_{length} > 1\ Hour \quad (Ob.2)$$

$$Job_{width} > 1\ CPU \quad AND \quad Job_{length} \leq 1\ Hour \quad (Ob.3)$$

$$Job_{width} > 1\ CPU \quad AND \quad Job_{length} > 1\ Hour \quad (Ob.4)$$

TABLE VI. OBSERVATION TABLE FOR ENTIRE WORKLOADS (JOB'S BREAKDOWNS FOR CORRELATION BETWEEN JOB'S WIDTH AND JOB'S LENGTH)

Job size		Short (S) <= 1 Hr	Long (L) > 1 Hr	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

A. Discussion

In conclusion, we explored the job characteristics before scheduling process. The important job characteristics are: **(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.

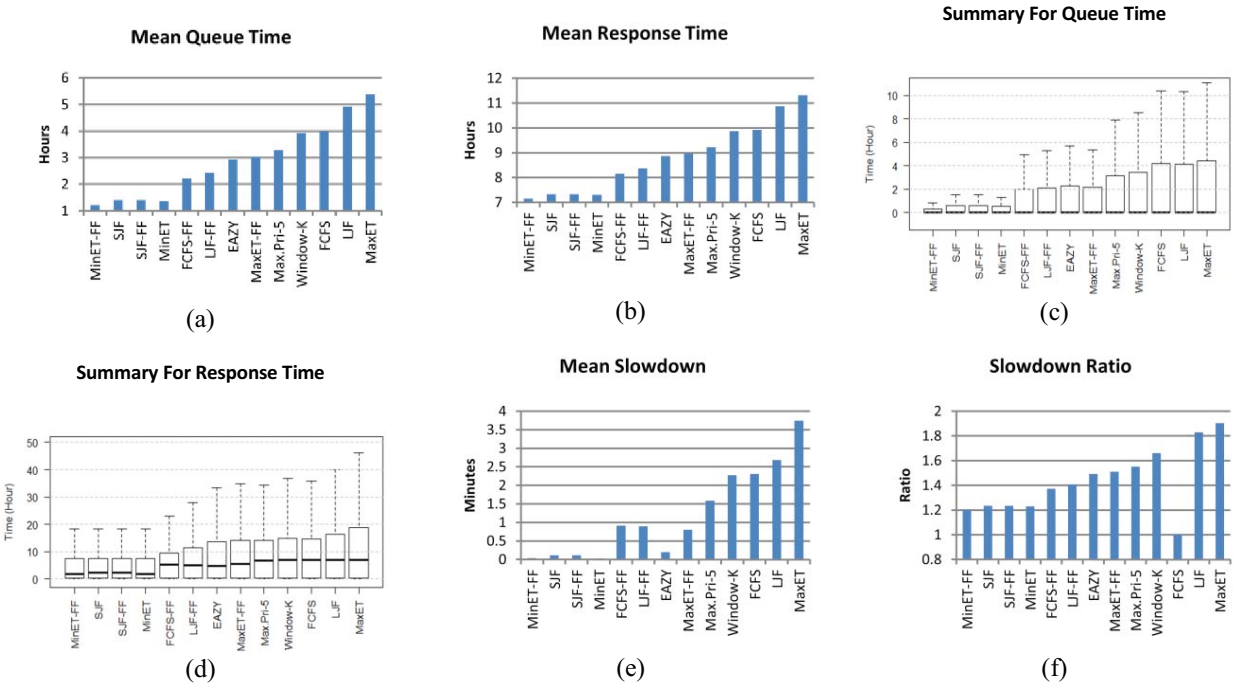
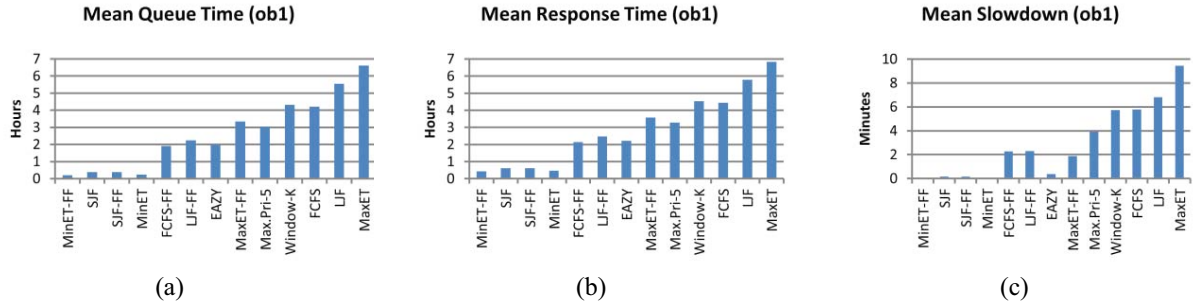
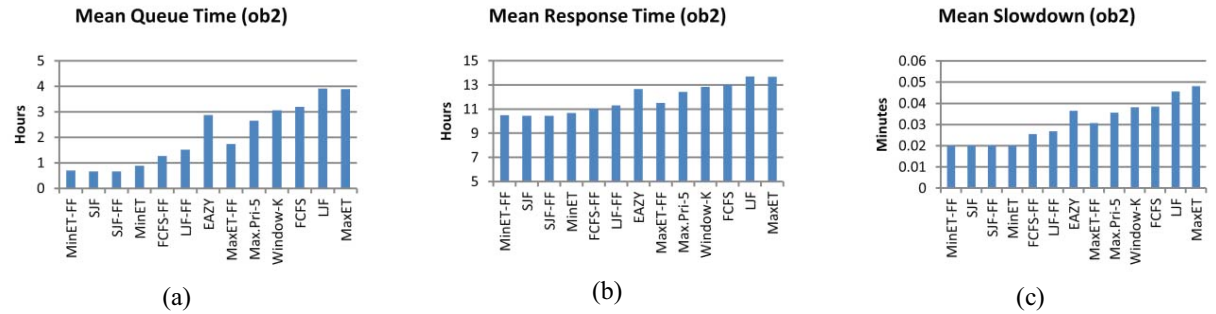


Fig. 5 Comparison results of all policies with overall workloads: **(a)** Mean queue-time, **(b)** Mean response-time, **(c)** Box-Plot for queue-time of all jobs, **(d)** Box-plot for response-time of all jobs, and **(f)** Mean slowdown.



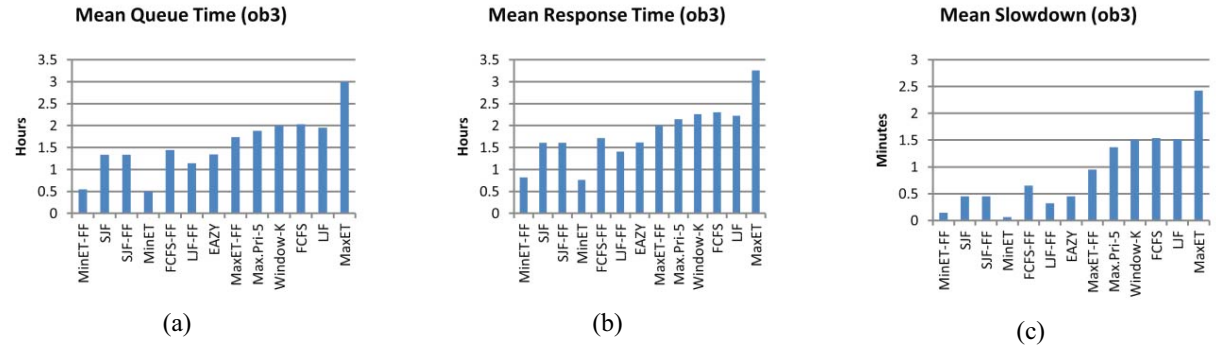
(a) (b) (c)

Fig. 6 Comparison results of all policies with overall workloads in first observation: **(a)** Mean queue-time, **(b)** Mean response-time, and **(c)** Mean slowdown



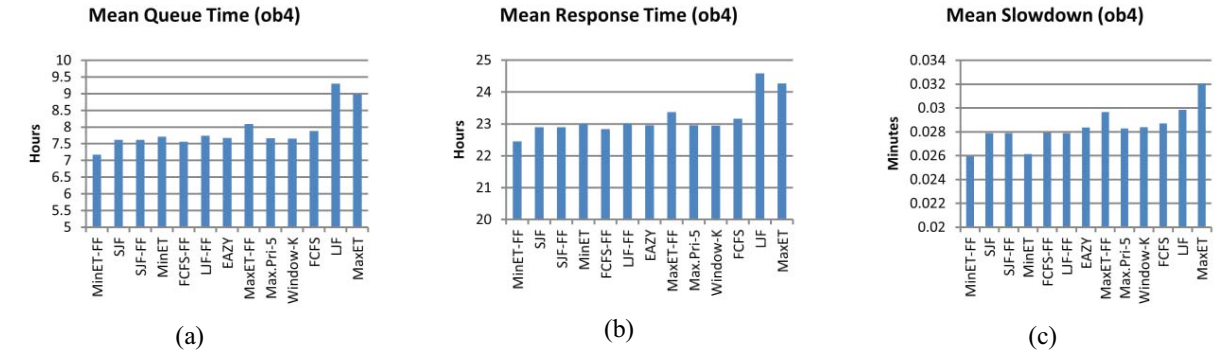
(a) (b) (c)

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



(a) (b) (c)

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



(a) (b) (c)

Fig. 9 Comparison results of all policies with overall workloads in fourth 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 computational 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.

The results of FCFS policy in all of the figures in the previous section are not satisfactory in terms of job queue time and response time metrics. A solution of the processors fragmentation problem 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), 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. 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 are explained below.

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.

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 major reason that SJF policy is superior to the rest policies in all the figures is due to the fact a 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, we would expect that MinET and MaxET produce almost the same 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.

In Fig. 6 and Fig. 7, we present the results for Narrow jobs. The number of jobs is 36% in Fig. 6 and 42% in Fig. 8 of the total jobs in the overall workload. In Fig. 6, the jobs belong to SN category, while in Fig. 7 the jobs belong to LN category. In both figures, the scheduling policies of Class-I dominate the policies belonging to other classes.

The results for Wide jobs (i.e., SW and LW) are shown in Fig. 8 and Fig. 9**,** and their job percentage is 10% and 11%, respectively. In Fig. 8, the scheduling policies of both Class-I and Class-II exhibit better results as compared to the policies in Class-III, while Fig. 9 shows that the results are almost the same for all of the scheduling policies.

In the above sections, we thoroughly discussed the characterization and analysis of the real data center workload and several job-scheduling policies. We investigated that most of the job scheduling policies are affected significantly by certain workload characteristics.

VI. CONCLUSION

To enhance the performance of the cloud data centers, job scheduling and resource allocation techniques are needed to maintain various levels of QoS. We studied a set of job scheduling policies, and characterized real data center workload for optimizing system's performance. A total of thirteen job scheduling policies were studied to analyze and compare the results. The set of job scheduling policies considered in this work includes **(a)** priority-based policies, **(b)** tuning policy, **(c)** backfilling techniques, and **(d)** window-based technique. We used four metrics (mean queue time, mean response time, mean slowdown, 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. The workload exhibited a wide range of job heterogeneity. We first characterized the workload and unveiled different job characteristics, such as that **(a)** the jobs of the system demand either single CPU or even number of CPUs for execution, and **(b)** 47% and 53% of jobs are executed within an hour and more than an hour, respectively. Moreover, we examined and analyzed the results of different job scheduling policies. Specifically, we observed that MinET and SJF when combined with FF technique exhibit better results as compared to the other policies.

Our analysis revealed that a single policy is not sufficient for resource management in parallel computational environments. Such environments need to implement dynamic and adaptive scheduling policies. In future work, we will investigate configuration-based policies to balance the scheduling policies for non-uniform workloads, as well as energy-ware policies.

ACKNOWLEDGMENTS

This work is funded in part by the grant of the National Natural Science Foundation of China (No. 61202417), the public technology service platform for Cloud computing inspection, detection and application standardization, the

implementation and technical service of security guard system for Cloud computing, the national development, and the reform commission PRC.

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 Transactions on Parallel and Distributed Systems, Vol. 22, Issue: 6 , 2011 , pp. 931 – 945.

[2] Xu, C. Z., Rao, J., and Bu, X. "URL: A unified reinforcement learning approach for autonomic cloud management". Journal of Parallel and Distributed Computing, 72(2), 2012, pp. 95-105.

[3] Wei T., Dongxu R., Zhiling L., Narayan D., "Adaptive Metric-Aware Job Scheduling for Production Supercomputers," 2012 41st International Conference on Parallel Processing Workshops, (ICPPW'12) pp. 107-115, 2012

[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, September 2009, pp. 13-18.

[5] Khan S.U., C. Ardil, "A Weighted Sum Technique for the Joint Optimization of Performance and Power Consumption in Data Centers," International Journal of Electrical, Computer, and Systems Engineering, vol. 3, no. 1, pp. 35-40, 2009.

[6] Khan S.U., Ahmad, I., "Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation," Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International , vol., no., pp.10 pp., 25-29 April 2006

[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, December 2009, pp. 410-417.

[8] Tracy D. Braun, *et. al.*, 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.* 61, 6 (June 2001), 810-837

[9] Arndt, O., et al., A comparative study of online scheduling algorithms for networks of workstations. Cluster Computing, 2000. 3(2): p. 95-112

[10] Baraglia R., *et al,*. "A multi-criteria job scheduling framework for large computing farms", Journal of Computer and System Sciences, ELSEVIER, 79, pp. 230–244 , 2013.

[11] Feitelson D.G., Tsafrir D., and Krakov D., "Experience with the Parallel Workloads Archive". *Technical Report 2012-6*, School of Computer Science and Engineering, The Hebrew University of Jerusalem, April 2012.

[12] Feitelson. D.G., "Workload modeling for performance evaluation". Workshop on Job Scheduling Strategies for Parallel Processing , Lecture Notes in Computer Science, 2459:114–141, 2002.

[13] Steven H., David S., and Timothy O', Donnell. "Analysis of the Early Workload on the Cornell Theory Center IBM Sp2", ACM SIGMETRICS Conference on Measurement and Modeling of Computer System, May 1996. Poster.

[14] Srinivasan S., Kettimuthu R., Subramani V., and Sadayappan P., "Selective Reservation Strategies for Backfill Job Scheduling", Workshop on Job Scheduling Strategies for Parallel Processing , Lecture Notes in Computer Science, pp. 55-71, July 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", Journal of Parallel and Distributed Computing, Vol. 59, Issue 2, Nov. 1999, Pages 107–131

[16] Feitelson D.G, et al.,"Theory and Practice in Parallel Job Scheduling". In Proceedings of the Job Scheduling Strategies for Parallel Processing (IPPS '97),

[17] Litka D.A., "The ANLIIBM SP Scheduling System," Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Springer, vol. 945, 1995, pp. 295-303,

[18] Mu'alem A.W., Feitelson D.G.. "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling". IEEE Transactions on Parallel and Distributed Systems, Vol. 12(6), 2001, pp. 529-543.

[19] Ababneh I. and Bani-Mohammad S., "A new window-based job scheduling scheme for 2D mesh multicomputers". ELSEVIER, Simulation Modelling Practice and Theory, 19(1):482–493, 2011.

[20] Horton I., "Ivor Horton's Beginning Java 2, JDK 5 Edition", Text Book, December 2004

[21] Saleh K., "Object model in Java: elements and application", Information and Software Technology ,Vol. 41, Issue 4, 15 March 1999, pp.235–241

[22] Chandio A.A., Zhu D., Sodhro A.H.. "Integration of Inter-Connectivity of Information System (i3) using Web Services". In Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS), Lecture Notes in Engineering and Computer Science, Vol. 2195, pp. 651-655 (2012)

[23] 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, Lect. Notes Comput. Sci. vol. 1659, 1999, pp. 66-89

[24] Parallel Workload Archive. http://www.cs.huji.ac.il/labs/parallel/workload/

[25] 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, 1998, pp 25-46.

[26] Lublin, U., Feitelson, D. "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs." J. Parallel and Distributed Computing, Vol. 63(11), 2003, pp.1105-1122.

[27] Chiang S.-H. and Vernon M. K.. "Characteristics of a large shared memory production workload", Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in ComputerScience, 2221: 159–187, 2001

[28] Downey A.B., "A parallel workload model and its implications for processor allocation", 6th International Symposium of High Performance Distributed Computing, 1997, pp-112-123.

[29] Wang, L., Khan S.U, and Dayal J., "Thermal aware workload placement with task-temperature profiles in a data center". The Journal of Supercomputing, 61(3): p. 780-803, 2012

[30] 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), 2013, (to appear).