Stavrinides, G.L., Rodrigo Duro, F., Karatza, H., García Blas, J., Carretero, J. (2017). Different aspects of workflow scheduling in large-scale distributed systems. *Simulation Modelling Practice and Theory*, 70, pp. 120-134.

DOI: 10.1016/j.simpat.2016.10.009

Different Aspects of Workflow Scheduling in Large-Scale Distributed Systems

Georgios L. Stavrinides[a,*], Francisco Rodrigo Duro[b], Helen D. Karatza[a], Javier Garcia Blas[b], Jesus Carretero[b]

[a] *Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece*
[b] *Computer Architecture and Communication Area, University Carlos III, Avda. Universidad, 30, 28911 Madrid, Spain*

Abstract

As large-scale distributed systems gain momentum, the scheduling of workflow applications with multiple requirements in such computing platforms has become a crucial area of research. In this paper, we investigate the workflow scheduling problem in large-scale distributed systems, from the Quality of Service (QoS) and data locality perspectives. We present a scheduling approach, considering two models of synchronization for the tasks in a workflow application: (a) communication through the network and (b) communication through temporary files. Specifically, we investigate via simulation the performance of a heterogeneous distributed system, where multiple soft real-time workflow applications arrive dynamically. The applications are scheduled under various tardiness bounds, taking into account the communication cost in the first case study and the I/O cost and data locality in the second. The simulation results provide useful insights into the impact of tardiness bound and data locality on the system performance.

Keywords: Workflow scheduling, Large-scale distributed systems, Ultrascale systems, Quality of Service, Data locality

*Corresponding author. Tel.: +30 2310 997974.
Email addresses: gstavrin@csd.auth.gr (Georgios L. Stavrinides), frodrigo@arcos.inf.uc3m.es (Francisco Rodrigo Duro), karatza@csd.auth.gr (Helen D. Karatza), fjblas@inf.uc3m.es (Javier Garcia Blas), jcarrete@inf.uc3m.es (Jesus Carretero)

1. Introduction

The constant progress in computing and communication technologies has led to the emergence of *ultrascale computing*. Ultrascale computing systems are envisioned as large-scale complex systems, joining parallel and distributed computing resources that may be located at multiple sites. The applications in these environments usually concern complex problems and feature coarse-grained parallelism. That is, their component tasks do not require any communication with each other during processing, but only before or after their execution, forming a *workflow*. In order to determine the result of such complex applications, their individual tasks are scheduled and executed coordinately at various nodes of the system, according to their precedence constraints [1].

1.1. Motivation

The scheduling of workflow applications with multiple requirements in such computing platforms has become a crucial area of research. Workflow applications usually have *soft deadlines*, which may be missed by a bounded amount of time (i.e. the applications are *real-time*). Thus, the *Quality of Service (QoS)* of the system often concerns parameters like the completion rate, the timeliness, the makespan and the tardiness of the applications.

Furthermore, workflow applications often require specific data in order to start execution, which may be available at different nodes of the system. Current systems from both high performance computing (HPC) and cloud domains do not efficiently support data-intensive workflows. On the one hand, typical HPC systems use monolithic parallel file systems for data sharing, such as GPFS [2] and Lustre [3]. On the other hand, in order to provide portability and scalable I/O, one of the alternatives for mass data storage in cloud platforms is the use of remote shared storage systems. Usually, the aim of this approach is to provide a unified interface and a scalable storage solution for cloud-based applications through storage services, such as Amazon S3.

Consequently, *QoS* and *data locality* are two important aspects of workflow scheduling in ultrascale systems, that should be taken into account in order to achieve good performance [4, 5].

1.2. Contribution

In this paper, we investigate the workflow scheduling problem in large-scale distributed systems, from the QoS and data locality perspectives. We present a scheduling approach, considering two models of synchronization for the tasks in a workflow application: (a) communication through the network and (b) communication through temporary files. Specifically, we investigate via simulation the performance of a heterogeneous distributed system, where multiple soft real-time workflow applications arrive dynamically. The applications are scheduled under various tardiness bounds, taking into account the communication cost in the first case study and the I/O cost and data locality in the second.

The remainder of the paper is organized as follows: Section 2 gives an overview of related literature. Section 3 describes the QoS objectives during workflow scheduling in ultrascale systems and presents a relevant case study. Section 4 presents an alternative case study, from the data locality perspective, where data awareness is incorporated into the scheduling algorithm. Section 5 presents the performance evaluation of the system from the QoS and data locality perspectives and discusses the insights obtained by the simulation experiments in each case study. Finally, Section 6 summarizes and concludes the paper.

2. Related Work

As the scheduling and execution of complex applications in large-scale distributed systems continues to gather significant attention from the research community, QoS, data locality and workflow-oriented storage systems are some of the most crucial topics.

2.1. Quality of Service

The scheduling problem of complex applications with various constraints and requirements has been studied extensively in the literature [6, 7, 8, 9, 10, 11, 12, 13]. For workflow applications in particular, list scheduling heuristics are the simplest, most practical, easiest to implement and often outperform other scheduling approaches [14]. According to this method, all the component tasks of the applications are prioritized according to a particular common parameter (e.g. deadline, computational cost etc.) and then arranged in a list, ordered according to their priority. Subsequently, each task is allocated to the processor that minimizes a specific cost parameter, such as the estimated start time of the task.

Genez et al. in [15], propose the Integer Linear Program (ILP) policy, in an attempt to solve the problem of scheduling a workflow application in a Software as as Service (SaaS) or Platform as a Service (PaaS) cloud with two levels of service level agreements (SLAs). The first SLA level is between the end-user and the SaaS/ PaaS provider and concerns the deadline within which the user's workflow must be completed. The second SLA level is between the SaaS/ PaaS provider and multiple Infrastructure as a Service (IaaS) providers, concerning the characteristics and pricing of the virtual machines on which the user's workflow will be executed. The ultimate goal of the ILP strategy is to find a feasible mapping between the tasks of the workflow and the virtual machines from multiple IaaS providers, so that the total monetary cost is minimized and the makespan of the workflow is at most equal to the deadline required by the user. It is shown that the proposed policy can provide low-cost solutions, while meeting the deadline of the workflow. However, it only deals with the static scheduling of a single workflow, without utilizing any schedule gaps.

In modern large-scale distributed computing platforms, executing simultaneously multiple workflow applications of different users, sharing the same underlying resources, is an inevitable requirement. Multiple workflow applications have been used in service-oriented and cloud computing environments. For example, the Amazon Simple Workflow Service (SWF) can be used for the cre-

ation and deployment of multiple workflow applications in the Amazon EC2 cloud [16]. Recent research showed that utilizing gaps in the schedule of the compute nodes, formed by the inter-task dependencies and data communication costs, is a promising approach for efficient multiple workflow scheduling [17, 18]. Towards this direction, Jiang et al. in [19] present the Path Clustering Heuristic with Distributed Gap Search (PCH-DGS), for the scheduling of multiple workflow applications in a heterogeneous cloud. According to their proposed strategy, the tasks of a workflow are first partitioned into groups, in an attempt to minimize the communication cost between them. Subsequently, each group of tasks is inserted into the first available time gap in a processor's schedule. In case the gap cannot accommodate all of the tasks of a group, the rest of the group's tasks are inserted into the next available gap in the schedule of the same or other processor of the cloud, in a recursive manner.

In [20], Tsai et al. propose an Adaptive Dual-Criteria task group allocation approach, featuring two mechanisms: (a) an adjustable schedule gap selection and (b) an adaptive task group rearrangement mechanism. First, the tasks in each workflow are clustered into several groups in order to minimize inter-task communication costs. Subsequently, the task groups are prioritized and then allocated onto an appropriate resource, using the proposed allocation approach. Specifically, according to the adjustable schedule gap selection mechanism, a score is calculated for each schedule gap, taking into account the earliest finish time of the task group, the size of the gap and an adjustable parameter, for adjusting the relative weights of the two aforementioned attributes. The gap with the lowest score is considered for selection. Furthermore, according to the adaptive task group rearrangement mechanism, in case a task group does not fit into the gap under consideration, it is split into several subgroups that are allocated during the subsequent allocation steps, taking into account the communication cost between the subgroups. The main disadvantages of the approaches mentioned above, PCH-DGS and Adaptive Dual-Criteria, are: (a) they are static and (b) they are not suitable for workflows with time constraints, a typical requirement of workloads in ultrascale platforms.

2.2. Data Locality

On the one hand, HPC is mostly based on the data processed and generated by scientific applications. Data are typically stored in high-performance parallel file systems, such as Lustre [3] and GPFS [2], for future processing and verification (checkpointing). On the other hand, the analysis of large datasets mainly depends on infrastructures where storage and computation resources are not completely decoupled, as in the case of Hadoop Distributed File System (HDFS) [21].

Data locality is a major factor for reducing data movement and thus execution time, increasing the possibility to meet deadlines. To achieve this objective, several solutions have been proposed. Hadoop [22], uses the MapReduce paradigm [23] to provide a data-aware programming model that facilitates the exploitation of data locality and achieves better scalability. Processes or virtual machines are offloaded over nodes where data have been spread on HDFS to avoid the data transfer required before and after computation. CloneCloud [24] and MAUI [25] are examples of compute offloading in distant fixed clouds. More recent solutions, such as Spark [26] and Tachyon [27], have shown two fundamental issues: first, the importance of in-memory storage and data locality for improving performance in data-intensive applications, and second, the necessity of taking advantage of the new high-speed network technologies in I/O operations.

In any case, Xu and Mao [28] pointed out that the transmission of large data items should occur within a tight user-machine interaction loop. In [29], Wang et al. propose a model called Bottleneck-Aware Allocation (BAA) that provides a new definition of fairness for allocation of multiple resources. BAA is complementary to our solution, given that traffic bottlenecks are not covered in this work.

Finally, another solution that addresses QoS in storage systems is QoSC [30]. The authors propose a data placement policy based on the QoS of the HDFS's DataNodes. A DataNode with high QoS has a heavier weight than one with low QoS, when the DataNode selection policy is enforced. In our proposal, QoS

of a storage node is determined by both cache capacity and access hit ratio.

2.3. Workflow-Oriented Storage Systems

Workflow engines, such as the Data Mining Cloud Framework (DMCF) [31], Swift [32], Pegasus [33], and OmpSs [34], are software systems for designing and executing data analysis workflows. The majority of the workflow engines rely on the default shared storage (parallel file system in HPC infrastructures or the storage service offered by the public cloud provider) for any I/O-related operation. This implies that the I/O performance of tasks is limited by the performance of the default storage and can be greatly hit by contention. Locality-aware techniques are becoming more and more important, in order to avoid these problems. The locality-aware policies presented in our work could be beneficial for workflow engines.

Related with locality-aware storage specifically designed for workflow engines, Costa et al. [35] propose the utilization of the file attributes of MosaStore, in order to provide communication between the workflow engine and the file system, by using hints. The workflow engine can provide these hints directly to the file system or the file system can infer patterns by analyzing data accesses. The MosaStore approach is based on a centralized metadata server that could become a bottleneck in large-scale systems.

The Any-scale Many-task computing File System (AMFS) framework presented in [36], offers programmers a simple scripting language for the execution of parallel applications in memory. We share with AMFS the approach of fully distributing metadata. However, AMFS requires explicit specification of which data are stored in memory and which data will be persistent, while our goal is to be able to transparently provide persistence mechanisms to the programmer.

3. QoS-Driven Workflow Scheduling

As mentioned earlier, one of the major challenges of workflow scheduling in large-scale distributed systems is to meet certain QoS criteria. These may

include the completion rate, the timeliness, the makespan, and the tardiness of the applications. In the following subsections, a particular case study is presented. We investigate via simulation the performance of a heterogeneous distributed system, where multiple soft real-time workflow applications arrive dynamically and are scheduled under various tardiness bounds. The system performance is evaluated in terms of specific QoS metrics.

3.1. Problem Formulation

3.1.1. System Model

In this case study, the target distributed system is considered to consist of a set P of q heterogeneous processors, that are fully connected by a heterogeneous network. Each processor p_i serves its own local queue of tasks and has an execution rate μ_i. The transfer rate between two processors p_i and p_j is denoted by ν_{ij}. The workflow applications arrive at a central scheduler, where their unscheduled tasks wait in a global 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 degree* HD of the system denotes the relative difference in the processing speed of the processors, as well as in the transfer rate of the communication links. The execution rate of each processor is uniformly distributed in the range $[\overline{\mu} \cdot (1 - HD/2), \overline{\mu} \cdot (1 + HD/2)]$, 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 $[\overline{\nu} \cdot (1 - HD/2), \overline{\nu} \cdot (1 + HD/2)]$, where $\overline{\nu}$ is the mean data transfer rate of the communication links. The same heterogeneity degree is used for the calculation of both the execution rate and data transfer rate of each processor and communication link respectively, since most modern distributed systems rely on virtualized resources and thus feature a similar (moderate) degree of heterogeneity regarding their computational and communication resources. The target system is illustrated in Figure 1.

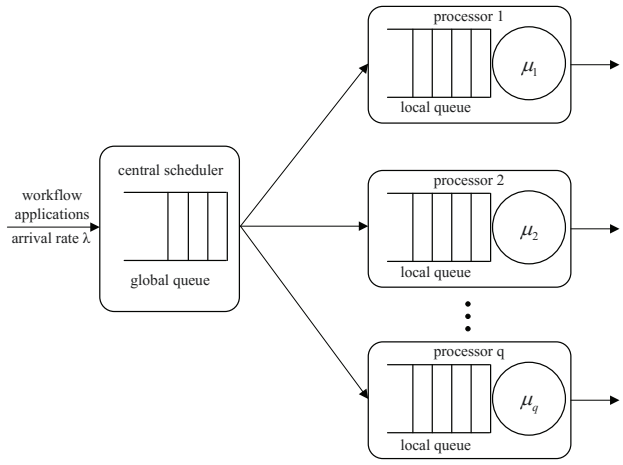

Figure 1: The model of the distributed system under study.

3.1.2. Workload Model

It is assumed that multiple real-time workflow applications arrive dynamically at the system, in a Poisson stream with rate λ [37, 38]. Each application is represented by a *directed acyclic graph (DAG)* $G = (V, E)$, 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 of the application, whereas a directed edge e_{ij} between two tasks n_i and n_j represents the data that must be transmitted from the first task to the other.

A workflow application may have one or more entry tasks and one or more exit tasks. It is assumed that the entry tasks of the application do not require any input data in order to begin and therefore can be processed on any processor. Intermediate tasks, however, require that all of their input data (produced by their parent tasks) are available on their assigned processor, in order to be able to start execution. It is assumed that the component tasks of an application are not preemptible, as preemption of real-time tasks may ultimately lead to performance degradation [39].

Each task n_i in a workflow application 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 defined as:

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

where μ_j is the execution rate of processor p_j.

Each edge e_{ij} between two tasks 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 vm_n) and is given by:

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

where ν_{mn} is the data transfer rate of the communication link between the processors p_m and p_n. In case both tasks n_i and n_j are scheduled on the same processor, the communication cost of the edge e_{ij} is considered negligible. 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. The *critical path length* CPL is the length of the longest path in the graph.

Each workflow application has a *soft end-to-end deadline* D, which can be missed by an amount of time T, called *tardiness* [40]. Tardiness is given by:

$$T = \begin{cases} AFT - D & \text{if } AFT > D \\ 0 & \text{if } AFT \leq D \end{cases} \tag{3}$$

where AFT is the *actual finish time* of the application. The tardiness of an application may be unbounded or bounded. The *tardiness bound* TB of an application is defined as:

$$TB = TF \cdot CPL \tag{4}$$

where TF is a constant, called the *tardiness factor* of the application.

The *makespan* (i.e. schedule length) M of a workflow application is given by:

$$M = AFT - AST \tag{5}$$

where AST is the *actual start time* of the application.

The *communication to computation ratio* CCR of a workflow application is the ratio of its average communication cost to its average computational cost on the target system and is defined as:

$$CCR = \frac{\sum_{e_{ij} \in E} \overline{Comm(e_{ij})}}{\sum_{n_i \in V} \overline{Comp(n_i)}} \tag{6}$$

where V and E are respectively the sets of the tasks and the edges of the application. $\overline{Comm(e_{ij})}$ 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.

It is assumed that the computational volume of a task in a workflow is exponential with mean $\overline{w}$, whereas the communication volume of an edge is exponential with mean $\overline{c}$. The mean communication volume $\overline{c}$ is calculated from (6), for a given CCR and $\overline{w}$. An example of a workflow application represented as a directed acyclic graph is shown in Figure 2.

3.2. Scheduling Approach

In order to schedule the ready tasks in the global queue of the system, a list scheduling heuristic is employed, which consists of two phases: (a) a task selection phase and (b) a processor selection phase. Specifically, we use our proposed algorithm *Earliest Deadline First with Best Fit (EDF_BF)*, as described in [4, 17, 41, 42], since it exhibits promising performance. The key feature of our proposed policy is the incorporation of the *Best Fit* bin packing technique [43, 44] into the processor selection phase, in order to improve the performance of the system, by utilizing possible idle time gaps that may form in the schedule of a processor. The proposed approach could be employed for the scheduling of web services and applications in web and cloud environments,

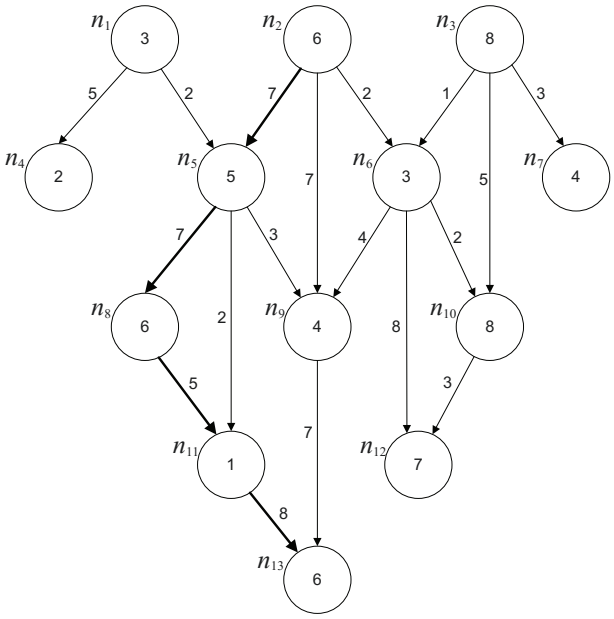

Figure 2: A workflow application represented as a directed acyclic graph with three entry tasks, four exit tasks and six intermediate tasks. The number in each node denotes the average computational cost of the represented task. The number on each edge denotes the average communication cost between the two tasks that it connects. The critical path of the graph is depicted with thick arrows.

where scheduling efficiency and performance are extremely important factors. The EDF_BF scheduling strategy is described below.

3.2.1. Task Selection Phase

The task with the highest priority for scheduling is the one that its application has the earliest end-to-end deadline. That is, the prioritization of the tasks is based on the *Earliest Deadline First (EDF)* policy.

3.2.2. Processor Selection Phase

Subsequently, the scheduler assigns the selected task from the previous step to the processor that can provide it with the earliest *estimated start time EST*,

with any ties broken randomly. The EST of a ready task n_i on a processor p_n is defined as:

$$EST(n_i, p_n) = \max\{t_{data}(n_i, p_n), t_{idle}(n_i, p_n)\} \quad (7)$$

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 $t_{idle}(n_i, p_n)$, we first find the *potential position* of the ready task n_i on processor p_n. This is the position at which the ready task n_i would be placed in the local queue of processor p_n, if it was actually assigned to that particular processor. The potential position of a ready task in a processor's local queue is determined by taking into account the task's priority and by utilizing possible gaps in the processor's schedule, as follows:

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 utilized gap in the schedule of processor p_n. Otherwise, some utilized schedule gaps may be canceled out. The scheduled tasks placed in the area between the head of the queue and the initial potential position of the ready task, form the *searchable area* of the queue.
2. Subsequently, the tasks in the searchable area of the queue are examined whether they can give schedule gaps, starting from the task at the head of the queue. A gap is candidate to be utilized by the ready task n_i only when it can accommodate its computational cost. Moreover, the ready task n_i must not delay the succeeding task n_j in the processor's queue. That is, the following condition must hold:

$$Comp(n_i, p_n) \leq t_{data}(n_j, p_n) - EST(n_i, p_n) \quad (8)$$

In order to place the ready task n_i into a schedule gap, the *Best Fit (BF)* bin packing technique is employed. According to this heuristic, the task

is inserted into the gap where its computational cost fits and where it leaves the minimum unused time possible. The position of a ready task that is inserted into a gap in the processor's schedule is called the *final potential position* of the task. In case a ready task cannot be inserted into a schedule gap, its final potential position is the same as its initial potential position in the processor's queue.

4. Data Locality-Driven Workflow Scheduling

In the previous case study, we considered that the synchronization of the component tasks of a workflow application is achieved through communication via the network. However, as mentioned in the previous sections, the data flow between the intermediate tasks of a workflow application may be based on the exchange of files. In this case study, we extend the former model, assuming that files are used in the input, output and intermediate stages of the workflow, in order to evaluate the impact of using data location information on the performance of workflow scheduling in ultrascale systems. We investigate whether data locality exploitation can reduce the communication times and consequently minimize the average makespan of the workflow applications.

4.1. Utilization of Hercules Distributed In-Memory Storage Solution

In order to exploit data locality, we incorporated into our model the *Hercules* distributed in-memory storage solution, as presented in our previous work in [5]. Hercules provides the following advantages: scalability, ease of deployment, flexibility and performance [45, 46]. Furthermore, it has been proven that Hercules can benefit workflow engines in HPC and cloud environments, by facilitating the exploitation of data locality, independently of the underlying infrastructure [47, 48]. Hercules consists of two levels: a client-side user-level library (Hercules front-end) on top of server-side I/O nodes (Hercules back-end). The applications use the information provided by the Hercules user-level client library, in order to access data on the server-side I/O nodes.

Hercules I/O servers can be deployed using either dedicated or shared resources. In the dedicated resources case, Hercules I/O servers are deployed on dedicated nodes, with the sole objective of providing an alternative I/O solution to the shared file system. In the shared resources case, Hercules I/O servers are deployed onto the compute nodes. Specifically, a Hercules server is deployed on each available compute node, providing better I/O scalability than typical shared file systems, where the number of I/O nodes is statically configured. Moreover, the co-location of the application, Hercules client and Hercules I/O server on the same compute node, enables the possibility of local in-memory data access. This feature, combined with locality-aware schedulers, exposes and exploits data locality.

In this case study, we assume that Hercules is utilized according to the shared resources approach, where a Hercules I/O server is deployed on each compute node. When a task of a workflow application is scheduled for execution on a compute node of the distributed system, it may access its required data through the Hercules client on the local Hercules I/O server, in case they are locally available. Otherwise, it may access the data through the Hercules client on a remote Hercules I/O server (residing on another compute node), where they are available. Alternatively, in case the data are not available in the Hercules storage system, the task can access them from the shared parallel file system (GPFS). This is depicted in Figure 3.

In order to show how Hercules interacts with the workflow engine, we have adapted the workflow application model so that the communication between the tasks is achieved through temporary files. Specifically, for the example application shown in Figure 2, we consider that the three entry tasks read one file each, the four exit tasks write one resulting file each and the communication to/from the intermediary tasks is performed by write/read operations over intermediate files. The six intermediary tasks must wait to read a file created by their parent tasks. After their execution, they must write a file for communicating the results to their child tasks. This is depicted in Figure 4. As in the previous model, the number in each node denotes the average computational cost of the

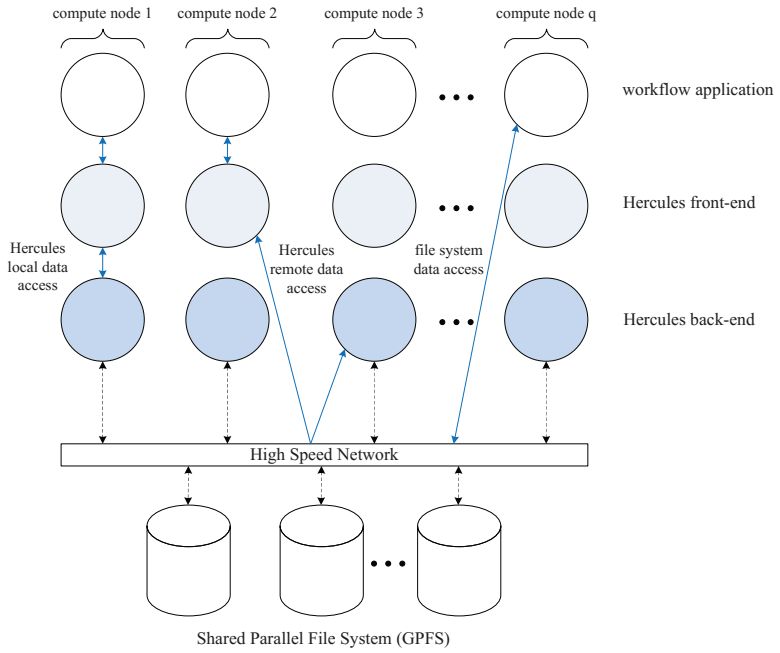

Figure 3: The model of the Hercules distributed in-memory storage solution.

represented task. However, the number on each edge now denotes the average I/O cost between the tasks that it connects, depending on the file size.

In this model, the communication link in the system is replaced by temporary files, which could be stored in Hercules local back-ends, Hercules remote back-ends and the underlying parallel file system (i.e. OrangeFS, Lustre or GPFS). The communication cost of the edge e_{ij} between two tasks n_i and n_j is substituted by the I/O cost of creating/accessing the temporary files, which highly depends on where the data are stored. Given the single path and single copy characteristics of Hercules, data can only be stored in one level of the file system. H_{ij} represents the probability of finding a file generated by task n_i

and required by the child task n_j, in the Hercules subsystem. Every file not stored in Hercules will be transferred to/from GPFS, with an inverse probability of $1 - H_{ij}$. Assuming a file system with three levels (local Hercules, remote Hercules and shared file system), each level will expose a different hit ratio, depending on the effectiveness of each server at providing data locality services. This cost is denoted by:

$$IO\left((n_i, p_m), (n_j, p_n)\right) = H_{ij} \cdot (IO_{local} + IO_{remote}) + (1 - H_{ij}) \cdot IO_{shared} \quad (9)$$

Based on this general equation, we can further describe each of the previous I/O-related costs. Hercules costs will be especially related to the hit ratio of the local level and the remote level, which are represented by hl_{ij} and hr_{ij}, respectively. The costs related with Hercules communications can be modeled as:

$$IO_{local} = hl_{ij} \cdot (f_{ij}/bm_{mn}) \quad (10)$$

$$IO_{remote} = hr_{ij} \cdot (f_{ij}/bn_{mn}) \quad (11)$$

where hl_{ij} is the local hit ratio, reflecting the probability of finding a file generated by task n_i, which was executed on processor p_m, on the processor p_n that the subsequent task n_j is going to be executed. hr_{ij} is the remote hit ratio, representing the probability of finding the file on any other processor. bm_{mn} is the bandwidth of the local accesses to the in-memory storage system, whereas bn_{mn} is the bandwidth of the remote accesses to the in-memory storage system, performed through the fully-connected network. f_{ij} is the file size, representing the amount of data transferred from processor p_m to processor p_n. Additionally, as stated before, based on the single path and single copy characteristics of the Hercules subsystem, each data item can only be stored in one of the hierarchy levels, which implies that:

$$hr_{ij} = 1 - hl_{ij} \quad (12)$$

Finally, the cost of accessing file data in the shared file system can be de-

scribed as:

$$IO_{shared} = f_{ij}/\rho_{mn} \tag{13}$$

where ρ_{mn} corresponds to the throughput provided by the shared file system. It is directly affected by the contention problem, depending on the number N of tasks concurrently accessing the storage subsystem:

$$\rho_{mn} = MAX\rho_{mn}/(1 - H_{ij}) \cdot N \tag{14}$$

where $MAX\rho_{mn}$ is the maximum theoretical throughput offered by the shared file system in perfect conditions. It is degraded by the file system contention.

Based on the modification of the communication concept into file access operations, instead of the CCR parameter of a workflow application, in this case study we employ the notion of the *I/O to computation ratio IOCR* of a workflow application, which is the ratio of its average storage cost to its average computational cost on the target system. It is defined as:

$$IOCR = \frac{\sum_{e_{ij} \in E} \overline{IO(e_{ij})}}{\sum_{n_i \in V} \overline{Comp(n_i)}} \tag{15}$$

where $\overline{IO(e_{ij})}$ is the average I/O cost between tasks n_i and n_j, whereas $\overline{Comp(n_i)}$ represents the average computational cost of the task n_i over all of the processors in the system.

As shown in Figure 4, the workflow application previously depicted in Figure 2, is modified so that the data dependencies between the tasks are now represented as file accesses. In the Figure, the communication costs are now replaced by the corresponding I/O costs.

4.2. Incorporation of Data-Awareness into the Scheduling Algorithm

In order to schedule the ready tasks in the global queue of the system, as in the previous case study, a list scheduling heuristic is employed, which consists of two phases: (a) a task selection phase and (b) a processor selection phase. Again, the EDF_BF algorithm is utilized. However, in this case study, data-locality criteria are incorporated into the processor selection phase, in order to

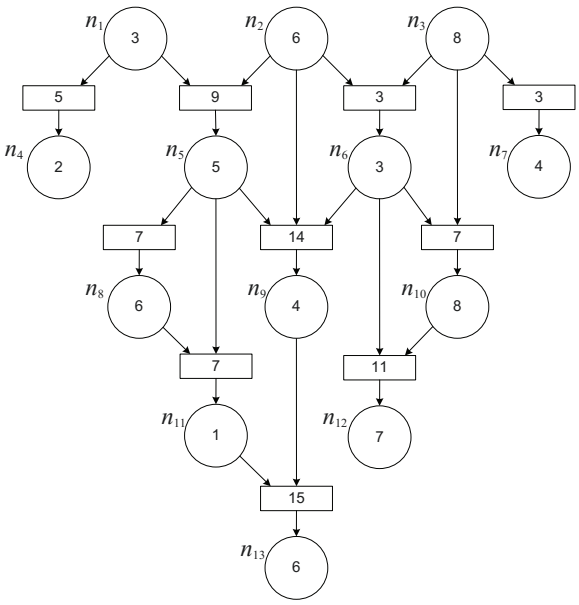

Figure 4: The application DAG including file dependencies. Files can be stored in a traditional shared file system, a Hercules I/O solution deployment or a combination of both.

avoid data movement between processors, as well as between a processor and the global file system.

4.2.1. Task Selection Phase

As previously stated, the task with the highest priority for scheduling is the one that its application has the earliest end-to-end deadline. Thus, the prioritization of the tasks is based on the EDF policy.

4.2.2. Processor Selection Phase

As in the previous case study, the execution of a data-dependent task whose data are not available, is delayed until all of the data dependencies are fulfilled. In this model, the scheduler works in combination with Hercules in order to select the resources for each task. Specifically, the scheduler takes into account the data location information in order to assign the selected task to a proces-

sor. The data are made available locally or remotely by Hercules. Whenever Hercules retrieves the data from a parent task, it moves them to the child task (if necessary) and notifies it about the availability. A task can be placed on the data node or as near as possible to the data dependency. This mechanism can be used for various processor allocation policies, which move the task execution where the data are available, provided that Hercules returns the information about data placement.

The EST of a ready task n_i on a processor p_n is defined again as:

$$EST(n_i, p_n) = \max\{t_{data}(n_i, p_n), t_{idle}(n_i, p_n)\} \tag{16}$$

where $t_{data}(n_i, p_n)$ is the time at which all data input files 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. As in the previous case study, the term $t_{idle}(n_i, p_n)$ is calculated by utilizing schedule gaps with the BF policy. However, now $t_{data}(n_i, p_n)$ changes depending on the service model for the data files: Hercules local data access, Hercules remote data access and shared global file system, as shown in Figure 3.

5. Performance Evaluation

The system performance was evaluated via simulation, using specific metrics. First, the framework under study was evaluated from the QoS perspective. Subsequently, the data-locality features were incorporated into the framework and the system performance was evaluated from the data locality perspective.

5.1. Evaluation Metrics

The performance of the system was evaluated in terms of the following QoS metrics:

- *Application completion ratio* (ACR), which is given by:

$$ACR = \frac{TNCA}{TNAA} \tag{17}$$

where $TNCA$ is the *total number of completed applications*, i.e. the total number of workflow applications that completed execution either within or beyond their deadline. $TNAA$ is the *total number of application arrivals* at the system, during the observed time period.

- *Application guarantee ratio* (AGR), which defined as:

$$AGR = \frac{TNGA}{TNAA} \tag{18}$$

where $TNGA$ is the *total number of guaranteed applications*, i.e. the total number of workflow applications that completed execution within their deadline, without exhibiting any tardiness.

- *Average makespan* ($\overline{M}$), which is the average makespan of all of the completed workflow applications.

- *Average tardiness* ($\overline{T}$), which is the average tardiness of all of the completed workflow applications.

5.2. Experimental Setup

The performance of the framework under study was evaluated via simulation. Due to the complexity of the system and the workload models, we implemented our own discrete-event simulation program in C++, tailored to the requirements and specifications of the particular framework. Furthermore, in order to obtain unbiased simulation results, no specific workload traces were used, but synthetic workload was used instead. Specifically, we used our own random DAG generator, as described in [49].

We conducted a series of simulation runs, using the independent replications method. Specifically, we ran 30 replications of the simulation program for each set of input parameters, with different seeds of random numbers. The termination condition of each replication was the completion of 10^6 workflow applications. For every mean value, a 95% confidence interval was evaluated.

Table 1: Simulation Input Parameters.

Parameter	**Value**
Number of completed DAGs	10^6
Number of processors	$q = 64$
Scheduling method	EDF_BF
Mean processor execution rate	$\overline{\mu} = 1$
Mean link transfer rate	$\overline{\nu} = 1$
Mean file system throughput rate (data locality case)	$\overline{\rho} = 1$
System heterogeneity degree	$HD = 0.5$
Max number of tasks per DAG	$X = 64$
Number of tasks per DAG	$a \sim U[1, X]$
DAG arrival rate	$\lambda = \{1, 1.25, 1.5, 1.75\}$
DAG relative deadline	$RD \sim U[CPL, 2CPL]$
Communication to computation ratio	$CCR = 1$
I/O to computation ratio (data locality case)	$IOCR = \{0.25, 0.5, 0.75, 1\}$
Mean task computational volume	$\overline{w} = 1$
DAG tardiness factor	$TF = 0.2$
DAG tardiness bound	$TB = \{0, 0.2CPL, unbounded\}$

The half-widths of all of the confidence intervals are less than 5% of their respective mean values. The simulation input parameters used in our experiments are shown in Table 1.

The performance of the system, with respect to the arrival rate λ of the workflow applications, was evaluated under different bounds of tardiness. Specifically, we investigated the effects of the following tardiness bounds TB on the performance parameters: (a) the case where no tardiness is allowed for each workflow application (i.e. $TB = 0$), which is essentially equivalent to the case of hard deadlines, (b) the case where the allowed tardiness for each workflow application is bounded by a value proportional to its critical path length (i.e. $TB = TF \cdot CPL$) and (c) the case where the maximum allowed tardiness is unbounded. The simulation results are analyzed in the following subsection.

The features of the storage system are characterized by the mean file system throughput rate ($\overline{\rho}$) and the I/O to computation ratio ($IOCR$) parameters. We considered different scenarios, in order to investigate the effect of data locality

using Hercules. For example, in the case of lack of data locality, $IOCR$ is equal to 1, which means that all files are forwarded to the infrastructure file system. A $IOCR = 0.25$ corresponds to a scenario where maximum data locality is achieved.

5.3. QoS Simulation Results Analysis

The performance of the system in terms of the application completion ratio ACR metric, is shown in Figure 5. It can be observed that in the case where the maximum allowed tardiness of the applications is unbounded, the ACR is always almost equal to 1, for all workload conditions. This is because, without any tardiness bound, all of the applications are allowed to complete their execution, without any time constraints. On the other hand, when the tardiness bound is $TB = 0$, which essentially means that the applications have to finish their execution within their deadline, because otherwise they are lost, the ACR is low. When the tardiness is bounded by a specific amount of time ($TB = 0.2CPL$), the ACR has a value between the two extreme cases.

Figure 6 shows the QoS level of the system in terms of the application guarantee ratio AGR metric. It can be observed that the AGR decreases as the tardiness bound increases. Since the computational and communication volumes of each application are exponentially distributed, they more often have small values and sometimes very large values. That is, a large percentage of the applications that arrive at the system exhibit moderate computational and communication requirements, whereas there is a small percentage of them that are computationally and communication intensive. Therefore, as the tardiness bound increases, a larger number of the few computationally and communication intensive applications is allowed to complete its execution, delaying the other applications in the system, leading to more deadline misses.

For the same reason, the average makespan $\overline{M}$ and the average tardiness $\overline{T}$ of the completed applications increase as the tardiness bound increases. This can be observed in Figures 7 and 8, respectively. It can also be observed that with the increase of the arrival rate λ of the applications, all of the QoS metrics

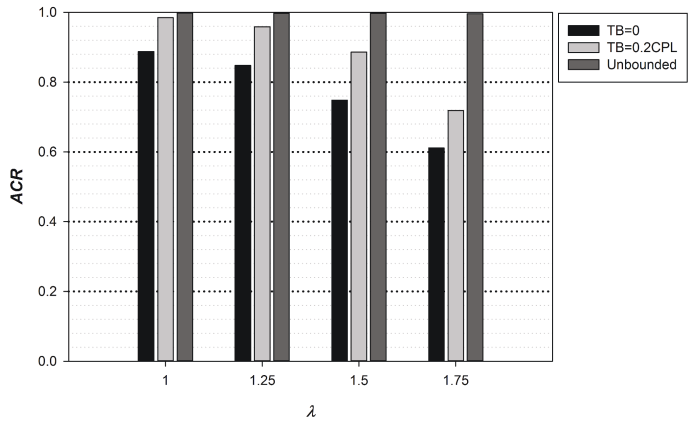

Figure 5: The application completion ratio (ACR) vs. the arrival rate λ, for tardiness bound $TB = 0$, $TB = 0.2CPL$ and for unbounded tardiness.

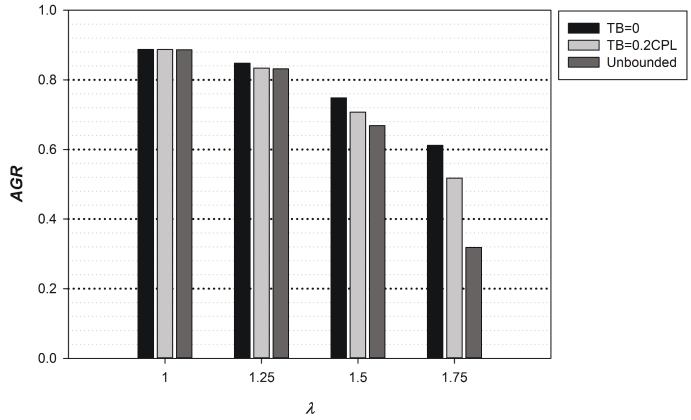

Figure 6: The application guarantee ratio (AGR) vs. the arrival rate λ, for tardiness bound $TB = 0$, $TB = 0.2CPL$ and for unbounded tardiness.

exhibit poorer performance. This is due to the heavier workload, which has a negative impact on the system performance.

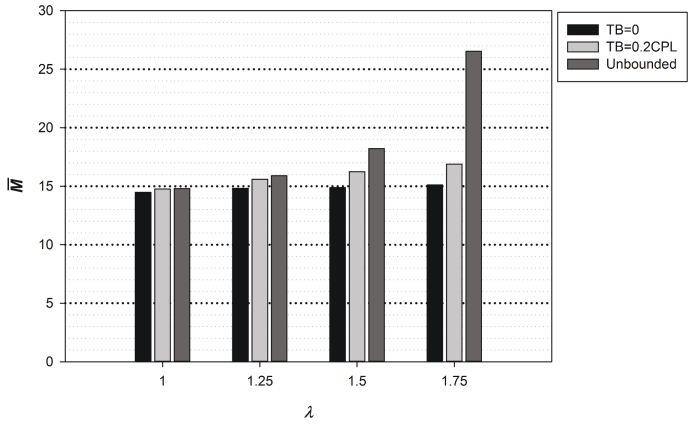

Figure 7: The average makespan ($\overline{M}$) of the completed applications vs. the arrival rate λ, for tardiness bound $TB = 0$, $TB = 0.2CPL$ and for unbounded tardiness.

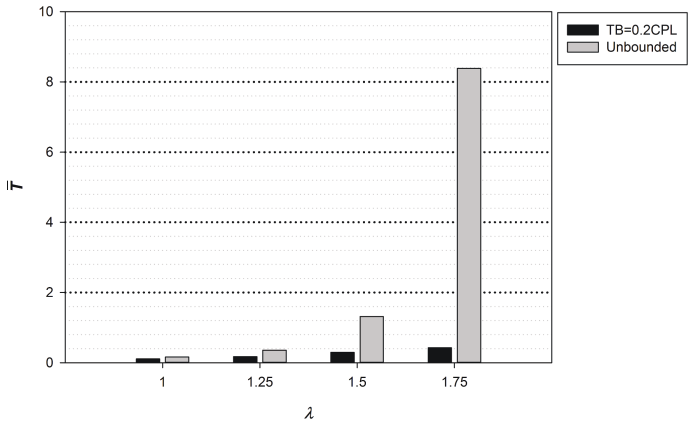

Figure 8: The average tardiness ($\overline{T}$) of the completed applications vs. the arrival rate λ, for tardiness bound $TB = 0.2CPL$ and for unbounded tardiness.

5.4. Data Locality Simulation Results Analysis

In order to investigate how data locality affects the performance of the system, we simulated different scenarios with workflows featuring various data locality characteristics. As a starting point, Figure 9 shows how the quantity

Table 2: IOCR-Data Locality Relationship Input Parameters.

Parameter	**Value**
Memory bandwidth	3500 MB/s
Average Hercules remote throughput	100 MB/s
Average file data volume	100 MB
Max. shared file system throughput	2000 MB/s
Average number of concurrent tasks	100

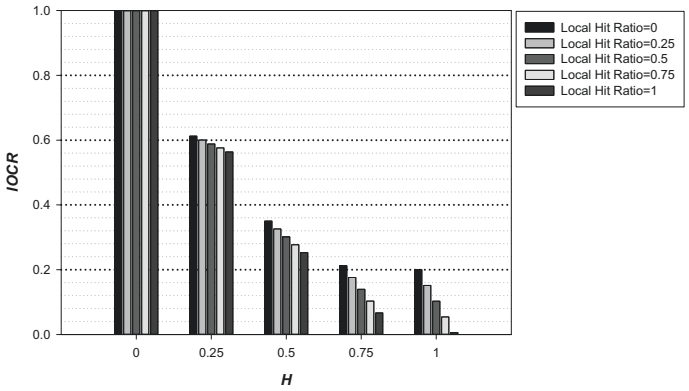

Figure 9: The dependence of I/O to computation ratio ($IOCR$) on the data locality behavior of the tasks and the percentage of data available in the Hercules subsystem.

of data available in Hercules (H) and the data locality behavior of the workflow (hit ratio of Hercules local accesses, hl) affects the I/O to computation ratio (IOCR), based on the parameters of Table 2. As can be seen in Figure 9, for an increasing ratio of available data in the Hercules subsystem, the shared file system contention is greatly reduced, significantly reducing IOCR. Local data access benefits the I/O performance of the system, reducing the I/O cost, especially when more data are available in the Hercules file system.

Figure 10 shows the performance of the system in terms of the average makespan $\overline{M}$ for arrival rate $\lambda = 1.75$ and unbounded tardiness, for workflow applications where the dependencies of their tasks are solved using files, as shown in Figure 4. We simulated scenarios with different IOCR values, in order to show

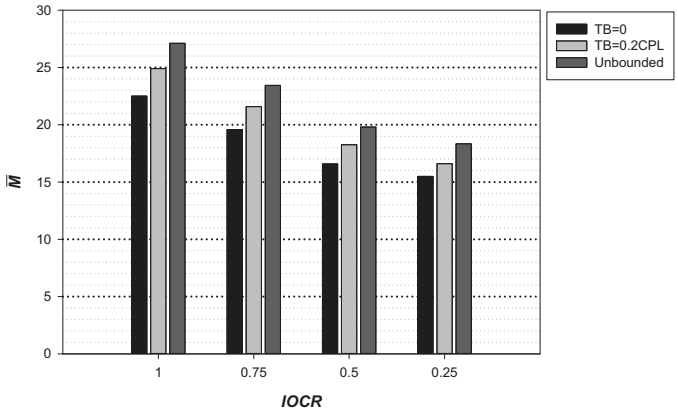

Figure 10: The average makespan ($\overline{M}$) of the completed applications vs. the I/O to computation ratio ($IOCR$), for arrival rate $\lambda = 1.75$ and unbounded tardiness.

how different data locality behaviors affect the makespan of the workflows. We chose the maximum arrival rate used in previous simulations, as well as an unbounded tardiness, in order to stress the file system, as a high arrival rate and an unbounded tardiness mean that more files are created and thus there is more contention in the file system. As shown in Figure 10, even for the same arrival rate, the average makespan $\overline{M}$ changes, depending on the data locality of the temporary files created to facilitate the communication between the tasks. In case only the shared file system is used ($IOCR = 1$), all files must be created, written, and read from it, increasing both metadata and data operations in the storage devices. Thus, the makespan metric exhibits poorer performance. To evaluate the effect of Hercules, we have simulated three scenarios, providing: (a) low data locality ($IOCR = 0.75$), (b) moderate data locality ($IOCR = 0.5$) and (c) high data locality ($IOCR = 0.25$) . As we increase data locality, the cost of data movement operations is lower, since more data are copied using local or remote memory from Hercules servers. The average makespan is decreased by an average of 48% between the two data locality extreme cases ($IOCR = 1$ and $IOCR = 0.25$).

6. Conclusions

In this paper, we investigated the workflow scheduling problem in large-scale distributed systems, from both the QoS and data locality perspectives. We considered two case studies, where the communication between the component tasks of a workflow is achieved through the network in the first and through temporary files in the second, utilizing the Hercules distributed in-memory storage solution. A system and workload model have been formulated, characterizing the major properties under consideration. Specifically, we investigated by simulation the performance of a heterogeneous distributed system, where multiple soft real-time workflow applications arrive dynamically. The applications were scheduled taking into account the communication cost in the first case study and the I/O cost and data locality in the second. The system performance was evaluated in terms of specific metrics.

In the first case study, the simulation results show that the value of the tardiness bound has a different impact on the various performance parameters. Specifically, the application completion ratio (ACR) improves as the tardiness bound increases, whereas the application guarantee ratio (AGR) and the average makespan ($\overline{M}$) deteriorate. In the second case study, the results of the simulation experiments show that the data locality behavior of the tasks considerably affects the system performance. Specifically, the average makespan ($\overline{M}$) of the completed applications improves as the I/O to computation ratio ($IOCR$) decreases. That is, the utilization of Hercules is beneficial to the system performance, due to the data locality techniques provided by our solution.

Acknowledgments

The work presented in this paper has been partially supported by EU, under the COST program Action IC1305, *"Network for Sustainable Ultrascale Computing (NESUS)"*, and by the the Ministerio de Economía y Competitividad, Spain, under the project TIN2013-41350-P, *"Scalable Data Management Techniques for High-End Computing Systems"*.

[1] G. L. Stavrinides, 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), 2011, pp. 273–287. doi:10.1007/978-3-642-21713-5_20.

[2] F. B. Schmuck, R. L. Haskin, Gpfs: A shared-disk file system for large computing clusters, in: Proceedings of the Conference on File and Storage Technologies (FAST'02), 2002, pp. 231–244.

[3] P. J. Braam, The Lustre storage architecture, Cluster File Systems, Inc., 2004. URL: http://www.lustre.org/documentation.html.

[4] G. L. Stavrinides, H. D. Karatza, Scheduling real-time jobs in distributed systems - simulation and performance analysis, in: Proceedings of the 1st International Workshop on Sustainable Ultrascale Computing Systems (NESUS'14), 2014, pp. 13–18.

[5] F. R. Duro, J. G. Blas, F. Isaila, J. M. Wozniak, J. Carretero, R. Ross, Flexible data-aware scheduling for workflows over an in-memory object store, in: Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'16), 2016, pp. 321–324. doi:10.1109/CCGrid.2016.40.

[6] G. L. Stavrinides, 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), 2008, pp. 1–7.

[7] G. L. Stavrinides, 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 85 (2009) 525–536.

[8] H. Yu, Y. Ha, B. Veeravalli, Quality-driven dynamic scheduling for real-time adaptive applications on multiprocessor systems, IEEE Transactions on Computers 62 (2013) 2026–2040.

[9] H. Arabnejad, J. G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Transactions on Parallel and Distributed Systems 25 (2014) 682–694.

[10] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, Future Generation Computer Systems 48 (2015) 1–18.

[11] G. L. Stavrinides, H. D. Karatza, Scheduling different types of applications in a saas cloud, in: Proceedings of the 6th International Symposium on Business Modeling and Software Design (BMSD'16), 2016, pp. 144–151.

[12] G. L. Stavrinides, H. D. Karatza, Scheduling real-time parallel applications in saas clouds in the presence of transient software failures, in: Proceedings of the 2016 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'16), 2016, pp. 1–8. doi:10.1109/SPECTS.2016.7570524.

[13] L. Liu, M. Zhang, R. Buyya, Q. Fan, Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing, Concurrency and Computation: Practice and Experience (2016) in press.

[14] H. Topcuoglu, S. Hariri, M. Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (2002) 260–274.

[15] T. A. L. Genez, L. F. Bittencourt, E. R. M. Madeira, Workflow scheduling for saas/ paas cloud providers considering two sla levels, in: Proceedings of the 2012 IEEE Network Operations and Management Symposium (NOMS'12), 2012, pp. 906–912. doi:10.1109/NOMS.2012.6212007.

[16] Y. Chen, W. T. Tsai, Service-Oriented Computing and Web Software Integration: From Principles to Development, 5th ed., Kendall Hunt Publishing, 2015.

[17] G. L. Stavrinides, 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), 2014, pp. 65:1–65:8. doi:10.1145/2674396.2674469.

[18] G. L. Stavrinides, H. D. Karatza, A cost-effective and qos-aware approach to scheduling real-time workflow applications in paas and saas clouds, in: Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud'15), 2015, pp. 231–239. doi:10.1109/FiCloud.2015.93.

[19] H. J. Jiang, K. C. Huang, H. Y. Chang, D. S. Gu, 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), 2011, pp. 282–293. doi:10.1007/978-3-642-24650-0_24.

[20] Y. L. Tsai, H. C. Liu, K. C. Huang, Adaptive dual-criteria task group allocation for clustering-based multi-workflow scheduling on parallel computing platform, The Journal of Supercomputing 71 (2015) 3811–3831.

[21] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST'10), 2010, pp. 1–10. doi:10.1109/MSST.2010.5496972.

[22] T. White, Hadoop: The Definitive Guide, 3rd ed., O'Reilly Media, 2012.

[23] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (2008) 107–113.

[24] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: Proceedings of the Sixth Conference on Computer Systems (EuroSys'11), 2011, pp. 301–314. doi:10.1145/1966445.1966473.

[25] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10), 2010, pp. 49–62. doi:10.1145/1814433.1814441.

[26] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12), 2012, pp. 2–2.

[27] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, I. Stoica, Reliable, memory speed storage for cluster computing frameworks, in: Proceedings of the 2014 ACM Symposium on Cloud Computing (SOCC'14), 2014, pp. 6:1–6:15. doi:10.1145/2670979.2670985.

[28] Y. Xu, S. Mao, A survey of mobile cloud computing for rich media applications, IEEE Wireless Communications 20 (2013) 46–53.

[29] H. Wang, Resource Allocation Models for Multi-Tiered Storage: Balancing System Efficiency and QoS, Ph.D. thesis, Rice University, Houston, TX, 2015. URL: https://scholarship.rice.edu/handle/1911/88385.

[30] B. Yang, G. Song, Y. Chen, Y. Zheng, Y. Wu, Qos-aware indiscriminate volume storage cloud, Concurrency and Computation: Practice and Experience (2016).

[31] F. Marozzo, D. Talia, P. Trunfio, Js4cloud: script-based workflow programming for scalable data analysis on cloud platforms, Concurrency and Computation: Practice and Experience 27 (2015) 5214–5237.

[32] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, I. Foster, Swift: A language for distributed parallel scripting, Parallel Computing 37 (2011) 633–652.

[33] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny, K. Wenger, Pegasus, a workflow management system for science automation, Future Generation Computer Systems 46 (2015) 17–35.

[34] J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. M. Badia, E. Ayguade, J. Labarta, Productive cluster programming with ompss, in: Proceedings of the 17th International European Conference on Parallel and Distributed Computing, (Euro-Par'11), 2011, pp. 555–566. doi:10.1007/978-3-642-23400-2_52.

[35] L. B. Costa, H. Yang, E. Vairavanathan, A. Barros, K. Maheshwari, G. Fedak, D. Katz, M. Wilde, M. Ripeanu, S. Al-Kiswany, The case for workflow-aware storage: an opportunity study, Journal of Grid Computing 13 (2015) 95–113.

[36] Z. Zhang, D. S. Katz, T. G. Armstrong, J. M. Wozniak, I. Foster, Parallelizing the execution of sequential scripts, in: Proceedings of the 2013 International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13), 2013, pp. 31:1–31:12. doi:10.1145/2503210.2503222.

[37] H. D. Karatza, Performance of gang scheduling strategies in a parallel system, Simulation Modelling Practice and Theory 17 (2009) 430–441.

[38] H. K. Tang, P. Ramanathan, K. Morrow, Inserting placeholder slack to improve run-time scheduling of non-preemptible real-time tasks in hetero-

geneous systems, in: Proceedings of the 27th International Conference on VLSI Design and 13th International Conference on Embedded Systems 2014, 2014, pp. 168–173. doi:10.1109/VLSID.2014.36.

[39] G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 3rd ed., Springer, 2011. doi:10.1007/978-1-4614-0676-1.

[40] J. A. Stankovic, M. Spuri, K. Ramamritham, G. C. Buttazzo, Deadline scheduling for real-time systems: EDF and related algorithms, Springer Science & Business Media, 1998. doi:10.1007/978-1-4615-5535-3.

[41] G. L. Stavrinides, 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 19 (2011) 540–552.

[42] G. L. Stavrinides, 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 28 (2012) 977–988.

[43] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: survey and classification, Springer, 2013, pp. 455–531. doi:10.1007/978-1-4419-7997-1_35.

[44] R. Lewis, X. Song, K. Dowsland, J. Thompson, An investigation into two bin packing problems with ordering and orientation implications, European Journal of Operational Research 213 (2011) 52–65.

[45] B. Fitzpatrick, Distributed caching with memcached, Linux Journal 2004 (2004) 5–.

[46] J. L. Carlson, Redis in Action, Manning Publications Co., 2013.

[47] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, I. T. Foster, Swift/t: Large-scale application composition via distributed-memory dataflow processing, in: Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'13), 2013, pp. 95–102. doi:10.1109/CCGrid.2013.99.

[48] F. R. Duro, F. Marozzo, J. G. Blas, J. Carretero, D. Talia, P. Trunfio, Evaluating data caching techniques in dmcf workflows using hercules, in: Proceedings of the 2nd International Workshop on Sustainable Ultrascale Computing Systems (NESUS'15), 2015, pp. 95–106.

[49] G. L. Stavrinides, 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 83 (2010) 1004–1014.