A Data-Aware MultiWorkflow Cluster Scheduler

César Acevedo, Porfidio Hernández, Antonio Espinosa and Victor Méndez
Computer Architecture and Operating System, Univ. Autónoma de Barcelona, Barcelona, Spain

Keywords: Multiple, Multiworkflows, Data-Aware, Cluster.

Abstract: Previous scheduling research work is based on the analysis of the computational time of application workflows. Current use of clusters deals with the execution of multiworkflows that may share applications and input files. In order to reduce the makespan of such multiworkflows adequate data allocation policies should be applied to reduce input data latency. We propose a scheduling strategy for multiworkflows that considers the data location of shared input files in different locations of the storage system of the cluster. For that, we first merge all workflows in a study and evaluate the global design pattern obtained. Then, we apply a classic list scheduling heuristic considering the location of the input files in the storage system to reduce the communication overhead of the applications. We have evaluated our proposal with an initial set of experimental environments showing promising results of up to 20% makespan improvement.

1 INTRODUCTION

Scheduling a workflow with precedence constraints is an important problem in scheduling theory and has been shown to be NP-Hard (Pinedo, 2012). There are many studies on how to schedule a single workflow, specially when trying to schedule tasks onto heterogeneous domains (Topcuoglu et al., 1999). There is an increasing interest in executing several workflows simultaneously. The problem of defining which application from the multiple workflows is going to be executed in a specific node of a cluster, has been described in several works like (Bittencourt and Madeira, 2010), (Stavrinides and Karatza, 2011), (Zhao and Sakellariou, 2006), (Hönig and Schiffmann, 2006), (Yu and Shi, 2008), (N'Takpé and Suter, 2009), (Afrati et al., 1988), (Rahman et al., 2007), (Gu and Wu, 2010).

Current scientific applications must deal with large data sets, usually demanding large amounts of computation and communication times. In many cases, systems undergo undesired idle times between data transfers and application execution (Stavrinides and Karatza, 2011). Our proposal is to make use of these idle times between scheduled tasks with bin packing techniques with a list scheduling approach. Workflow-aware storage is the key to select the location of the data as relevant criteria for the application scheduling. This idea has previously been used to reduce the I/O load of a cloud system (Wang et al., 2011).

We present on figure 1 a simple scenario where a scheduler needs to decide for one workflow the need of locating input, temporal and output data files on the hierarchical storage system. If we extend the problem to multiworkflows to reduce the data access latency, we need to resolve some issues. If an input or temporal files are going to be read several times by many applications at a given time and the cost of reading once from a distributed file system and writing on a local disk or local memory of the computing node is low; then we need to store locally these data files previous to the execution. We will need to manage files when a local storage is full and define where to locate output files if they are final results on to distributed file system to avoid using critical space on smaller storage systems (Costa et al., 2015).

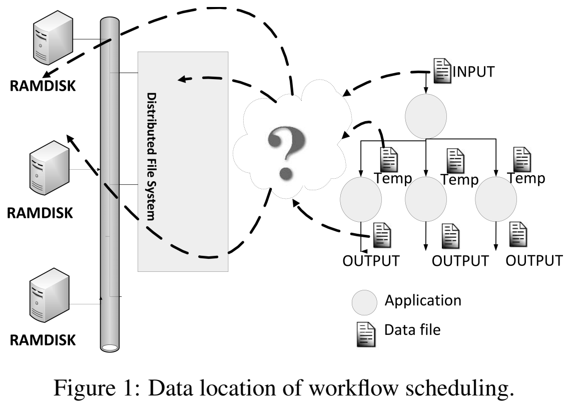

Figure 1: Data location of workflow scheduling.

Executing multiples workflows exposes another problem for distributed file systems like Lustre. Traditionally it is difficult to predict performance on shared High Performance Computing environments (Zhao and Hu, 2010). Applying special techniques on access optimization (Meswani et al., 2010) is also complicated. Therefore we have decided to use ”share nothing” nodes or storage hierarchies where predictability index is better such as local file systems or Ramdisk.

Due to the peculiarity of bioinformatics workflows that are composed of many applications that share files, we can take advantage of reading a cached file. An extension of the classic model of execution and mapping of workflow application of this kind of architecture to a Shared Input File policy is proposed. The amount of data needs to be taken into account on the specifications of the abstract graph of the workflow to exploit hierarchical storage systems.

In this paper, we propose a scheduling algorithm for multiworkflows in a cluster environment.

We propose to evaluate the rendering pattern of the workflow and analyze a list of attributes from it. We are considering input and output data size and location, branch and depth factors for each workflow. Then we apply a storage-aware mechanism to reduce the cost of I/O operations of the system. For that, we analyze shared input files in the storage hierarchy and merge workflows in a study into a new meta-workflow. We apply critical path analysis to reduce the makespan of multiworkflows.

This approach has been evaluated with an initial set of experimental environments. With classic List Scheduling with NFS as the storage system and List Scheduling for data-aware multiworkflows using a local disk. Another experimental scenario with List Scheduling and local disk and ramdisk for the data-aware approach. The last experiment with shared input file of 2 files with sizes of 2048Mb, 1024Mb, and 512Mb. The promising initial results show that our scheduler improves makespan in up to 20%.

The rest of the paper is organized as follows. Related work is discussed in Section II. Then we describe the scheduler architecture in Section III. Section IV elaborates the experiment design and evaluates the performance of proposed algorithm along with other versions. Finally, we summarize and lay out the future work in Section V.

2 RELATED WORK

Prior research focuses on scheduling heuristics for single workflow (Kwok and Ahmad, 1999) provides the taxonomy of approaches that classifies static algorithms according to functionalities. Clustering approaches reduce communication costs by grouping applications with high communication cost into a cluster (Yang and Gerasoulis, 1994). Duplication heuristic (Park et al., 1997) provides a method for an application to transmit data to succeeding applications creating duplicate applications on the destination compute node or processor. Level based method divides workflows into independent applications levels and then apply the map of applications to resources (Mandal et al., 2005).

List scheduling algorithms (Ilavarasan and Thambidurai, 2007) try to implement low complexity heuristics for heterogeneous environments. Other alternatives (Bolze et al.,) minimize the makespan of the workflow, providing a sorted list of applications according to their prioritization such as Critical Path (Topcuoglu et al., 1999).

A method to schedule multiworkflow (Barbosa and Monteiro, 2008) uses list scheduling heuristic maximizing the resource usage by assigning a variable amount of resources to an application instead of a fixed set of resources. Also, (Bittencourt and Madeira, 2010) provides a path clustering heuristic with 4 different application orders to be submitted, like sequential, gap search, interleave and graph groups methods.

However, for list scheduling, not much research has been done that takes into account that many applications have evolved from compute-intensive to data-intensive. Neither workflow-aware scheduling as (Costa et al., 2015) that provides methods to expose data location information that generally are hidden to exploit a per-file access optimization. The next step to data location research has been done. Some examples like PACman (Ananthanarayanan et al., 2012), RamCloud (Ousterhout et al., 2011), and RamDisk (Wickberg and Carothers, 2012) provide data location techniques but not all of them in a cluster-based environment. When these applications are from scientific field, input files are shared by many of them. CoScan (Wang et al., 2011) studies how caching input files improves execution time when several applications are going to read the same information.

In order to expose the information about data location of multiworkflows, we need to have a global view of it. For (Zhao and Sakellariou, 2006) and (Hönig and Schiffmann, 2006) a meta-scheduler for multiple DAGs shows a way of merging multiple workflows into one, to improve the overall parallelism. Expose the information about data location would be easier to implement on list scheduling heuristics previous to a scheduling stage.

A scientific workflow scheduling taxonomy has been presented by (Yu and Buyya, 2005) with classification and characterization of various approaches. For high-level features to aid in a coherent description of workflow logic and design (flow-charting, logical diagramming, template write down, among others).

More recently (Cerezo et al., 2013), the same design principle opened to disaggregated systems between the abstract layer and the concrete layer. In the abstract layer, the workflow parallelization possibilities are described in a declarative way. The actual execution and distribution of the jobs are managed at the concrete level. A list based scheduling heuristic could be applied on the meta-workflow with a policy to cache shared input files and temporal files. In order to improve the access latency to data a storage hierarchy as a local disk and local ramdisk of the compute node is used.

For our proposal, we try to fit a data-aware multiworkflows scheduling in a cluster environment. Selecting a list-based scheduling implemented with an abstract meta-workflow model. We supply to our scheduler with information about computation time, data location, sizes and shared input files to reduce data access latency on the storage hierarchy.

3 SCHEDULER ARCHITECTURE

A representative characterization of real workflows structures composed by bioinformatics applications as showed on figure 2 is introduced here. We can see that the reference file is a shared input for two different applications; moreover, we realize that applications executed in the cluster are from workflows used for genome alignment, variant analysis, and data file format transformation. We select 4 of the most representative bioinformatics applications; seen in table 1.

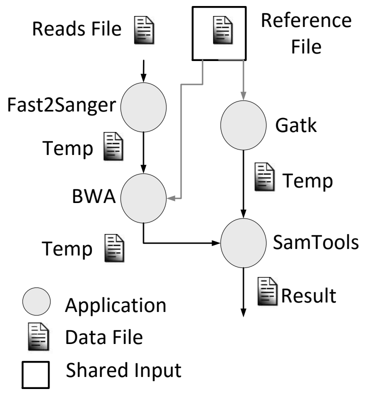

Figure 2: Shared input bioinformatics workflow.

Table 1: Workflow applications considered.

Applications	Bound	Objective
Fast2Sanger	I/O	Format Transformation
BWA	CPU	Alignment
Sam2Bam	I/O	Format Transformation
Gatk	CPU	Variant Analysis

As shown in figure 3, based on the type of bioinformatics research executed in the cluster, we define that there are two main patterns of execution: sequential and parallel branch applications.

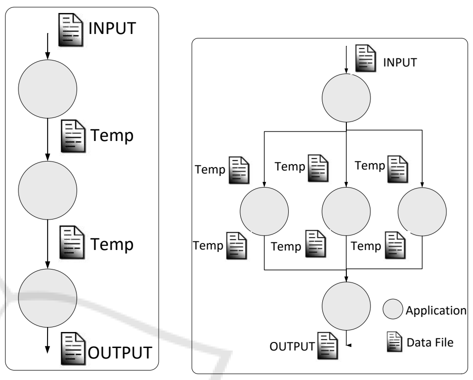

Figure 3: Bioinformatics workflow patterns.

These applications take as input data files with different sizes. We classify files as small when they are below 1 MB, medium below 1 GB and big above 1 GB. Our first approach is to implement a data locality policy and avoid a complete statistical analysis of all possible combinations for locating a file in the different levels of the hierarchy: distributed file system on NFS, a local disk and a local ramdisk of a computational node.

For practical purpose, modules are designed as shown in figure 4. Different workflows from different users could coexist on the system at once. In the prescheduling stage, we will evaluate the attributes of the workflow pattern design received from the user-level. We expose information needed to perform two main operations; first we merge all applications from the studied workflows into a single meta workflow to evaluate a priority list according to a critical path based heuristic using computation time. Second, with the information exposed from merging the applications of the workflows, we know how many applications are using the same data file as shared input. With this information about shared input files, we use local ramdisk, local hard disk or distributed file system as target locations in the hierarchical storage system to allocate them. At last, at the scheduler level, we use the priority list to assign the highest application to a queue of a specific node of the cluster. For these assignments, we consider which node holds the data input for those applications.

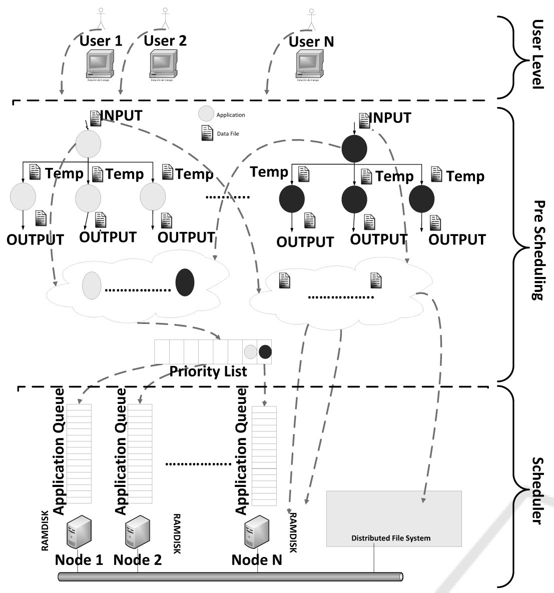

Figure 4: Scheduler architecture design.

With these specifications we propose an architecture model show in figure 5 for our scheduler. Model is composed by four main modules: the App Controller, Data Placement, App Scheduling and the Resource Status.

- `App Controller` – all workflows analysed through the workflow API are merged into one meta workflow. Applications in the workflow are scheduled according to the application computation time and communication time log stored in the workflow engine. This information has previously been selected and categorized. Then, based on the critical path priorities, we build a list of applications using algorithm 1.
- `Data Placement` – at the very begining input datasets are located in the distributed file system and copied to the local disk of the cluster node where the entry applications are going to be executed. All temporal data files produced by the applications for next workflow stages should be stored in the same local disk or Ramdisk. Accordingly with algorithm 1, if the computational capacity of the node is full, a copy of the required data file will be issued in order to start the execution of the next applications of the workflow in a different node. This is done to ensure data independence between applications in different nodes and these file copy operations are kept in a list of data locations.

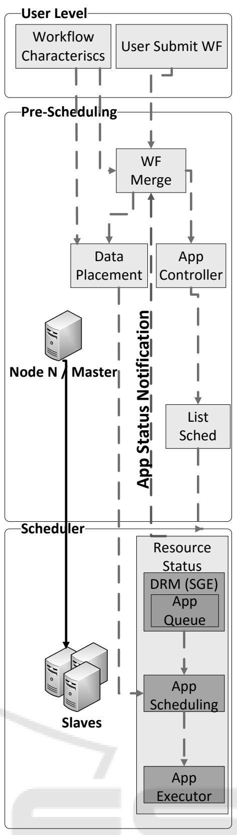

Figure 5: Scheduler architecture modules.

In the pre scheduling stage algorithm 1 from lines 1 to 3 we initialize values for the workflow characterization provided by user definition. Lines 5 to 13 merge all workflows into one meta-workflow adding 2 dummy nodes for start and end nodes. Following line 14 applies a critical path to this new meta-workflow. Data placement implementation is described from line 15. Here we classify data files as small, which are kept in the distributed file system. Also, which big or medium shared files need to be copied to the local storage system. In line 27, we describe a file replacement policy when local storage locations are full. In those cases, we need to copy data files back to the distributed file system to free storage space. Finally, in line 40, we describe how all output files are going to be written into the NFS file system. All these modules work on a master node of the system.

- `Resource Status` – the status of the resources and the implementation of some sub-modules are done with the help of scripts provided by the WF Engine and systems call to the DRM to retrieve and send information and instructions to the clus-

Algorithm 1: Pre Scheduling Stage.

```
   input  : New WorkFlow, User WF Characterizations
   output: Updated Meta Workflow, List Scheduling
1  for All Applications on New Workflow do
2      User Wf Characterization
3      AppId, weight, fileIn, fileInsize, fileOut, fileOutsize, Parents,
         BranchBrothers ;
4  end
5  while New Workflow Applications are not in Meta Workflow do
6      Node add of application from New Workflow to Meta
         Workflow;
7      if application of New Workflow is an entry then
8          edge from Start node of Meta Workflow to application of
             New Workflow;
9      end
10     if application has not successor then
11         edge from application of New Workflow to End node of
             Meta Workflow;
12     end
13 end
14 List = CriticalPath on Meta Workflow;
15 for All Applications on List do
16     if application is new then
17         if fileInSize = SMALL then
18             keep file on Distributed File System;
19         end
20         if (fileInSize = MEDIUM or BIG) and more than 1
             application use it then
21             if Ramdisk is not full then
22                 copy file input from Distributed File System
                     to local Ramdisk of Computational Node;
23             end
24             else
25                 Replace Input file from Ramdisk;
26                 if Replaced file is Temporal and
                     BranchBrothers are DONE then
27                     delete replaced file
28                 end
29                 else if Local disk is not full then
30                     copy replaced input file to Local disk of
                         Computational Node;
31                 end
32                 else
33                     Replace input file from LocalDisk;
34                     copy replaced input file to Distributed
                         File System;
35                 end
36             end
37         end
38     end
39     if FileOut is from Last Application then
40         Copy FileOut to NFS
41     end
42 end
```

ter. App Executor just submits the first application of the list provided by the App Controller. Instructions about which node and resource to use is provided here to the Distributed Resource Manager (DRM) so this can schedule at the internal application queue. In the meanwhile, the Data Placement use the information provided by the DRM to move output files to the distributed file system.

- `App Scheduling` – workflows are made of interdependent applications. The scheduler submits an individual application to the system queue only when they are ready to execute. The list of applications is sorted in one unique list of ready applications. The highest priority application will be on the top of the list to be mapped to a resource and will be sent to the Application Executor when all its parents are done. We apply a scheduling algorithm to the list of applications described in algorithm 2. The priority of an application can change dynamically while the application is not running whether the application belongs to different workflows or users. In this first approach, we do not perform an analysis to select the exact amount of computational resources needed; Instead, we limit the maximum selection to the maximum branch factor of the workflow.

We describe the scheduler module algorithm in 2. In line 1 we receive resource status from DRMAA and system calls. Once we have all the information about resources availability, at line 3 and 4 we sort the list of priorities and upgrade all applications status to STANDBY. We select the resources where to assign applications in line 5 taking into account the maximum branch of the workflow where the current application that is going to submit belongs. From lines 6 to 17 we make effective the resource assignment. If all parents of the application are DONE, we move the application to the end of the list to control the status later. If the status of one of the parents is not DONE, then the application is moved it to the first position to wait for all precedence to be checked and resources are free from it. At the end, from lines 18 to 24 we control if all the applications are DONE. A stop control upgrading status to READY/STANDBY/RUNNING OR DONE for each application is performed.

4 EXPERIMENT DESIGN AND EVALUATION

We introduce a experimental design to evaluate the Multiple Workflow Data-Aware Scheduler. We comparatively evaluate our approach against List Scheduling, List Scheduling with Local Ramdisk and Local Disk storage with Shared Input File in the Cluster. Cluster specifications are shown on table 3. We use a

Algorithm 2: Scheduler.

```
input  : Meta Workflow, List Scheduling, Available Resources
1  Available Resources = Resource Status trough DRMAA calls to the
     Distributed Resource Management and system calls;
2  while List of Priorities not empty do
3      upgrade application status to STANDBY ;
4      Sort List of Priorities;
5      Select Resources = Max(BranchBrothers of first on List);
6      for i = All Applications in List do
7          if all Parents of application == DONE and Selected
             Resources != 0 then
8              if fileInput of Application[i] already on place then
9                  execute Application on Selected Resource;
10                 Selected Resources = Selected Resources - 1
                   ;
11                 add to List of Priorities at the end;
12             end
13         end
14         else
15             add to List of Priorities at first;
16         end
17     end
18     for all applications in List do
19         if application status is DONE then
20             remove application from List;
21         end
22         upgrade application status to
             READY/STANDBY/RUNNING/DONE;
23     end
24 end
```

list of well characterized application to test the workflow system and then analyze a repository of historical execution times. We have two workflow patterns shown in figure 3 with sequential depth factor 8, and parallel branch factor from 2 to 6. In the table 2 we describe how many applications are considered in each pattern. We estimate the execution time using initial values extracted from NFS application executions. Cores and CPU speed, as well main memory and local ramdisk size are shown in table 3 for the 32 compute nodes and the table 4 for file system specifications.

Table 2: Synthetic workflow specifications.

WorkFlow	Applications	Exec Time(s)
Sequential	8	480
Branch	8	358
Seq + Branch	4 + 4	360

Table 3: Cluster specifications.

Cores	CPU(Ghz)	Mem(GB)	Ramdisk(GB)
4	2.0	12	6.2

Workflows are constructed with a synthetic set of applications that simulate the behavior of bioinformatics applications described in table 5, with different bounds of CPU and I/O.

Table 4: File System Specs.

LocalRamdisk	DistFileSys	LocalFileSys
tmpfs	nfs	ext4

Table 5: Synthetic application specifications.

Apps	Exec Time(s)	IO Read(Mb)	IO Write(Mb)	RSS(Mb)	CPU Util(%)
app1	11400	197	304	800	45
app2	1440	67	69	180	98
app3	1020	160	54	480	99
app4	1380	10	47	300	99

We compare here the behavior of the scheduling algorithm in 3 stages. The Classic List Scheduling on NFS shows the obtained makespan of using input, output and temporal files on NFS. Next we have another 2 different implementations of our Multi Workflow Data-Aware Scheduler (MWS DA Local Disk) where all the data sets are located in local disk when we copy input data sets to the local disk of the node where the execution would happen. And last scheduling (MWS DA Local Disk + Ramdisk) where Local Disk and Ramdisk are used, when the input data sets are placed on the local disk and ramdisk. As we can see for 8 to 128 cores and 2 files as shared input files of 2048Mb the algorithm scales very well because the cost of data read and write is drastically reduced when we move it to a closer location of the compute node, see figure 6.

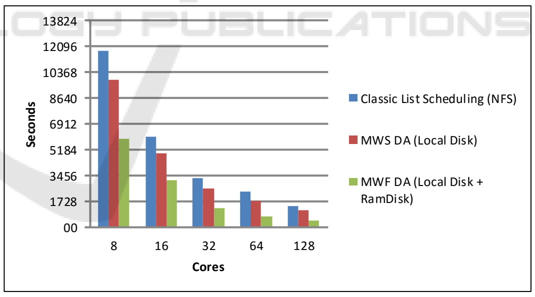

Figure 6: Makespan of 50 workflows on up to 128 cores.

We made different experiments from 8, 16, 64 up to 128 cores with shared input files. There was no significant performance variation for the different amount of cores. Figure 7 shows the result of using 128 cores on 2 shared input files with sizes of 2048Mb, 1024Mb, and 512Mb. The experimentation shows that for big and medium files makespan is decreased due to the relocation of files as we are reducing the number of times we read the same file from NFS or Local Disk. Nevertheless, for small files like 512Mb, the scalability does not improve because reading directly from the distributed file system is faster than moving data to a new location.

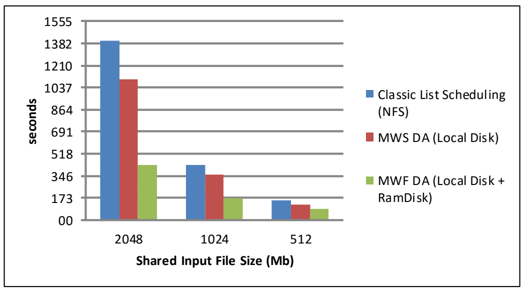

Figure 7: Makespan of 50 Workflows on 128 cores and different shared input files sizes.

The experiment provides us with some ground where we can conclude that our algorithm is better in the case of workflows that share data files in different levels of the memory hierarchy. Without using local storage our gain is about 11% for 50 Workflows running at the same time in a 8 core cluster and almost 20% for 128 cores as we can see in figure 6.

5 CONCLUSIONS AND OPEN LINES

We have studied the state of the art of schedulers for multiworkflows and their taxonomies, and then focus our work in the field of data-aware policies for clusters. We concentrate our efforts in studying disk I/O cluster bottlenecks. We characterize bioinformatics applications where some of them using same data files as input. Techniques like shared input files are desirable to prevent multiple file reads and to improve the performance of the system I/O.

We have considered a list of options for data replacement polices in ramdisk or local disk. To further increase efficiency of the policies, we should consider a better prediction technique of how many nodes, processors and cores.

Looking forward, this scheduler is ready to be integrated it to a real scientific workflow manager like Galaxy (Goecks et al., 2010) which is a web-based workflow manager widely used in the bioinformatics community.

ACKNOWLEDGEMENT

This work has been supported by project number TIN2014-53234-C2-1-R of Spanish Ministerio de Ciencia y Tecnología (MICINN). This work is co-founded by the EGI-Engage project (Horizon 2020) under Grant number 654142.

REFERENCES

Afrati, F., Papadimitriou, C. H., and Papageorgiou, G. (1988). Scheduling dags to minimize time and communication. In *VLSI Algorithms and Architectures*, pages 134–138. Springer.

Ananthanarayanan, G., Ghodsi, A., Wang, A., Borthakur, D., Kandula, S., Shenker, S., and Stoica, I. (2012). Pacman: coordinated memory caching for parallel jobs. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 20–20. USENIX Association.

Barbosa, J. and Monteiro, A. P. (2008). A list scheduling algorithm for scheduling multi-user jobs on clusters. In *High Performance Computing for Computational Science-VECPAR 2008*, pages 123–136. Springer.

Bittencourt, L. F. and Madeira, E. R. (2010). Towards the scheduling of multiple workflows on computational grids. *Journal of grid computing*, 8(3):419–441.

Bolze, R., Desprez, F., and Insard, B. Evaluation of online multi-workflow heuristics based on list scheduling methods. Technical report, Gwendia ANR-06-MDCA-009.

Cerezo, N., Montagnat, J., and Blay-Fornarino, M. (2013). Computer-assisted scientific workflow design. *Journal of grid computing*, 11(3):585–612.

Costa, L. B., Yang, H., Vairavanathan, E., Barros, A., Maheshwari, K., Fedak, G., Katz, D., Wilde, M., Ripeanu, M., and Al-Kiswany, S. (2015). The case for workflow-aware storage: An opportunity study. *Journal of Grid Computing*, 13(1):95–113.

Goecks, J., Nekrutenko, A., Taylor, J., et al. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86.

Gu, Y. and Wu, Q. (2010). Optimizing distributed computing workflows in heterogeneous network environments. In *Distributed Computing and Networking*, pages 142–154. Springer.

Hönig, U. and Schiffmann, W. (2006). A meta-algorithm for scheduling multiple dags in homogeneous system environments. In *Proceedings of the eighteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'06)*.

Ilavarasan, E. and Thambidurai, P. (2007). Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer sciences*, 3(2):94–103.

Kwok, Y.-K. and Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471.

Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B., and Johnsson, L. (2005). Scheduling strategies for mapping application workflows onto the grid. In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 125–134. IEEE.

Meswani, M. R., Laurenzano, M. A., Carrington, L., and Snavely, A. (2010). Modeling and predicting disk i/o time of hpc applications. In *High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2010 DoD*, pages 478–486. IEEE.

N'Takpé, T. and Suter, F. (2009). Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE.

Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Ongaro, D., Parulkar, G., et al. (2011). The case for ramcloud. *Communications of the ACM*, 54(7):121–130.

Park, G.-L., Shirazi, B., and Marquis, J. (1997). Dfrn: A new approach for duplication based scheduling for distributed memory multiprocessor systems. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 157–166. IEEE.

Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

Rahman, M., Venugopal, S., and Buyya, R. (2007). A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *e-Science and Grid Computing, IEEE International Conference on*, pages 35–42. IEEE.

Stavrinides, G. L. and Karatza, H. D. (2011). Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. *Simulation Modelling Practice and Theory*, 19(1):540–552.

Topcuoglu, H., Hariri, S., and Wu, M.-Y. (1999). Task scheduling algorithms for heterogeneous processors. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 3–14. IEEE.

Wang, X., Olston, C., Sarma, A. D., and Burns, R. (2011). Coscan: cooperative scan sharing in the cloud. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 11. ACM.

Wickberg, T. and Carothers, C. (2012). The ramdisk storage accelerator: a method of accelerating i/o performance on hpc systems using ramdisks. In *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, page 5. ACM.

Yang, T. and Gerasoulis, A. (1994). Dsc: Scheduling parallel tasks on an unbounded number of processors. *Parallel and Distributed Systems, IEEE Transactions on*, 5(9):951–967.

Yu, J. and Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record*, 34(3):44–49.

Yu, Z. and Shi, W. (2008). A planner-guided scheduling strategy for multiple workflow applications. In *Parallel Processing-Workshops, 2008. ICPP-W'08. International Conference on*, pages 1–8. IEEE.

Zhao, H. and Sakellariou, R. (2006). Scheduling multiple dags onto heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 14–pp. IEEE.

Zhao, T. and Hu, J. (2010). Performance evaluation of parallel file system based on lustre and grey theory. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 118–123. IEEE.