Comput Syst Sci & Eng (2019) 2: 79–90
© 2019 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

A load balanced task scheduling heuristic for large-scale computing systems

Sardar Khaliq uz Zaman[1], **Tahir Maqsood**[1], **Mazhar Ali**[1], **Kashif Bilal**[1], **Sajjad A. Madani**[1], **Atta ur Rehman Khan**[2]*

[1] *Department of Computer Science, COMSATS University, Pakistan*
[2] *Faculty of Computing and Information Technology, Sohar University, Oman.*

Optimal task allocation in Large-Scale Computing Systems (LSCSs) that endeavors to balance the load across limited computing resources is considered an NP-hard problem. MinMin algorithm is one of the most widely used heuristic for scheduling tasks on limited computing resources. The MinMin minimizes makespan compared to other algorithms, such as Heterogeneous Earliest Finish Time (HEFT), duplication based algorithms, and clustering algorithms. However, MinMin results in unbalanced utilization of resources especially when majority of tasks have lower computational requirements. In this work we consider a computational model where each machine has certain bounded capacity to execute a predefined number of tasks simultaneously. Based on aforementioned model, a task scheduling heuristic Extended High to Low Load (ExH2LL) is proposed that attempts to balance the workload across the available computing resources while improving the resource utilization and reducing the makespan. ExH2LL dynamically identifies task-to-machine assignment considering the existing load on all machines. We compare ExH2LL with MinMin, H2LL, Improved MinMin Task Scheduling (IMMTS), Load Balanced MaxMin (LBM), and M-Level Suffrage-Based Scheduling Algorithm (MSSA). Simulation results show that ExH2LL outperforms the compared heuristics with respect to makespan and resource utilization. Moreover, we formally model and verify the working of ExH2LL using High Level Petri Nets, Satisfiability Modulo Theories Library, and Z3 Solver.

Keywords: Large-Scale Computing Systems, Load Balancing, List-based Task Scheduling, Makespan, Resource Utilization

1. INTRODUCTION

Large-scale computing encompasses variety of concepts and systems, such as supercomputers, cluster computing, grid computing, cloud computing, and multicore systems. Large-Scale Computing Systems (LSCSs) support wide range of services, such as file storage, computational services, and resource provisioning [1-2]. Some of the application areas where LSCSs have been extensively utilized include: processing of huge data produced in the high-energy nuclear physics experiments, astronomical computations, bioinformatics, geophysics, 3D Modeling, neural sciences, e-health, medical image processing, large-scale recommender systems [3-5]. The LSCSs architectures include simple and low-power multi-core systems, application-specific processors, and heterogeneous computing systems.

Even with the disparity of system architectures, all variants of LSCSs face issues concerning workload scalability, application throughput, reliability, energy consumption, resource utilization and availability, data storage, and heat dissipation [6]. Efficient task scheduling can help in addressing majority of the aforementioned issues [6]. However, optimal task scheduling on limited computing resources is an NP-hard problem [7]. Moreover, heterogeneity of computing resources and tasks introduce interesting tradeoffs among the scheduling performance metrics [8]. Parameters that are used to evaluate the performance of task scheduling heuristics include, but are not limited to makespan,

*Corresponding Author: dr@attaurrehman.com energy consumption, and resource utilization [9]. Makespan optimization is considered as one of the primary objectives that indicates the overall efficiency of task scheduling heuristics [6]. Further, majority of the LSCSs are housed in large data centers [10]. However, data centers rarely operate at their maximum rated load, i.e., 10% to 50% of the servers are fully utilized at any given time [10]. Therefore, improving resource utilization is another important parameter that needs to be considered by scheduling heuristics. Similarly, due to increased energy costs and demand, efficient utilization of available electrical power is of paramount importance [11]. For instance, according to a recent study, an idle server consumes nearly 60-70% of the power consumed while operating at full capacity [12]. Therefore, it is desirable to effectively utilize the idle servers to enhance the resource utilization and reduce makespan. However, optimization of the aforesaid performance parameters, i.e., makespan and resource utilization, leads to an increase in power consumption and higher task missing rates which is not acceptable in large scale computing systems such as clouds [10, 13]. In this work, we have conducted an in-depth study to analyze the tradeoffs among the multiple optimization criteria that include makespan optimization, resource utilization, and energy consumption with different task scheduling heuristics.

Being an NP-hard problem, task scheduling has attracted significant attention in the field of large scale computing. The efficiency of task scheduling algorithm in LSCSs, directly affects the overall makespan and implicitly affects power consumption and resource utilization of the system [6, 14]. In literature, majority of the research concerning performance parameters discussed earlier, is reported for batch mode approaches based on the MinMin algorithm because of its well-founded performance, minimum makespan, and simplicity [15, 16]. The MinMin heuristic schedules tasks having minimum expected time to complete on the fastest machine. However, it results in uneven load distribution and poor resource utilization by scheduling more tasks on certain faster machines while keeping others idle or lightly loaded. Uneven load distribution among computing nodes leads to poor resource utilization as well as energy wastage [15, 17]. The High to Low Load (H2LL) heuristic is an extension of MinMin that promises to generate lower makespan [18]. However, considering the proposed system model, wherein each machine has a specified computing capacity to accommodate limited number of tasks, H2LL is unable to schedule some of the tasks because of infeasible task to machine mapping due to limited available computing capacity or memory of the selected machine. To this end, we propose an extension of H2LL, named as ExH2LL, which tackles the problem by dynamically selecting the appropriate alternate machine with lowest utilization for the assignment of incoming task.

1.1 Motivation

The problem tackled in this study is twofold. First, we detail the uneven load distribution and higher makespan when smaller tasks are in majority, concerning MinMin algorithm. Second, the task missing phenomenon is highlighted with respect to the H2LL algorithm.

The MinMin algorithm schedules the task based on Estimated

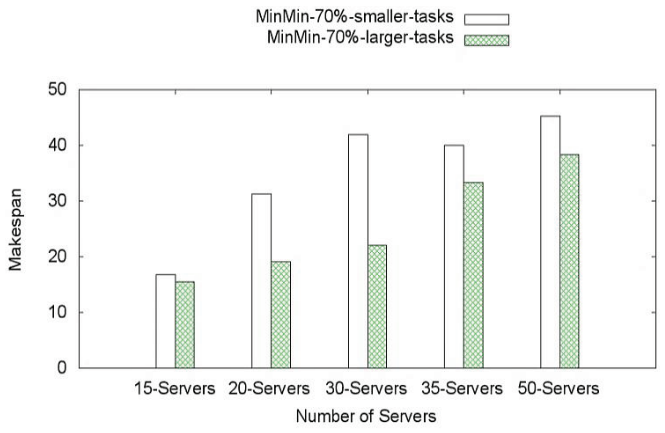


Figure 1 Makespan of MinMin (Task distribution wise).

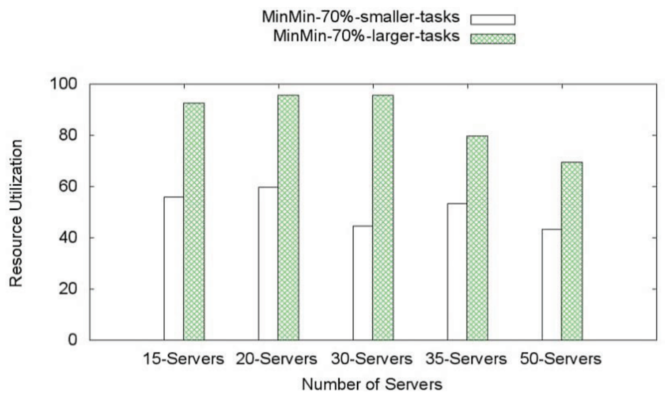


Figure 2 Resource Utilization of MinMin (Task distribution wise).

Time to Completion (ETC). The ETC refers to the execution time of a task plus the ready time. The MinMin algorithm first schedules the smaller tasks and then the larger tasks are scheduled. The task with overall minimum ETC is scheduled on the fastest machine [33]. The sizes of task and their distribution are given higher consideration. Therefore, the task distribution comprises three cases: **(a)** majority of the tasks are large, **(b)** majority of the tasks are small, and **(c)** random proportion of smaller and larger tasks. Here larger tasks refer to the tasks having higher computational requirements in terms of Millions of Instructions (MIs) and smaller tasks refer to the tasks with lower MIs requirements. In the case where the smaller tasks are more, MinMin produces comparatively higher makespan and results in lower resource utilization than the case where the majority of tasks are large [34], as shown in figure 1, 2. The H2LL endeavors to solve the problem associated with MinMin and optimizes the makespan. H2LL achieves this by introducing a two-phase heuristic described in Section 5, but at the cost of task misses. The H2LL algorithm can miss the tasks in case where we have machines with limited computing capacity and memory, i.e., there is a limit on the number of tasks that can be scheduled on each machine simultaneously within the specific time interval.

Particularly, after the task and machine selection, a state may occur due to the capacity constraint that prevents the selected task to be scheduled at the selected machine that reported the minimum completion time. In such cases, a task reassignment strategy is required that will not only map the tasks considering the makespan optimization but also attempts to evenly balance the resource utilization. Therefore, in this work, we propose ExH2LL task scheduling heuristic that jointly optimizes the makespan and resource utilization.

1.2 Contribution

The main contributions of this paper are summarized as follows: We propose an improved heuristic named ExH2LL that optimizes the makespan, improves the resource utilization, and eliminates the task misses by evenly distributing the workload among the available machines. The proposed ExH2LL algorithm identifies the resource or machine with least utilization rate for the assignment of incoming task. Moreover, we perform the formal modeling and verification of the ExH2LL by using High Level Petri Nets (HLPN), Satisfiability Modulo Theories Library (SMT-Lib), and Z3 solver.

The remainder of the paper is organized as follows. We summarize the existing work in Section 2. Section 3 details the system and application model, Expected Time to Compute (ETC) model, and power estimation model used in the proposed work. The proposed heuristic (ExH2LL) is introduced in Section 4 followed by formal analysis and verification in Section 5. Section 6 provides the simulation results and the comparisons of ExH2LL with the state-of-the-art heuristics. Finally, Section 7 concludes the paper with some directions for future work.

2. EXISTING WORK

During the past decade, research on the task scheduling has focused on finding the best, among the available suboptimal solutions. The solutions are considered suboptimal, because they always have some limitations in terms of performance, energy consumption, and resource utilization [19]. The task scheduling heuristics are divided into two major classes namely heuristic based and meta heuristic based or Guided Random Search Based (GRSB) approaches. The heuristic based and GRSB approaches are also referred as deterministic and non-deterministic methods, respectively [15]. The heuristic-based algorithms provide the solutions with polynomial time complexity [6, 19]. The solutions provided by the heuristics-based algorithms are approximate, but a sub set of these may result in better solutions with low complexity and minimum makespan. The GRSB algorithms also generate approximate and good solutions. However, the quality of those solutions may be enhanced by performing more iterations at the expense of higher computational cost [20]. The Heuristic-based scheduling is further divided into three subcategories: **(a)** clustering based, **(b)** duplication based, and **(c)** list-based scheduling [6, 19]. The clustering-based heuristics form clusters of tasks which are to be allocated to multiple processors. Some examples in this category include Clustering for Heterogeneous Processors (CHP), Clustering And Scheduling System (CASS), and Triplet [21]. The clustering heuristics have lower communication overhead, but at the cost of higher makespan and increased complexity. Moreover, each cluster needs dedicated processor for its tasks' execution. Furthermore, these schemes form clusters of the similar tasks which, makes it suitable for homogeneous systems. The duplication heuristics offer the shorter makespan and communication overhead, but they have higher time complexity and execution of the task is duplicated [22]. Some of the duplication algorithms are Heterogeneous Limited Duplication (HLD), Heterogeneous Critical Parents with Fast Duplicator (HCPFD), and Heterogeneous Earliest Finish-time with Duplication (HEFD) [22]. To decrease the communication cost of a large number of dependent tasks, the dependent tasks are recognized and executed redundantly. The aforesaid duplications provide the fault tolerance and reliability, but results in significant wasting of processing capacity and energy [22]. The List scheduling heuristics produce the schedules that do not compromise the makespan and possess lower complexity than GRSB and clustering-based heuristics [19]. Therefore, these heuristics are widely used for scheduling in LSCSs. The list-based heuristics are further categorized into batch mode and dependency mode approaches. The batch mode approaches organize the tasks according to their execution time, while the dependency mode schemes, such as Heterogeneous Earliest Finish Time (HEFT) [23], Predict Earliest Finish Time (PEFT) [6], and Mapping Heuristic (MH) [23] sort the tasks according to the length of their critical path. The critical path is defined as the maximum, sum of communication time and average execution time, beginning from that task to the exit task [19]. The feature which makes dependency mode heuristics more feasible is that the larger tasks are completed prior to the smaller ones, reducing the overall execution time. However, the complexity of dependency mode approaches is higher than the batch mode heuristics, as it must compute the critical path of all the tasks. Although HEFT and PEFT both are highly valued among heuristic algorithms for the scheduling problem, yet in complex scenarios these schemes cannot succeed to produce the optimal scheduling. The reason is that both are essentially the greedy algorithms [20]. Therefore, in literature most of the research is reported for batch mode approaches and more specifically the variants of MinMin algorithm because of its well-founded performance, minimum makespan, and simplicity.

The conventional batch mode algorithms such as, First in First out (FiFo) and Round Robin (RR) suffer from convoy effect and increased complexity while calculating suitable time quantum [24]. Some other schemes that are based on batch mode are: MinMin, MaxMin, and Suffrage Heuristics [7, 18], Energy Aware Fast Scheduling Heuristics (EAFSH) [15], energy aware MinMin algorithm [16], QoS guided MinMin [25], Segmented MinMin [26], and Opportunistic Load Balancing (OLB). The OLB does not use the execution or completion time, of the tasks to map them to available resources. The OLB maps the tasks randomly. The QoS Guided MinMin focuses to provide quality of service by specifying network parameters like bandwidth for each task and then assign them to machines based on the QoS parameter. Some studies regarding load balancing techniques have also been reported like Load Balanced MinMin (LBMM) [27], Hierarchical Load Balancing Algorithm (HLBA) [24], Resource Aware Scheduling Algorithm (RASA) [28], and Dynamic Load Balancing Algorithm (DLBA) [29]. However, these approaches also have various shortcomings. For instance, LBMM does not consider high machine heterogeneity. In RASA, task arrival time, execution cost on each resource, and communication costs have not been addressed. In DLBA threshold value for load balancing level is fixed and static. The threshold value is made dynamic in further extension named HLBA, but the length of execution time of incoming tasks has not been considered in both DLBA and HLBA, which results in uneven distribution of tasks, as done in the MinMin. Moreover, some recent efforts promising the minimum makespan include; Improved MinMin Task Scheduling (IMMTS) [30], H2LL [18], Load Balanced MaxMin (LBM) [31] and M-Level Suffrage-Based Scheduling Algorithm (MSSA) [32]. IMMTS defines a statistical selection criterion for each task. Then the machine, on which selected-task produces the minimum completion time, is nominated for task-to-machine mapping in the next phase. This scheme involves the inefficient resource utilization and higher makespan in some cases. The higher makespan and inefficient resource utilization in IMMTS is caused due to its statistical task selection criterion, which either executes the smaller tasks or larger tasks first, thus makes the heuristic unfeasible for heterogeneous systems.

The working of classical MinMin algorithm involves two phases. In the first phase it finds the fastest machine m among available resources which gives the minimum ETC. In the second phase the task t having minimum ETC on the machine m, is identified and scheduled on machine m. Some of the recent variants of MinMin, such as H2LL [18] and LBMM heuristics claim to optimize the makespan, but sometimes they miss several tasks needed to be executed, as detailed in section 5.1. The H2LL, being a two-phase heuristic, first finds the *best-score* (lowest makespan) of the system. Then, it finds the task $t_k$with maximum completion time against the machine $m_k$reporting the lowest makespan in each iteration. In the next phase, H2LL finds the minimum completion time (*new-score*) of previously selected task t_k against all the machines. Based on the comparisons of *best-score* and *new-score*, H2LL assigns the incoming tasks to the machine reporting lowest score. At this point, H2LL fails to assign some of the tasks in case the machine selected for task placement, having lowest score, does not have the required capacity to accommodate the task.

In this work we propose an extension of the H2LL heuristic that not only optimizes the makespan but also reduces the task missing rate while improving the resource utilization. We compare the ExH2LL heuristic with MinMin, IMMTS, H2LL, LBM, and MSSA heuristics. The ExH2LL dynamically identifies the resource or machine with least utilization rate for the assignment of incoming task. Moreover, our results show that the ExH2LL outperformed the compared heuristics with respect to makespan and resource utilization, but at the cost of slightly increased power consumption.

3. SYSTEM MODELS

3.1 Computational Model

The system model used in this paper resembles the classical bin packing model presented in [35], wherein the tasks are scheduled on the machines as they arrive. T represents the set of tasks $\{t_1, t_2, \ldots\ldots, t_n\}$. Each task $t_i \in T$ has computational requirements req_i in MIs. Similarly, M represents the set of machines $\{m_1, m_2, \ldots\ldots, m_x\}$ and processing speed of each machine $m_j \in M$ is defined by s_j. In this work we consider a model where each machine has a certain capacity cap_j to accommodate a predefined number of tasks. The capacity cap_j of a machine defines the maximum number of tasks that can be processed by the given machine simultaneously [36]. This is true for the computational platforms, such as cloud computing and multicore processor architectures. For instance, in cloud environment each physical machine can host several virtual machines. In turn, each virtual machine can execute the assigned tasks independently. Similarly, multicore architectures comprise of multiple processing cores that can execute multiple tasks in parallel. The number of tasks supported by each physical machine depends on the number of virtual machines created or the number of available processing cores. We assume that processing speed s_j assigned to each virtual machine or processing core of a given physical machine is same.

3.2 Expected Time to Compute Model

We use the ETC model defined by [37], where in ETC matrix of size $T \times M$ is computed, as shown in Figure 3. Each position *ETC[i][j]* in the ETC matrix specifies the expected time to compute a certain task t_i on machine m_j. The ETC matrix is generally assumed to be predetermined [15]. However, mostly in the previous works [38-43] the degree of the heterogeneity to produce the ETC matrix, exploits two methods.

- First one is the range of execution times of tasks
- Second is their coefficient-of-variance (CoV)

Nevertheless, these methods lack to represent the variation in heterogeneity completely [39]. In the proposed model with heterogeneous machines and tasks in the workflow, ETC for a task is calculated by first, adding ready time of the task, represented by $ready_i$, to the computational requirements req_i of t_i and then dividing the summed workload by the processing speed of resource m_j, as described in [37]. ETC matrix is calculated using the Equation 1.

$$\sum_{i=1}^{t}\sum_{j=1}^{m} ETCij = \frac{req_i + ready_i}{s_j} \tag{1}$$

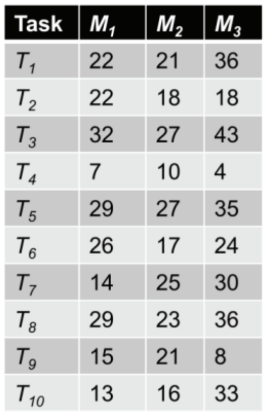
Moreover, the problem tackled in this study is to assign the resources to reduce the makespan in heterogeneous computing environment. The makespan of a schedule S is the time when all the tasks are executed and can be calculated using Equation 2.

$$makespan = C_{max}(S) \tag{2}$$

Where schedule S is a function, which maps a task to the machine that executes it and $C_{max}(S)$ indicates the maximum value among completion times of all machines [44]. More elaborately, it is the time when the last task T_{exit} in the system is completed [44].

3.3 Power Model

In most of the data centers, energy consumption by the processing machines is calculated by investigating each of the IT



Task	M_1	M_2	M_3
T_1	22	21	36
T_2	22	18	18
T_3	32	27	43
T_4	7	10	4
T_5	29	27	35
T_6	26	17	24
T_7	14	25	30
T_8	29	23	36
T_9	15	21	8
T_{10}	13	16	33

Figure 3 An example expected time to compute (ETC) matrix.

equipment‘s individual contribution, i.e., CPU (80%), storage media (10%), and network devices (10%) [45-46]. The CPU devours the major portion of electricity as compared to other resources in the system. Moreover, idle CPUs also consume 70% of, the total power used when fully operational, leading to the wastage of processing capacity. Therefore, in the present study we propose a strategy, to use the wasted electrical power of computing machines, which will reduce the makespan, as well as improve the resource utilization. The aforesaid reduction and improvement is achieved by shifting the work load to idle or lightly loaded machines. The power model proposed in [10] has been implemented and illustrated in Equation 3.

$$P(u) = k \times P_m + (1-k) \times P_m \times u \tag{3}$$

Here maximum power consumed by a machine at full utilization is denoted by "P_m". The fraction of power consumed by an idle machine is denoted by "k". The fraction of CPU utilization of a particular machine is denoted by "u''. In this implementation "P_m" is fixed to 250 watts, since many modern servers use this value as mentioned in recent studies [10, 18]. Moreover, according to Standard Performance Evaluation Corporation (SPEC) power benchmark 2010 report, the average power consumption by fully functional machines was reported to be 259 Watts [47].

4. PROPOSED HEURISTIC

To tackle the scheduling problem discussed above, we propose a task scheduling heuristic named Extended High to Low Load (ExH2LL). The ExH2LL is a two-phase algorithm that attempts to optimize the makespan, improve the resource utilization, and eliminate task misses by evenly distributing the workload among the available computing resources.

4.1 ExH2LL Operations

In the first phase, the ExH2LL algorithm finds the machine m_j that has the highest makespan (line-1 in Table 1). The highest maekspan is determined after mapping all tasks on each machine using MinMin heuristic. The MinMin heuristic schedules the task based on ETC. The MinMin heuristic first schedules the smaller tasks and then the larger. The task with overall minimum ETC is scheduled on the fastest machine and the slowest machine delivers the reference makespan for ExH2LL algorithm. In the second phase of ExH2LL, the task t_k having maximum completion time at the machine m_j (lines 2-4) is selected. In next step, the ExH2LL finds the minimum completion time *minCT* of task t_k on each of the machines and the machine m_k that produces the *minCT* (line-5). The working of H2LL and ExH2LL is similar to a certain extent that, in the case of H2LL only the *minCT* is compared with makespan. Further, if the aforesaid condition is not satisfied, then the mechanism to select the alternate resource is not specified in the H2LL algorithm which results in task misses. However, as proposed in ExH2LL, along the makespan comparison, the capacity of corresponding machine is also verified as per task requirements. More elaborately, if *minCT* is less than the makespan, and machine m_k has the capacity to accommodate t_k then the task t_k is scheduled on resource m_k (lines 6-7). But, if *minCT* is greater than the *makespan*, and the machine m_j that produced the initial makespan can accommodate t_k, then the task t_k is scheduled on the resource m_j (lines 8-9).

Table 1 Working of ExH2LL heuristic.

```
Algorithm 1: ExH2LL heuristic
Inputs: a valid schedule from phase 1, set of machines M, set of tasks $T$
1. $m_j$ = machine reporting makespan during 1^{st} phase
2. do
3. for each $m_j \in M$
4. Find the task $t_k$ with maximum CT on machine $m_j$,
5. Find the $minCT$ of task $t_k$ and the machine $m_k$
6. if $minCT < makespan$ AND $m_k$ can accommodate $t_k$
7. Assign $t_k$ to machine $m_k$
8. Else if $minCT > makespan$ AND $m_j$ can accommodate $t_k$
9. Assign $t_k$ to $m_j$
10. Else find the resource utilization rate $RU\ \forall m$
11. Find the resource $m_l$ with minimum $RU$ having capacity to accommodate $t_k$
12. Assign $t_k$ to resource $m_l$
13. Delete $t_k$ from $T$
14. Update machines used and remaining capacity
15. End for
16. while $(T \neq \varphi)$
```

However, if none of the aforementioned conditions are satisfied then a mechanism to select the alternate resource is also proposed in ExH2LL. The proposed mechanism dynamically identifies the resource or machine with least utilization rate having required capacity for the assignment of task t_k(lines 10-12). Conclusively, two distinguishing features of ExH2LL play critical role to accommodate all the tasks and makespan reduction are: **(a)** task to machine mapping and **(b)** resource identification for load balancing. Task to machine mapping, verifies that which machine best fits the particular task, i.e., the machine that reported makespan in first phase or the machine having *minCT*. Moreover, it is also ensured that the selected machine has the required capacity to accommodate the selected task. In case of failure when there is no suitable machine found according to the above criteria, a mechanism is proposed for alternate resource identification for load balancing. The resource identification mechanism dynamically determines the current resource utilization rate of each machine in the system and enables ExH2LL heuristic to evenly distribute the load among the available computing resources. The resource utilization rate is calculated using the Equation 4. Moreover, the provided solution also ensures that each task in the system is assigned to a machine, thus preventing task misses. Finally, once a resource has been allocated to a task, then the mapped task is deleted from the task list and capacities of machines are updated. The process is repeated until all the tasks are mapped (lines 13-15).

$$Ru_m = \frac{Ex_m}{Cap_m} \times 100 \tag{4}$$

Where Ru_m denotes the resource utilization and Cap_m denotes the total capacity of a machine m, respectively. Further, Ex_m denotes the total execution time of machine m, which can be calculated by adding the computational requirements req_i of all the tasks (starting from first to last task) mapped on that machine, as shown in Equation 5. In Equation 5, n represents the total number of tasks scheduled on machine.

$$Ex_m = \sum_{i=0}^{n} req_i \tag{5}$$

5. FORMAL VERIFICATION AND ANALYSIS OF EXH2LL

We provide the basic introduction to HLPN, SMT-Lib, and Z3 solver for a clear understanding of the reader, as follows.

5.1 High Level Petri Nets

In wide variety of systems, the Petri Nets are extensively applied for their mathematical and graphical modeling [48]. In this wok HLPN, is used for the formal verification of ExH2LL heuristic.

5.2 SMT-Lib and Z3 Solver

To check the satisfiability of formula over theories under consideration, SMT is used [49] which offers a collective input platform and benchmarking framework. Being developed at a Microsoft Research, SMT-Lib with Z3 solver is used to prove the theorems. To verify whether the set of formulas are satisfiable in the built-in theories of SMT-Lib, Z3 solver being an automated satisfiability checker is used. The correctness of the system is checked during the verification process. In this work, bounded model checking is used to verify the ExH2LL heuristic. The HLPN model regarding ExH2LL heuristic is shown in Figure 4. The development of petri nets model involves identification of data types and places, and the mappings of data types to places. Table 2 and 3 show the Data types and their mappings, respectively. The rectangular black boxes show the transitions and belong to set T, whereas the circles represent the places and belong to set P in HLPN model. The working of proposed model is discussed in Section 5. In this section we define formulas to map on transitions. Initially we place unassigned tasks, available machines, execution times, and ready times for machines at a place mt for model checking purposes. Following Formula maps to the transition task-machine.

$$\begin{aligned} R(task\_machine) = {} & \forall x_1 \in X_1, \forall x_2 \in X_2 | \\ & x_2 1 = x_1 1 \wedge x_2 2 = x_1 2 \wedge x_2 3 = x_1 3 \wedge x_2 4 = x_1 4 \wedge \\ & X_2' = X_2 \cup \{x_2 1, x_2 2, x_2 3, x_2 4\} \end{aligned} \tag{6}$$

Table 2 Data types for HLPN model.

Data Type	Description
PmList	A list containing the ids of the machines in the system.
PtList	A list containing LID of the tasks to be executed in the system.
PetList	A list containing execution times of the tasks.
PrList	A list containing ready time of the machines.
ms	A number representing makespan of the system.
m_ms	The id of the machine reporting the ms.
ct	An array containing completion times of the tasks against machines.
tmCT	The task with maximum completion time.
PreqList	A list storing computational requirements of the tasks.
PccList	A list storing computational capacity of the machines.
m_k	The id of the machine producing minimum completion time.
m_j	The id of the machine producing m$_k$.
mct	The minimum completion time of a task.

Table 3 Places and mappings used in HLPN model.

Place	Mapping
$\varphi\,(Node)$	P*(PmList×PtList×PetList×PrList× ct×ms× tmCT×m$_j$×m$_k$× mct ×PreqList×PccList)*
$\phi\,(mt)$	P*(PmList×PtList×PetList×PrList × m_ms×PreqList× PccList)*

The MinMin computes the completion time for the whole system through the reporting system. The following equation is mapped at the transition CT.

$$\begin{aligned} R(CT) = {} & \forall x_3 \in X_3, \forall x_7 \in X_7 | \\ & x_3 4 = reportingserver() \wedge \\ & x_3 5 = cal\_exec\_time(x_7 3, x_7 4) \wedge \\ & X_3' = X_3 \cup \{x_3 4, x_3 5\} \end{aligned} \tag{7}$$

The maximum completion time of any task is selected as the makespan and the task and machine producing makespan. The

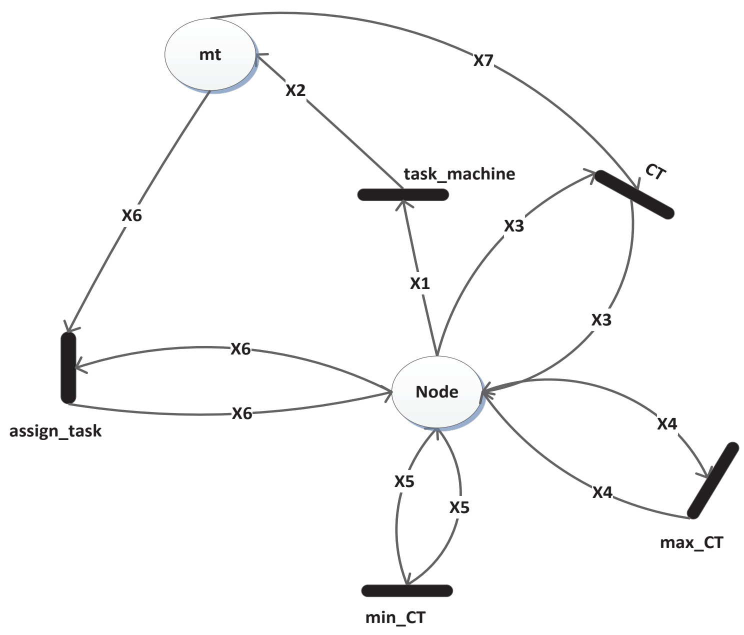


Figure 4 HLPN model for ExH2LL.

following transition and equation represents the aforesaid process.

$$R(\max\_CT) = \forall x_4 \in X_4 |$$
$$x_46 = \max CT(x_45) \wedge x_47 = task\_\max CT(x_45) \wedge x_48 = mac\_\max CT(x_45) \wedge \quad (8)$$
$$X_4' = X_4 \cup \{x_46, x_47, x_48\} \quad (9)$$

Afterwards, the machine that produces minimum completion time for the task selected in the previous rule. The rule is mapped at the transition *minCT*.

$$R(\min\_CT) = \forall x_5 \in X_5 |$$
$$x_59 = mac\_\min CT(x_57) \wedge x_510 = \min CT(x_57) \wedge$$
$$X_5' = X_5 \cup \{x_59, x_510\} \quad (10)$$

Finally, the tasks are assigned to the machines according to the criteria detailed in the proposed methodology.

$$R(assign\_task) = \forall x_6 \in X_6, \forall x_8 \in X_8 |$$
$$x_610 < x_66 \&\& req(x_86) < comp(x_87) \wedge$$
$$x_69 \leftarrow x_67 \wedge$$
$$X_6' = X_6 \cup \{x_69, x_67\}$$
$$x_610 > x_66 \&\& req(x_86) < comp(x_87) \wedge$$
$$x_68 \leftarrow x_67 \wedge$$
$$X_6' = X_6 \cup \{x_68, x_67\} \quad (11)$$

5.3 Verification Property

We have verified the proposed heuristic to ensure that it works in accordance with the specifications and generate the correct results. The verified property is described as following:

- The proposed model correctly calculates the makespan and identifies the machine producing the makespan.
- The proposed methodology assigns the tasks according to the specifications detailed in the proposed methodology.

The model of proposed heuristic given above has been translated to SMT-Lib and verified with the help of Z3 solver. The Z3 solver indicated that the proposed model is workable and executes itself in line with the specified properties. Z3 solver took 0.028 seconds to execute the working of proposed heuristic.

6. EXPERIMENTAL EVALUATION

To evaluate and compare the ExH2LL with MinMin, IMMTS, H2LL, LBM, and MSSA algorithms, a series of simulation experiments were performed. Similar to the settings presented in [12, 18, 50], the number of tasks per *WF* varies from 100 to 500 and the numbers of instructions for each task are selected randomly from 100 to 5000 million instructions to create heterogeneous WFs. Moreover, the number of hosted servers ranges from 15 to 50 with different computational capacities. The aforementioned settings provide wide range cases to extensively evaluate and compare performance of the scheduling algorithms. Furthermore, WFs were randomly generated according to three different distributions, i.e., uniform random distribution, 70% smaller tasks, and 70% larger tasks. The details of simulation parameters are summarized in Table 4.

Simulations have been conducted with different DAG based workflows for each of the implemented case. For each case, average readings have been taken over multiple DAGs to extend the validity of simulation scenarios. For each scenario based on server count, we have used three types of task distribution as shown in Table 4. The aforesaid task distributions are based on our problem formulation which states that the MinMin algorithm performs well when larger tasks are more than the smaller tasks in

Table 4 Parameters needed for WFs

Parameters	Value		
Size of WF (No. of Tasks/WF)	{100, 200, 300,400,500}		
Size of Task (No. of Instructions/Task)	{100 - 5000} MIs		
No. of Servers for each WF	Case 1: 15 servers with 100 tasks Case 2: 20 servers with 200 tasks Case 3: 30 servers with 300 tasks Case 4: 35 servers with 400 tasks Case 5: 50 servers with 500 tasks		
Distribution used to create WFs	Uniform random tasks distribution	70% Tasks with lower requirement (smaller tasks are more)	70% Tasks with higher requirement (larger tasks are more)

the workflow but, when the smaller tasks are more than the larger tasks, the quality of schedule decreases. In addition, we have also evaluated all the heuristics under uniform task distribution to produce fine grained results. Results of our experiments are in line with the results of [18] under higher task and machine heterogeneity instances, thereby validating our framework. To compare the results across different scenarios, results have been normalized on the values of MinMin algorithm for each scenario because MinMin exhibited higher values in most cases. In figure 5, the comparison of makespan over varying number of servers under uniform task distributions has been shown.

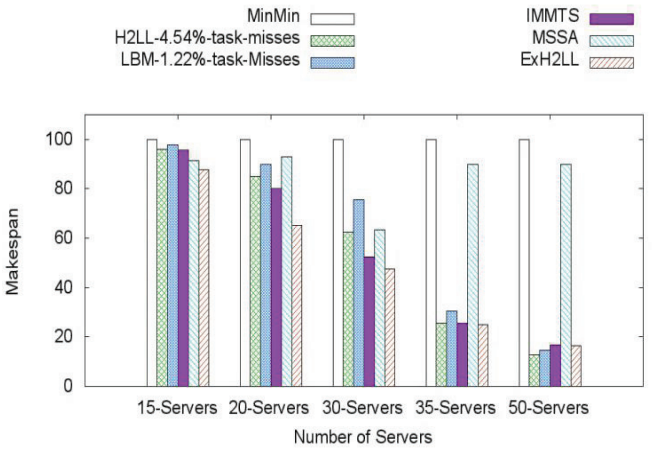


Figure 5 Makespan (Uniform Task Distribution).

The proposed heuristic outperforms the other algorithms in all cases except in the case of 50-servers where H2LL and LBM report lower makespans due to higher task missing rates. Moreover, we have also conducted the analysis to observe the task missing rate of H2LL and LBM heuristics on all of task distributions for each server case individually, as shown in figure 6. The results of figure 6 reveal that H2LL has 16% task missing rate

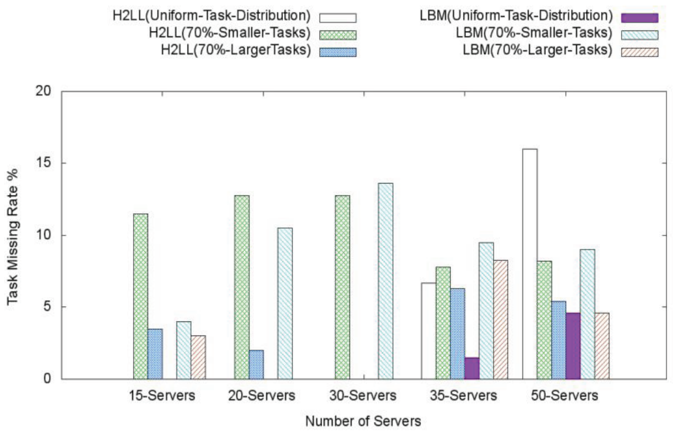


Figure 6 Average task missing rates of H2LL and LBM.

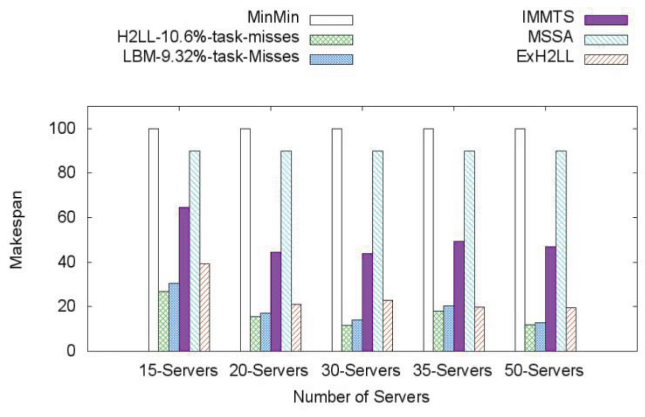


Figure 7 Makespan (70% Smaller Tasks).

while the LBM has 4.6% task missing rate. In Figure 7, where smaller tasks are more than the larger tasks, the results show that

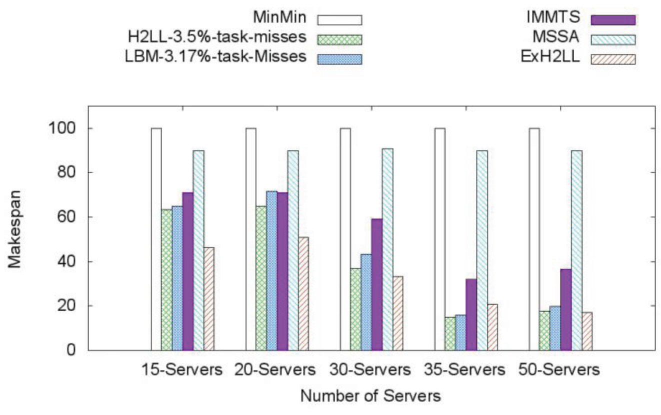


Figure 8 Makespan (70% Larger Tasks).

ExH2LL produced lowest makespan as compared to the other heuristics except the H2LL and LBM algorithms in all cases. The reason for H2LL and LBM having lower makespan is the higher task missing rate. In this case H2LL and LBM miss on average 10.6 % and 9.32 % tasks, respectively. MinMin heuristic degrades the systems performance by increasing makespan when number of smaller tasks are more than the larger tasks. Consequently, the overall makespan of MinMin and its variant MSSA, is also on a higher side as depicted in Figure 7.

Alternatively, the difference between, the overall makespan values for MinMin and MSSA heuristics, and other four heuristics, has reduced when majority of the tasks are large, i.e., 70% larger tasks. Majority of larger tasks is the ideal task distribution for MinMin, and its variants to produce lower makespan, as shown in Figure 8. Furthermore, Figure 8 shows that, IMMTS outperformed the MinMin because of its statistical task selection, superseded by H2LL, LBM, and ExH2LL. Moreover, ExH2LL outperformed in all the cases except 35-servers case, because in this case H2LL and LBM have 6.9% and 6.4% task missing rates on average, respectively.

The makespan and power consumption have tradeoff or conflict with each other, as explained in Section 1. Therefore, optimizing the makespan may increase the power consumption. The results in Figure 9, 10, and 11 show the power consumption for each of the task distribution scenarios along y-axis. Figure 9 reveals that in the worst case (35-servers) power consumption of ExH2LL heuristic has increased by 6watts as compared to MinMin algorithm. This increase in power consumption is amortized to achieve 84% makespan reduction as compared to MinMin heuristic. Moreover, the ExH2LL‘s power consumption in the worst case (35-servers) of 70% smaller tasks and 70% larger tasks distributions raises by 28watts and 11watts respectively, at the cost of higher makespan reduction. Moreover, in case of 70% smaller tasks distribution, H2LL heuristic has lower power consumption than the other heuristics even with lower makespan, because of higher task missing rates as shown in Figure 6. Similar behavior has been observed in the 70% larger tasks distribution, but with comparatively lesser ratios, as shown in Figure 11, because MinMin and its variants perform best in such cases.

The results in Figure 12, 13, and 14 show the normalized resource utilization. The results have been normalized on the values of ExH2LL because in this case it has the higher values

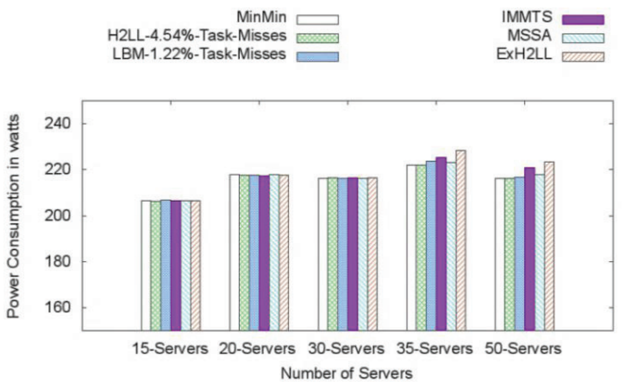


Figure 9 Power consumption (uniform task distribution).

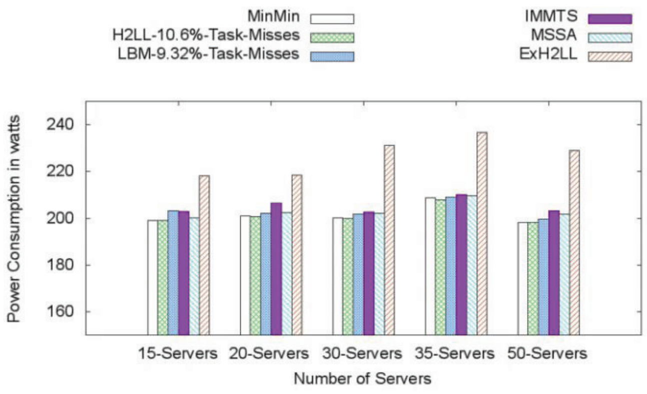


Figure 10 Power consumption (70 % smaller tasks).

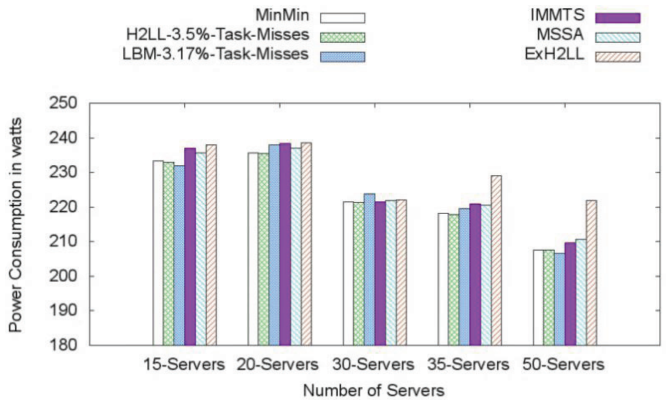


Figure 11 Power consumption (70% larger tasks).

in almost all cases. The resource utilization has increased for all the heuristics, because a uniform task distribution has been used, as shown in figure 12. On the other hand, under the 70% smaller task distribution the higher makespan and uneven load distribution leads to reduction in resource utilization rate regarding all the heuristics, except the proposed heuristic ExH2LL, because the 70% smaller tasks distribution is the worst most task distribution setting for the MinMin heuristic and its variants, as shown in Figure 13. Similar to uniform task distribution, under 70% larger tasks distribution almost all the heuristics showed significant increase in the utilization of resources with reference to the proposed algorithm. On the other hand, a common observation in figure 12, 13, and 14, portrays that the resource utilization in all the heuristics except the ExH2LL, starts to decrease when the

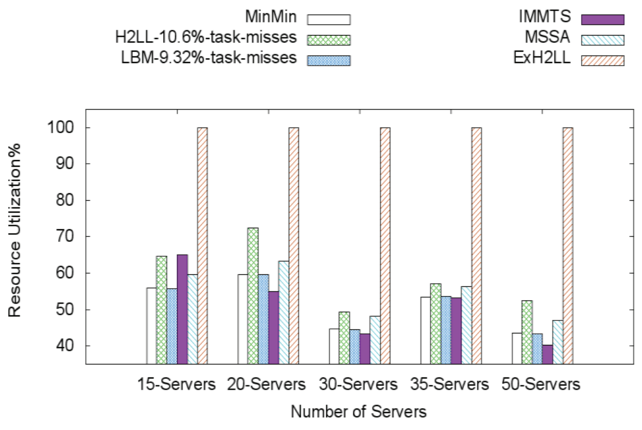


Figure 12 Resource Utilization (uniform task distribution).

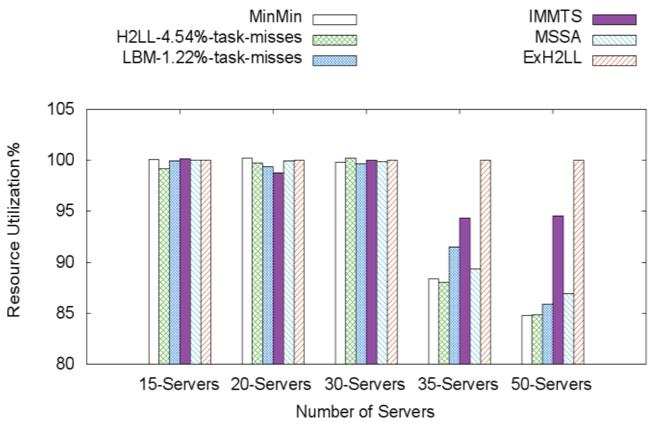


Figure 13 Resource Utilization (70% smaller tasks).

number of servers increases, due to unbalanced task distribution.

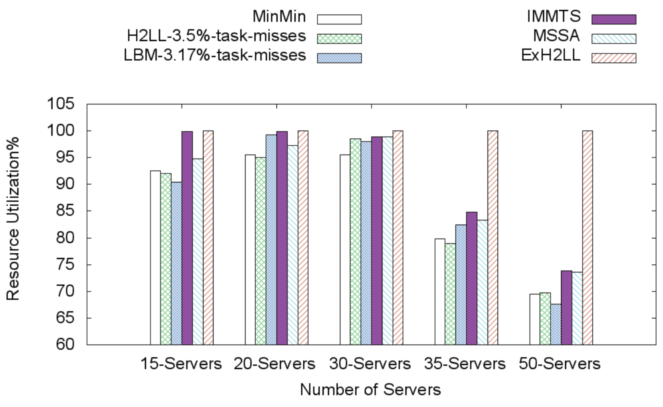


Figure 14 Resource Utilization (70% larger tasks).

7. CONCLUSIONS

Efficient resource utilization and makespan optimization play an important role in the design of modern LSCSs. There have been many proposals, such as Min-Min, IMMTS, H2LL, LBM and MSSA that attempt to optimize the makespan along with balancing the workload of the system. However, these proposals either fail to reduce the makespan in certain cases or do not result in efficient resource utilization. In this paper, we proposed an improved task scheduling heuristic for LSCSs that jointly optimizes two objective functions, i.e., makespan and power consumption. The proposed heuristic (ExH2LL) not only reduced the makespan and task missing rate, but also increased the resource utilization by evenly distributing the load among the available computing resources. The reduction of makespan caused slight increase in power consumption. However, the increase in power consumption is amortized by the significant decrease in makespan that leads to improved response time and reduced task missing rate. Moreover, another significant feature of the proposed heuristic is improved load distribution among the available resources.

Currently, none of the presented heuristics focused on communication cost, therefore, in our future work we intend to investigate the impact of reducing the communication cost on makespan, resource utilization, and power consumption. In this regard, literature review revealed that the schedules produced by duplication based heuristics have lower communication cost and makespan, but with higher resource wastage. Therefore, in our future work we aim to develop an energy efficient and resource-aware scheduling heuristic that not only optimizes the makespan but also reduces the communication cost.

REFERENCES

1. Benoit, A., Lefevre, L., Orgerie, A.C. and Raïs, I. Reducing the energy consumption of large-scale computing systems through combined shutdown policies with multiple constraints. *The International Journal of High Performance Computing Applications*, 32(1), pp.176-188, 2018.
2. Mustafa, S., Nazir, B., Hayat, A., Khan, A. R., Madani, S. A. “Resource management in cloud computing: Taxonomy, prospects, and challenges,” *Computers & Electrical Engineering*, vol. 47, pp. 186-203. 2015
3. Liu, J., Morales, L.P., Pacitti, E., Costan, A., Valduriez, P., Antoniu, G. and Mattoso. Efficient Scheduling of Scientific Workflows using Hot Metadata in a Multisite Cloud. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
4. Hassan, Q.F., Khan, A. R., Madani, S.A., “Internet of Things: Challenges, Advances, and Applications,” Chapman and Hall/CRC, 2018.
5. Valduriez, P., Mattoso, M., Akbarinia, R., Borges, H., Camata, J., Coutinho, A., Gaspar, D., Lemus, N., Liu, J., Lustosa, H. and Masseglia, F., August. Scientific Data Analysis Using Data-Intensive Scalable Computing: the SciDISC Project. *In LADaS: Latin America Data Science Workshop* (No. 2170). CEUR-WS. org., 2018.
6. Tyagi, R. and Gupta, S.K., A Survey on Scheduling Algorithms for Parallel and Distributed Systems. *In Silicon Photonics & High Performance Computing* (pp. 51-64). Springer, Singapore, 2018.
7. Nayak, S.C., Parida, S., Tripathy, C. and Pattnaik, P.K., Modeling of Task Scheduling Algorithm Using Petri-Net in Cloud Computing. *In Progress in Advanced Computing and Intelligent Engineering* (pp. 633-643). Springer, Singapore, 2018.
8. Gai, Keke, et al. “Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing,” *Journal of Network and Computer Applications*, vol. 59, pp. 46-54, 2016.
9. R. Lent, “Grid Scheduling with Makespan and Energy-Based Goals,” *Journal of Grid Computing,* vol. 13, pp. 527-546, 2015.
10. Kwon, S., Ntaimo, L. and Gautam, N., Demand Response in Data

Centers: Integration of Server Provisioning and Power Procurement. *IEEE Transactions on Smart Grid*, 2018.

11. Rossi, Fábio D., et al. "E-eco: Performance-aware energy-efficient cloud data center orchestration," *Journal of Network and Computer Applications*, vol. 78, pp. 83-96, 2016.
12. Sampaio, A.M. and Barbosa, J.G., A comparative cost analysis of fault-tolerance mechanisms for availability on the cloud. *Sustainable Computing: Informatics and Systems*, 19, pp.315-323, 2018.
13. C. Liu and S. Baskiyar, "A general distributed scalable grid scheduler for independent tasks," *Journal of Parallel and Distributed Computing,* vol. 69, pp. 307-314, 2009.
14. H. Khajemohammadi, A. Fanian, and T. A. Gulliver, "Efficient workflow scheduling for grid computing using a leveled multi-objective genetic algorithm," *Journal of Grid Computing,* vol. 12, pp. 637-663, 2014.
15. Zhang, P. and Zhou, M., "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Transactions on Automation Science and Engineering*, 15(2), pp.772-783, 2018.
16. Y. Li, Y. Liu, and D. Qian, "A heuristic energy-aware scheduling algorithm for heterogeneous clusters," *2009 15th International Conference on Parallel and Distributed Systems*, 2009, pp. 407-413.
17. Shaheen, Q., Shiraz, M., Khan, S., Majeed, R., Guizani, M., Khan, N. and Aseere, A.M., Towards Energy Saving in Computational Clouds: Taxonomy, Review, and Open Challenges. IEEE Access, 2018.
18. F. Pinel, B. Dorronsoro, J. E. Pecero, P. Bouvry, and S. U. Khan, "A two-phase heuristic for the energy-efficient scheduling of independent tasks on computational grids," *Cluster computing,* vol. 16, pp. 421-433, 2013.
19. J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," in *Metaheuristics for scheduling in distributed computing environments*, ed: Springer, 2008, pp. 173-214.
20. H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 13, pp. 260-274, 2002.
21. K. A. Manudhane and A. Wadhe, "Comparative Study of Static Task Scheduling Algorithms for Heterogeneous Systems," *International Journal on Computer Science & Engineering,* vol. 5, 2013.
22. J. Mei, K. Li, and K. Li, "A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems," *The Journal of Supercomputing,* vol. 68, pp. 1347-1377, 2014.
23. Y. Kang and Y. Lin, "A recursive algorithm for scheduling of tasks in a heterogeneous distributed environment," in *Biomedical Engineering and Informatics (BMEI), 4th International Conference on*, 2011, pp. 2099-2103.
24. Y.-H. Lee, S. Leu, and R.-S. Chang, "Improving job scheduling algorithms in a grid environment," *Future Generation Computer Systems,* vol. 27, pp. 991-998, 2011.
25. S. S. Chauhan and R. Joshi, "QoS guided heuristic algorithms for grid task scheduling," *International journal of computer applications,* vol. 2, pp. 24-31, 2010.
26. R.-S. Chang, C.-F. Lin, and J.-J. Chen, "Selecting the most fitting resource for task execution," *Future Generation Computer Systems,* vol. 27, pp. 227-231, 2011.
27. T. Kokilavani and D. D. G. Amalarethinam, "Load balanced min-min algorithm for static meta-task scheduling in grid computing," *International Journal of Computer Applications,* vol. 20, pp. 43-49, 2011.
28. S. Parsa and R. Entezari-Maleki, "RASA: A new task scheduling algorithm in grid environment," *World Applied sciences journal,* vol. 7, pp. 152-160, 2009.
29. P. Suri and M. Singh, "An efficient decentralized load balancing algorithm for grid," in *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, 2010, pp. 10-13.
30. S. Anousha and M. Ahmadi, "An Improved Min-Min Task Scheduling Algorithm in Grid Computing," in *Grid and Pervasive Computing*, ed: Springer, 2013, pp. 103-113.
31. T. K. Ghosh, R. Goswami, S. Bera, and S. Barman, "Load balanced static grid scheduling using Max-Min heuristic," in *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on*, 2012, pp. 419-423.
32. S. K. Panda and P. M. Khilar, "MSSA: A M-Level Sufferage-Based Scheduling Algorithm in Grid Environment," in *Distributed Computing and Internet Technology*, ed: Springer, 2013, pp. 410-419.
33. E. Kartal Tabak, B. Barla Cambazoglu, and C. Aykanat, "Improving the performance of independenttask assignment heuristics minmin, maxmin and sufferage," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 25, pp. 1244-1256, 2014.
34. P. Bindu, R. Venkatesan, and K. Ramalakshmi, "Perspective study on resource level load balancing in grid computing environments," in *Electronics Computer Technology (ICECT), 3rd International Conference on*, 2011, pp. 321-325.
35. L. Ahlroth, A. Schumacher, and P. Orponen, "Online bin packing with delay and holding costs," *Operations Research Letters,* vol. 41, pp. 1-6, 2013.
36. G. B. Mertzios, M. Shalom, A. Voloshin, P. W. Wong, and S. Zaks, "Optimizing busy time on parallel machines," *Theoretical Computer Science,* vol. 562, pp. 524-541, 2015.
37. F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future generation computer systems,* vol. 26, pp. 608-621, 2010.
38. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, and D. Hensgen, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing,* vol. 61, pp. 810-837, 2001.
39. A. M. Al-Qawasmeh, A. A. Maciejewski, H. Wang, J. Smith, H. J. Siegel, and J. Potter, "Statistical measures for quantifying task and machine heterogeneities," *The Journal of Supercomputing,* vol. 57, pp. 34-50, 2011.
40. S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering,* vol. 3, pp. 195-208, 2000.
41. B. Eslamnour and S. Ali, "Measuring robustness of computing systems," *Simulation Modelling Practice and Theory,* vol. 17, pp. 1457-1467, 2009.
42. D. Huang, Y. Yuan, L.-j. Zhang, and K.-q. Zhao, "Research on tasks scheduling algorithms for dynamic and uncertain computing grid based on a+ bi connection number of SPA," *Journal of Software,* vol. 4, pp. 1102-1109, 2009.
43. S. U. Khan and I. Ahmad, "Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, p. 10 pp.
44. P. Chitra, R. Rajaram, and P. Venkatesh, "Application and comparison of hybrid evolutionary multiobjective optimization algorithms for solving task scheduling problem on heterogeneous systems," *Applied Soft Computing,* vol. 11, pp. 2725-2734, 2011.
45. S. K. U. Zaman, A. R. Khan, J. Shuja, T. Maqsood, F. Rehman, S. Mustafa, "A Systems Overview of Commercial Data Centers: Initial Energy and Cost Analysis," *International Journal of Information Technology and Web Engineering*, vol. 14, pp. 42-65, 2019.
46. S. K. U. Zaman, A. R. Khan, S. U. R. Malik, A. N. Khan, T.

Maqsood, S. A. Madani, "Formal verification and performance evaluation of task scheduling heuristics for makespan optimization and workflow distribution in large-scale computing systems," *Computer Systems Science and Engineering*, vol. 32, pp. 227-241. 2017.

47. SPEC. (2010). *Fourth Quarter 2010 SPECpower_ssj2008 Results*. Available: http://www.spec.org/power_ssj2008/results/res2010q4/
48. T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE,* vol. 77, pp. 541-580, 1989.
49. L. de Moura and N. Bjørner, "Satisfiability Modulo Theories: An Appetizer," in *Formal Methods: Foundations and Applications*. vol. 5902, M. Oliveira and J. Woodcock, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 23-36.
50. J. O. Gutierrez-Garcia and K. M. Sim, "A family of heuristics for agent-based elastic cloud bag-of-tasks concurrent scheduling," *Future Generation Computer Systems,* vol. 29, pp. 1682-1699, 2013.