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 optimized 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 \( \lambda \) at a heterogeneous cluster that consists of a set of \( q \) fully connected processors. Each processor \( p_j \) serves its own local queue of tasks (it has its own local memory) and has an execution rate \( \mu_j \). The transfer rate between two processors \( p_j \) and \( p_k \) is denoted by \( \nu_{jk} \). 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 \([\bar{\mu} \cdot (1 - HF/2), \bar{\mu} \cdot (1 + HF/2)]\), where \( \bar{\mu} \) is the mean execution rate of the processors. The data transfer rate of each communication link is uniformly distributed in the range \([\bar{\nu} \cdot (1 - HF/2), \bar{\nu} \cdot (1 + HF/2)]\), where \( \bar{\nu} \) 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 \( n_j \) to task \( n_i \). Each node \( n_i \) in a DAG has a weight \( c_i \), 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:

\[
\text{Comp}(n_i, p_j) = w_i / \mu_j
\]

where \( \mu_j \) 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 \( \nu_{ij} \) 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 \( \nu_{ij} \) is incurred when data are transmitted from task \( n_j \) (scheduled on processor \( p_m \)) to task \( n_i \) (scheduled on processor \( p_n \)) and is defined as:

\[
\text{Comm}((n_j, p_m), (n_i, p_n)) = \nu_{ij} / \nu_{mn}
\]

where \( \nu_{mn} \) 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:

\[
CCR = \frac{\sum_{e_{ij} \in E} \text{Comm}(e_{ij})}{\sum_{n_i \in V} \text{Comp}(n_i)}
\]

where \( V \) and \( E \) are the sets of the nodes and the edges of the job respectively. \( \text{Comm}(e_{ij}) \) is the average communication cost of the edge \( e_{ij} \) over all of the communication links in the system, whereas \( \text{Comp}(n_i) \) 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.](image)

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 \( p_a \) is given by:

\[
EST(n_i, p_a) = \max \{ T_{data}(n_i, p_a), T_{delay}(n_i, p_a) \}
\]  

(4)

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

In order to calculate the term \( T_{delay}(n_i, p_a) \), the potential position of task \( n_i \) on processor \( p_a \) 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 \( p_a \), 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:

\[
JGR = \frac{TNJG}{TNJA}
\]

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 $p_j$.
**Output:** Final potential position of task $n_i$ in $p_j$'s queue.

1. find initialPotentialPosition of task $n_i$ in $p_j$'s queue.
2. determine exploitable area of $p_j$'s queue.
3. finalPotentialPosition ← initialPotentialPosition.
4. spareTime ← $-1$.
5. get first task $n_i$ in exploitable area of $p_j$'s queue.
6. repeat
   7. if task $n_i$ forms a schedule hole then
      8. if $\text{Comp}(n_i, p_j) \leq T_{\text{Fmax}}(n_i, p_j)$ then
         9. if Bin Packing Policy = First Fit then
            10. finalPotentialPosition ← currentQueuePosition.
      11. return finalPotentialPosition.
   12. else if Bin Packing Policy = Best Fit then
      13. if spareTime $\leq -1$ or spareTime $>$ spareTimeOfThisScheduleHole then
         14. spareTime ← spareTimeOfThisScheduleHole.
      15. finalPotentialPosition ← currentQueuePosition.
   16. end if
   17. else if Bin Packing Policy = Worst Fit then
      18. if spareTime $< $ spareTimeOfThisScheduleHole then
         19. spareTime ← spareTimeOfThisScheduleHole.
      20. finalPotentialPosition ← currentQueuePosition.
   21. end if
   22. end if
   23. end if
   24. end if
25. get next task $n_i$ in exploitable area of $p_j$'s queue.
26. until all tasks in exploitable area of $p_j$'s queue are examined.
27. return finalPotentialPosition.

The simulation input parameters are summarized in Table 1. The computational volume of a task in a graph is exponential with mean $\lambda$. The communication volume of an edge is exponential with mean $\gamma$. The relative deadline of each job is uniformly distributed in the range $[\text{CPL}, 2\text{CPL}]$, 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 HH $= 0.5$. That is, the target system is considered to feature a moderate degree of heterogeneity.

<table>
<thead>
<tr>
<th>Parameter description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processors in the system</td>
<td>$g = 64$</td>
</tr>
<tr>
<td>Mean execution rate of processors</td>
<td>$\lambda = 1$</td>
</tr>
<tr>
<td>Mean data trans. rate of comm. links</td>
<td>$\gamma = 1$</td>
</tr>
<tr>
<td>Heterogeneity factor</td>
<td>HH $= 0.5$</td>
</tr>
<tr>
<td>Number of tasks in each job</td>
<td>$a \sim U[1, 64]$</td>
</tr>
<tr>
<td>Arrival rate of the jobs</td>
<td>$\lambda \sim [0.2, 0.25, 0.3, 0.35]$</td>
</tr>
<tr>
<td>Relative deadline of each job</td>
<td>RD $\sim U[\text{CPL}, 2\text{CPL}]$</td>
</tr>
<tr>
<td>CCR of the jobs</td>
<td>CCR $\sim [0.1, 1.1, 10]$</td>
</tr>
<tr>
<td>Mean comp. volume of the tasks</td>
<td>$\omega = 10 \text{CCR} - 0.1$ and $\omega = 1 \text{CCR} - (1.10)$</td>
</tr>
</tbody>
</table>

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).

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 2 shows the simulation results in each of the above cases, respectively.

![JGR vs. λ for CCR = 0.1.](image)

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. 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.
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


