NESUS cost IC1305
Network for Sustainable Ultrascale Computing

Proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)
Porto, Portugal

Jesus Carretero, Javier Garcia Blas
Jorge Barbosa, Ricardo Morla
(Editors)

August 27-28, 2014

Scheduling Real-Time Jobs in Distributed Systems - Simulation and Performance Analysis

Georgios L. Stavrinides and Helen D. Karatza

Aristotle University of Thessaloniki, Greece

gstavrin@csd.auth.gr, karatza@csd.auth.gr

Abstract

One of the major challenges in ultrascale systems is the effective scheduling of complex jobs within strict timing constraints. The distributed and heterogeneous system resources constitute another critical issue that must be addressed by the employed scheduling strategy. In this paper, we investigate by simulation the performance of various policies for the scheduling of real-time directed acyclic graphs in a heterogeneous distributed environment. We apply bin packing techniques during the processor selection phase of the scheduling process, in order to utilize schedule gaps and thus enhance existing list scheduling methods. The simulation results show that the proposed policies outperform all of the other examined algorithms.

Keywords Scheduling, Distributed systems, Real-time jobs, Simulation, Performance evaluation

I. Introduction

The rapid developments in computing and communication technologies have led to the emergence of *ultrascale computing*, which provides a large-scale, heterogeneous distributed platform for the processing of complex jobs [1, 2, 3, 4]. The sustainability of a computing environment of such scale and complexity is one of the most crucial aspects of ultrascale computing.

I.1 Motivation

One of the major challenges in ultrascale systems is the *effective scheduling* and processing of a large number of *interdependent* tasks within strict *timing constraints*. Such tasks often have precedence constraints among them and thus form a *real-time directed acyclic graph (DAG)*, with an end-to-end deadline. In case a real-time job cannot meet its deadline, then depending on its criticality, its result will be useless or even worse, this may have catastrophic consequences on the environment under control [5]. The *distributed* and *heterogeneous* resources of the target system constitute another critical issue that must be addressed during the scheduling of real-time complex jobs [6].

I.2 Contribution

We investigate by simulation the performance of various policies for the scheduling of real-time DAGs in a heterogeneous distributed environment. Our goal is to apply *effective techniques* during the scheduling process, in order to guarantee that every real-time job will meet its deadline.

I.3 Related Work

A large number of job scheduling techniques have been developed and studied in the literature [7, 8, 9, 10, 11, 12, 13]. The most commonly used real-time scheduling algorithm is the *Earliest Deadline First (EDF)* [14]. According to this policy, the job with the earliest deadline has the highest priority for execution. An efficient and practical method for scheduling directed acyclic graphs, is the *list scheduling approach*, according to which the tasks are arranged in a prioritized list. Subsequently, each task is allocated to the processor that minimizes a cost function, such as the task estimated start time [15]. A simple list scheduling algorithm is the *Highest Level First (HLF)* [16], which prioritizes each component task according to the longest path from the particular task to an exit task in the DAG.

Based on the observation that idle time slots may form in the schedule of a processor due to the data dependencies of the tasks in a DAG, Kruatrachue and Lewis in [17] propose the *Insertion Scheduling Heuristic (ISH)*. According to this method which is based on HLF, during the processor selection phase, a task may be inserted into an idle time slot in a processor's schedule, as long as it does not delay the execution of the succeeding task in the schedule and provided that it cannot start earlier on any other processor. Topcuoglu et al. in [18] present the *Heterogeneous Earliest Finish Time (HEFT)* list scheduling strategy, which is essentially an alternative version of ISH, adapted for heterogeneous systems.

An improved version of HEFT is presented in [15] by Arabnejad and Barbosa. It introduces a look ahead feature based on an optimistic cost table. Jiang et al. in [19] present a novel clustering algorithm, the *Path Clustering Heuristic with Distributed Gap Search (PCH-DGS)*, for the scheduling of multiple DAGs in a heterogeneous cloud. Their proposed method tries to insert each group of tasks into the first available idle time slot in a processor's schedule (a DAG's tasks are partitioned into groups in an attempt to minimize the communication cost between them). In case the time gap cannot accommodate all of the tasks of the group, the rest of the group's tasks are inserted into the next available schedule gap of the same or other processor.

All of the above algorithms are static and do not take into account any timing constraints. Moreover, they essentially utilize schedule gaps according to the First Fit bin packing technique [20]. Cheng et al. propose in [21] a scheduling heuristic, *Least Space-Time First (LSTF)*, that takes into account both the precedence and the timing constraints among the tasks. However, their algorithm does not utilize any schedule idle time slots. In this paper, we apply various bin packing techniques (First Fit, Best Fit and Worst Fit) during the processor selection phase of the scheduling process, in order to utilize schedule gaps and thus enhance existing list scheduling methods. Moreover, our policies are suitable for the dynamic scheduling of multiple real-time DAGs.

II. System and Workload Models

The real-time complex jobs arrive in a Poisson stream with rate λ at a heterogeneous cluster that consists of a set of q fully connected processors. Each processor p_i serves its own local queue of tasks (it has its own local memory) and has an execution rate μ_i. The transfer rate between two processors p_i and p_j is denoted by ν_{ij}. The processor execution rates and the communication links data transfer rates may vary. The heterogeneous cluster is dedicated to real-time jobs and it may be part of a computational grid or cloud. The jobs arrive at a central scheduler [22], where their unscheduled tasks wait in a global waiting queue until they get ready to be scheduled. A task becomes ready to be scheduled when it has no predecessors or when all of its parent tasks have finished execution.

The *heterogeneity factor HF* of the system denotes the difference in the speed of the processors, as well as in the transfer rate of the communication links. The execution rate of each processor in the system is uniformly distributed in the range $[\overline{\mu}\cdot(1-HF/2), \overline{\mu}\cdot(1+HF/2)]$, where $\overline{\mu}$ is the mean execution rate of the processors. The data transfer rate of each communication link is uniformly distributed in the range $[\overline{\nu}\cdot(1-HF/2), \overline{\nu}\cdot(1+HF/2)]$, where $\overline{\nu}$ is the mean data transfer rate of the communication links.

Each job that arrives at the cluster is a directed acyclic graph $G=(V,E)$, where V is the set of the nodes of the graph and E is the set of the directed edges between the nodes. Each node represents a component task n_i, whereas a directed edge e_{ij} between two tasks n_i and n_j represents the data that must be transmitted from task n_i to task n_j. Each node n_i in a DAG has a weight w_i, which denotes its *computational volume* (i.e. the amount of computational operations needed to be executed). The *computational cost* of the task n_i on a processor p_j is given by:

$$Comp(n_i, p_j) = w_i/\mu_j \tag{1}$$

where μ_j is the execution rate of processor p_j. The *level* L_i of a task n_i is the length of the longest path from the particular task to an exit task. The length of a path in the graph is the sum of the computational and communication costs of all of the tasks and edges, respectively, on the path.

Each edge e_{ij} between two nodes n_i and n_j has a weight c_{ij} which represents its *communication volume* (i.e. the amount of data needed to be transmitted between the two tasks). The *communication cost* of the edge e_{ij} is incurred when data are transmitted from task n_i (scheduled on processor p_m) to task n_j (scheduled on processor p_n) and is defined as:

$$Comm\left((n_i, p_m), (n_j, p_n)\right) = c_{ij}/\nu_{mn} \tag{2}$$

where ν_{mn} is the data transfer rate of the communication link between the processors p_m and p_n.

The *communication to computation ratio CCR* of a job is the ratio of its average communication cost to its average computational cost on a target system and is given by:

$$CCR = \frac{\sum_{e_{ij}\in E}\overline{Comm(e_{ij})}}{\sum_{n_i\in V}\overline{Comp(n_i)}} \tag{3}$$

where V and E are the sets of the nodes and the edges of the job respectively. $\overline{Comm(e_{ij})}$ is the average communication cost of the edge e_{ij} over all of the communication links in the system, whereas $\overline{Comp(n_i)}$ is the average computational cost of the task n_i over all of the processors in the system. An example task graph is illustrated in figure 1.

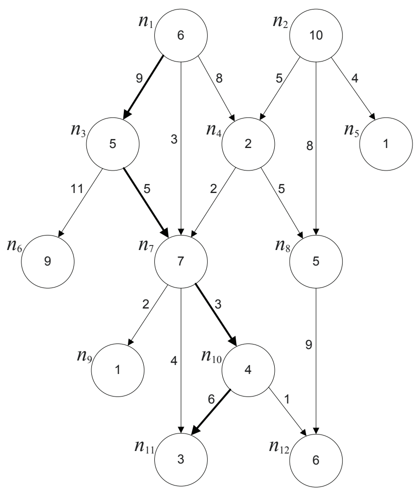

Figure 1: An example DAG with two entry tasks and five exit tasks. The number in each node denotes the average computational cost of the represented task. The number on each edge denotes the average communication cost between the two tasks that it connects. The critical path (i.e. the longest path) of the graph is depicted with thick arrows.

III. Scheduling Strategies

In order to schedule the ready tasks in the global waiting queue, a list scheduling heuristic is employed. This method consists of two phases: (a) a *task selection phase* and (b) a *processor selection phase*.

III.1 Task Selection Phase

Each task is assigned a priority according to one of the following policies:

- ***Earliest Deadline First (EDF)*:** the priority value of each task is equal to the absolute end-to-end deadline of its job. The task with the smallest priority value has the highest priority for scheduling.

- ***Highest Level First (HLF)*:** the priority value of each task is equal to its level. The task with the largest priority value has the highest priority for scheduling.

- ***Least Space-Time First (LSTF)*:** the priority value of a task n_i is equal to its *space-time* ST_i parameter, which is defined as $ST_i(t) = D - t - L_i$, where D is the absolute end-to-end deadline of the task's job, t is the current time instant and L_i is the task's level.

The tasks are arranged in a list, according to their priority. The task with the highest priority for scheduling is placed first in the list. In case two or more tasks have the same priority value, they are arranged in descending order of average computational costs.

III.2 Processor Selection Phase

Once a task is selected by the scheduler, it is allocated to the processor that can provide it with the earliest *estimated start time EST* (ties are broken randomly). The EST of a ready task n_i on a processor p_n is given by:

$$EST(n_i, p_n) = \max\{T_{data}(n_i, p_n), T_{idle}(n_i, p_n)\} \tag{4}$$

where $T_{data}(n_i, p_n)$ is the time at which all input data of task n_i will be available on processor p_n, whereas $T_{idle}(n_i, p_n)$ is the time at which p_n will be able to execute task n_i.

In order to calculate the term $T_{idle}(n_i, p_n)$, the *potential position* of task n_i on processor p_n is determined. This is the position at which the task n_i would be placed according to its priority in the local waiting queue of processor p_n, if it was actually assigned to that particular processor. An alternative, more effective method to determine the potential position of a task in a processor's queue is described below.

III.3 Alternative Method of Potential Position Calculation

According to our proposed method, during the processor selection phase of the scheduling process, the potential position of a ready task in a processor's queue is determined by taking into account not only the task's priority, but also the idle time slots in the processor's schedule that can be utilized. Specifically:

- **Step 1:** We first find the *initial potential position* at which the ready task n_i would be placed in the processor's queue, according to its priority and so that it does not precede the task that is placed after the last exploited idle time slot in the schedule of the processor. The scheduled tasks placed in the area between the head of the queue and the initial potential position of task n_i, form the exploitable area of the queue.

- **Step 2:** The tasks in the exploitable area of the queue are examined whether they can give idle time slots, starting from the task at the head of the queue. An idle time slot is candidate for exploitation by the ready task n_i only when it can accommodate its computational cost. Moreover, task n_i must not delay any succeeding tasks in the processor's queue.

  The task is inserted into an idle time slot according to one of the following bin packing policies:

  – ***First Fit (FF)*:** the task is placed into the first idle time slot where its computational cost fits.
  – ***Best Fit (BF)*:** the task is placed into the idle time slot where its computational cost fits and where it leaves the minimum unused time possible.
  – ***Worst Fit (WF)*:** the task is placed into the idle time slot where its computational cost fits and where it leaves the maximum unused time possible.

  The above procedure has as a result the calculation of the *final potential position* of the ready task n_i.

The pseudocode for the method described above is given in algorithm 1. The scheduling method used in this paper is an enhanced version of the one described in our previous work in [23]. Specifically, in this paper, in case a job misses its deadline, not only are its scheduled tasks that are waiting in processor local queues aborted, but also, all of the other tasks that are waiting in the particular queues are rescheduled (on the same processors), according to their priority. This is necessary, due to the fact that a task removal from a queue may lead to the cancellation of utilized idle time slots or to the creation of new ones that could be exploited by other tasks that are waiting in the queue. Other differences with our previous work in [23] include: (a) the CCR parameter is defined differently in this paper and (b) different values for the simulation input parameters are used.

IV. Performance Evaluation

IV.1 Performance Metric

The performance of the investigated scheduling policies was evaluated by simulation. In order to have full control on all of the required system and workload parameters, we implemented our own discrete-event simulation program in C++. As a performance metric, the *job guarantee ratio* JGR was employed, which is defined as:

$$JGR = \frac{TNJG}{TNJA} \tag{5}$$

where $TNJG$ is the total number of jobs guaranteed, i.e. the total number of jobs that met their deadline. $TNJA$ is the total number of job arrivals at the system, during the time period the system was observed.

IV.2 Simulation Input Parameters

In our simulation experiments we used synthetic workload, in order to obtain unbiased results. The task graphs were generated randomly, using our own custom DAG generator, as described in [24].

Algorithm 1 Alternative method of potential position calculation.

Input: A ready task n_i and a processor p_n.
Output: Final potential position of task n_i in p_n's queue.

find $initialPotentialPosition$ of task n_i in p_n's queue
determine exploitable area of p_n's queue
$finalPotentialPosition \leftarrow initialPotentialPosition$
$spareTime \leftarrow -1$
get first task n_j in exploitable area of p_n's queue
repeat
  if task n_j forms a schedule hole **then**
    if $Comp(n_i, p_n) \leq T_{data}(n_j, p_n) - EST(n_i, p_n)$ **then**
      if Bin Packing Policy = First Fit **then**
        $finalPotentialPosition \leftarrow currentQueuePosition$
        return $finalPotentialPosition$
      else if Bin Packing Policy = Best Fit **then**
        if $spareTime = -1$ **or** $spareTime > spareTimeOfThisScheduleHole$ **then**
          $spareTime \leftarrow spareTimeOfThisScheduleHole$
          $finalPotentialPosition \leftarrow currentQueuePosition$
        end if
      else if Bin Packing Policy = Worst Fit **then**
        if $spareTime < spareTimeOfThisScheduleHole$ **then**
          $spareTime \leftarrow spareTimeOfThisScheduleHole$
          $finalPotentialPosition \leftarrow currentQueuePosition$
        end if
      end if
    end if
  end if
  get next task n_j in exploitable area of p_n's queue
until all tasks in exploitable area of p_n's queue are examined
return $finalPotentialPosition$

The simulation input parameters are summarized in table 1. The computational volume of a task in a graph is exponential with mean $\overline{w}$. The communication volume of an edge is exponential with mean $\overline{c}$. The relative deadline of each job is uniformly distributed in the range $[CPL, 2CPL]$, where CPL is the length of the critical (i.e. the longest) path in the graph. The heterogeneity factor of the system is considered to be equal to $HF = 0.5$. That is, the target system is considered to feature a moderate degree of heterogeneity.

Parameter description	Value
Number of processors in the system	$q = 64$
Mean execution rate of processors	$\overline{\mu} = 1$
Mean data trans. rate of comm. links	$\overline{v} = 1$
Heterogeneity factor	$HF = 0.5$
Number of tasks in each job	$a \sim U[1, 64]$
Arrival rate of the jobs	$\lambda = \{0.2, 0.25, 0.3, 0.35\}$
Relative deadline of each job	$RD \sim U[CPL, 2CPL]$
CCR of the jobs	$CCR = \{0.1, 1, 10\}$
Mean comp. volume of the tasks	$\overline{w} = 10$ ($CCR = 0.1$) and $\overline{w} = 1$ ($CCR = \{1, 10\}$)

Table 1: Simulation input parameters.

IV.3 Simulation Results

We investigated the performance of the scheduling strategies included in table 2, with respect to the arrival rate of the jobs, for DAGs with various communication to computation ratios:

- computationally intensive DAGs ($CCR = 0.1$);
- moderate DAGs ($CCR = 1$);
- communication intensive DAGs ($CCR = 10$).

Scheduling Strategy	Task Selection Phase (task prioritization)	Processor Selection Phase (utilization of idle time slots)
EDF	Earliest Deadline First	No
EDF_FF	Earliest Deadline First	First Fit
EDF_BF	Earliest Deadline First	Best Fit
EDF_WF	Earliest Deadline First	Worst Fit
HLF	Highest Level First	No
HLF_FF	Highest Level First	First Fit
HLF_BF	Highest Level First	Best Fit
HLF_WF	Highest Level First	Worst Fit
LSTF	Least Space-Time First	No
LSTF_FF	Least Space-Time First	First Fit
LSTF_BF	Least Space-Time First	Best Fit
LSTF_WF	Least Space-Time First	Worst Fit

Table 2: Examined scheduling strategies.

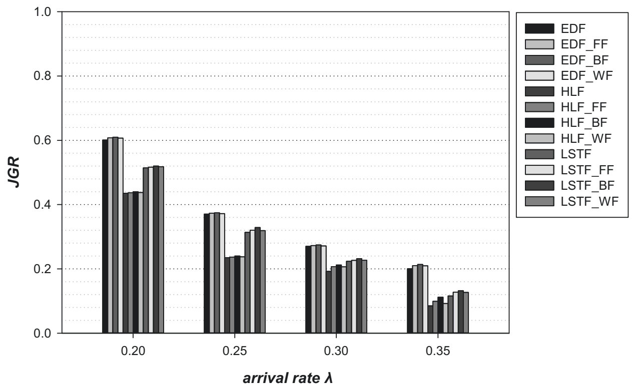

Figure 2: JGR vs. λ for $CCR = 0.1$.

Figures 2, 3 and 4 show the simulation results in each of the above cases, respectively.

The simulation results suggest that the scheduling strategies that employ the EDF policy in the task selection phase, exhibit better performance than the strategies that employ the HLF and the LSTF policies. This is more obvious in the case of computationally intensive DAGs. Furthermore, the proposed alternative versions of the scheduling algorithms that utilize idle time slots in the processor selection phase, outperform their respective counterparts that do not utilize idle time gaps.

Figure 5 shows the average improvement in the system performance for the proposed scheduling policies, compared to their counterpart methods that do not utilize schedule gaps. The improvement is more apparent in the case of computationally intensive workload. Specifically, the average improvement in this case is shown in table 3 for each scheduling strategy. Even though the scheduling strategies that employ the HLF and the LSTF policies in the task selection phase benefit more by the utilization of idle time slots in the processor selection phase than the respective strategies that use EDF, the latter outperform their corresponding counterparts.

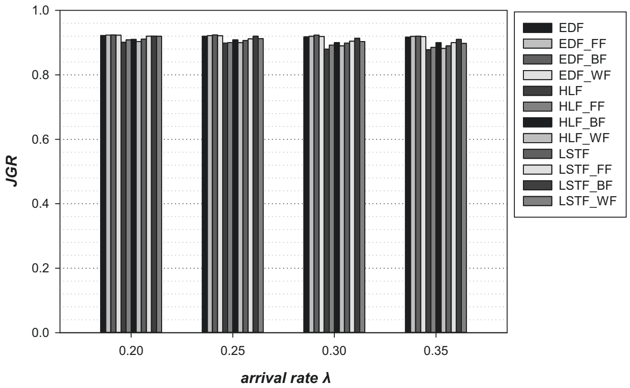

Figure 3: JGR vs. λ for $CCR = 1$.

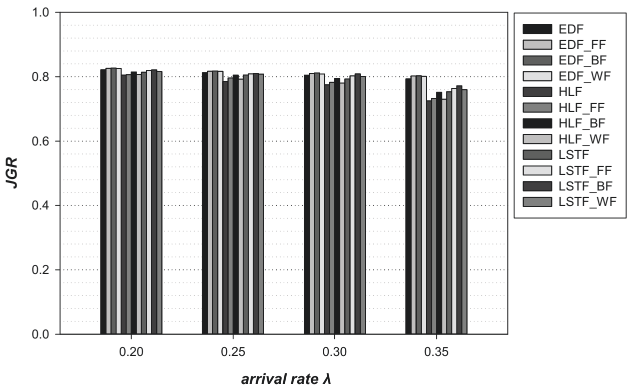

Figure 4: JGR vs. λ for $CCR = 10$.

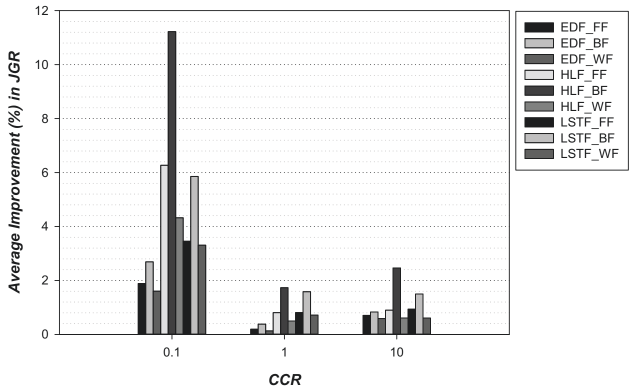

Figure 5: The average improvement (%) in the system performance for the proposed scheduling strategies, compared to their counterpart policies that do not utilize idle time slots.

Scheduling Strategy	**Average Improvement in JGR**
EDF_FF	1.89%
EDF_BF	2.69%
EDF_WF	1.60%
HLF_FF	6.27%
HLF_BF	11.22%
HLF_WF	4.32%
LSTF_FF	3.45%
LSTF_BF	5.86%
LSTF_WF	3.31%

Table 3: The average improvement in the system performance for the proposed scheduling strategies, in the case of computationally intensive workload.

V. Conclusions and Future Directions

In this paper, we investigated by simulation the performance of various policies for the scheduling of real-time DAGs in a heterogeneous distributed environment. We applied bin packing techniques during the processor selection phase of the scheduling process, in order to utilize schedule gaps and thus enhance existing list scheduling algorithms.

The simulation results suggest that in the case where the utilization of idle time slots is based on the Best Fit bin packing technique, the system exhibits better performance than in the case where the First Fit and the Worst Fit policies are used. Overall, the proposed EDF_BF scheduling strategy outperforms all of the other examined algorithms.

Ultrascale systems may utilize multicore architectures. Moreover, energy efficiency and fault tolerance are vital aspects of their sustainability [25, 26]. Therefore, our future research plans include the adaptation of our proposed scheduling strategies in order to meet those needs.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST program Action IC1305, "Network for Sustainable Ultrascale Computing (NESUS)".

References

[1] R. Boutaba, L. Cheng, and Q. Zhang, "On cloud computational models and the heterogeneity challenge", *Journal of Internet Services and Applications*, Springer, vol. 3, no. 1, pp. 77-86, 2012.

[2] J. Carretero and J. D. Garcia, "The Internet of Things: connecting the world", *Personal and Ubiquitous Computing*, Springer-Verlag, vol. 18, no. 2, pp. 445-447, 2014.

[3] J. Carretero and J. G. Blas, "Introduction to cloud computing: platforms and solutions", *Cluster Computing*, Springer, to appear.

[4] G. L. Stavrinides and H. D. Karatza, "The impact of resource heterogeneity on the timeliness of hard real-time complex jobs", in *Proceedings of the 7th International Conference on Pervasive Technologies Related to Assistive Environments (PETRA'14), Workshop on Distributed Sensor Systems for Assistive Environments (Di-Sensa)*, Island of Rhodes, Greece, May 2014, to appear.

[5] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed., Springer, New York, U.S.A., 2011.

[6] G. L. Stavrinides and H. D. Karatza, "The impact of input error on the scheduling of task graphs with imprecise computations in heterogeneous distributed real-time systems", in *Proceedings of the 18th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'11)*, Venice, Italy, June 2011, pp. 273-287.

[7] H. D. Karatza, "Performance of gang scheduling policies in the presence of critical sporadic jobs in distributed systems", in *Proceedings of the 2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'07)*, San Diego, U.S.A., July 2007, pp. 547-554.

[8] G. L. Stavrinides and H. D. Karatza, "Performance evaluation of gang scheduling in distributed real-time systems with possible software faults", in *Proceedings of the 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'08)*, Edinburgh, U.K., June 2008, pp. 1-7.

[9] H. D. Karatza, "Performance of gang scheduling strategies in a parallel system", *Simulation Modelling Practice and Theory*, Elsevier, vol. 17, no. 2, pp. 430-441, 2009.

[10] G. L. Stavrinides and H. D. Karatza, "Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes", *Future Generation Computer Systems*, Elsevier, vol. 28, no. 7, pp. 977-988, 2012.

[11] K. Chatterjee, A. Kossler, and U. Schmid, "Automated analysis of real-time scheduling using graph games", in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC'13)*, Philadelphia, U.S.A., April 2013, pp. 163-172.

[12] H. Yu, Y. Ha, and B. Veeravalli, "Quality-driven dynamic scheduling for real-time adaptive applications on multiprocessor systems", *IEEE Transactions on Computers*, IEEE, vol. 62, no. 10, pp. 2026-2040, 2013.

[13] H. D. Karatza, "Scheduling jobs with different characteristics in distributed systems", in *Proceedings of the 2014 International Conference on Computer, Information and Telecommunication Systems (CITS 2014)*, Jeju Island, South Korea, July 2014, pp. 1-5.

[14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the ACM*, ACM, vol. 20, no. 1, pp. 46-61, 1973.

[15] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table", *IEEE Transactions on Parallel and Distributed Systems*, IEEE, vol. 25, no. 3, pp. 682-694, 2014.

[16] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems", *Communications of the ACM*, ACM, vol. 17, no. 12, pp. 685-690, 1974.

[17] B. Kruatrachue and T. G. Lewis, "Duplication scheduling heuristic, a new precedence task scheduler for parallel systems", Technical Report 87-60-3, Oregon State University, Corvallis, Oregon, U.S.A., 1987.

[18] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Transactions on Parallel and Distributed Systems*, IEEE, vol. 13, no. 3, pp. 260-274, 2002.

[19] H. J. Jiang, K. C. Huang, H. Y. Chang, D. S. Gu, and P. J. Shih, "Scheduling concurrent workflows in HPC cloud through exploiting schedule gaps", in *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'11)*, Melbourne, Australia, October 2011, pp. 282-293.

[20] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: survey and classification", *Handbook of Combinatorial Optimization*, Springer, pp. 455-531, 2013.

[21] B. C. Cheng, A. D. Stoyenko, T. J. Marlowe, and S. K. Baruah, "LSTF: a new scheduling policy for complex real-time tasks in multiple processor systems", *Automatica*, Elsevier, vol. 33, no. 5, pp. 921-926, 1997.

[22] X. Zhu, X. Qin, and M. Qiu, "QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters", *IEEE Transactions on Computers*, IEEE, vol. 60, no. 6, pp. 800-812, 2011.

[23] G. L. Stavrinides and H. D. Karatza, "Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques", *Simulation Modelling Practice and Theory*, Elsevier, vol. 19, no. 1, pp. 540-552, 2011.

[24] G. L. Stavrinides and H. D. Karatza, "Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations", *Journal of Systems and Software*, Elsevier, vol. 83, no. 6, pp. 1004-1014, 2010.

[25] G. L. Stavrinides and H. D. Karatza, "Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations", *Simulation: Transactions of the Society for Modeling and Simulation International*, Sage Publications, Special Issue on Performance Evaluation of Computer and Telecommunication Systems, vol. 85, no. 8, pp. 525-536, 2009.

[26] G. Terzopoulos and H. D. Karatza, "Performance evaluation and energy consumption of a real-time heterogeneous grid system using DVS and DPM", *Simulation Modelling Practice and Theory*, Elsevier, vol. 36, pp. 33-43, 2013.