Future Generation Computer Systems 74 (2017) 51–62

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A Critical Path File Location (CPFL) algorithm for data-aware multiworkflow scheduling on HPC clusters

César Acevedo *, Porfidio Hernández, Antonio Espinosa, Víctor Méndez

Computer Architecture & Operating Systems Department (CAOS), Universitat Autònoma de Barcelona (UAB), Bellaterra (Barcelona), Spain

HIGHLIGHTS

- Multiworkflow scheduling strategy on a cluster is proposed.
- Critical path with data-aware.
- Scheduling is proposed to improve makespan of bioinformatic workflows.
- Simulator engine extension to scale on to a bigger cluster infrastructure and new storage hierarchy.

ARTICLE INFO

Article history:
Received 6 September 2016
Received in revised form
12 April 2017
Accepted 13 April 2017
Available online 24 April 2017

Keywords:
Multiworkflows
Cluster
Scheduler
Simulation
Critical path
Data processing

ABSTRACT

A representative set of workflows found in bioinformatics pipelines must deal with large data sets. Most scientific workflows are defined as Direct Acyclic Graphs (DAGs). Despite DAGs are useful to understand dependence relationships, they do not provide any information about input, output and temporal data files. This information about the location of files of data intensive applications helps to avoid performance issues.

This paper presents a multiworkflow store-aware scheduler in a cluster environment called Critical Path File Location (CPFL) policy where the access time to disk is more relevant than network, as an extension of the classical list scheduling policies. Our purpose is to find the best location of data files in a hierarchical storage system.

The resulting algorithm is tested in an HPC cluster and in a simulated cluster scenario with bioinformatics synthetic workflows, and largely used benchmarks like Montage and Epigenomics. The resulting simulator is tuned and validated with the first test results from the real infrastructure. The evaluation of our proposal shows promising results up to 70% on benchmarks in real HPC clusters using 128 cores and up to 69% of makespan improvement on simulated 512 cores clusters with a deviation between 0.9% and 3% regarding the real HPC cluster.

1. Introduction

Current scientific applications must deal with large data sets, usually demanding large amounts of computation and communication times. Schedulers are responsible for allocating applications to processors and ensure the execution precedence. Applications in a workflow are usually represented as a node in a graph.

Direct Acyclic Graphs (DAGs) are a good way of modeling task dependency relationships like those typically found in a workflow. Despite their wide usage to represent application stages, DAGs lack relevant information on how to deal with data files. That is, they do not show any detail on how input, output, and temporal data files are transferred to actual computational nodes where the applications run.

Fig. 1 shows an example of the many combinations that can be found when considering the possible locations of data files and a common computational cluster system architecture. Cluster nodes have their own memory hierarchy and their own secondary storage subsystem where we can find hard drive disks and other smaller but faster general purpose storage device. Also, nodes are usually connected to a distributed file system via a fast interconnection network. From the point of view of the data handling, workflow stages need to manage input, output, and many temporal files [1]. It becomes a challenge to determine which is the best location for all the files needed for the different computation steps defined in the workflow to get the best performance of the system.

* Corresponding author.
E-mail address: cesar.acevedo@caos.uab.es (C. Acevedo).

http://dx.doi.org/10.1016/j.future.2017.04.025

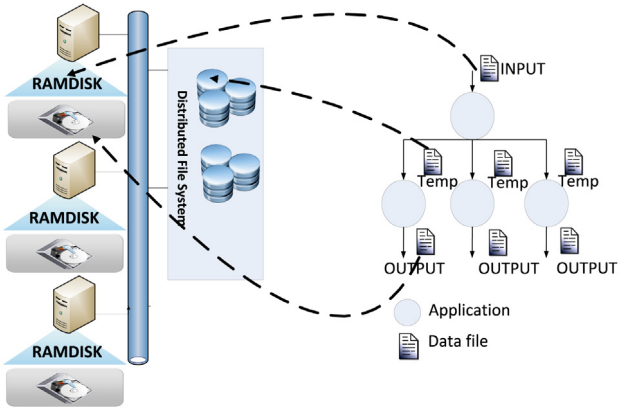


Fig. 1. Application graph and its relationship with data localization.

Scheduling a workflow with precedence constraints is an important problem in scheduling theory and has been shown to be NP-Hard [2]. There are many studies on how to manage a single workflow, specifically when trying to schedule tasks onto heterogeneous domains [3]. There has been 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 [4–12].

Workflow-aware storage strategies study data file locations in many levels of the storage and the memory hierarchy as relevant criteria for the application scheduling. These strategies have previously been used to reduce the I/O load of the network of cloud [13] and grid systems. In these systems, when shared files are read several times between different computational nodes, the performance of the system depends on the interconnection network capacity. In these cases, shared files are located on local disks to reduce the amount of I/O operations. In a cluster environment with a high-speed network the disk becomes the demanding resource, which generates I/O waiting times when many applications simultaneously request the access to files. In HPC systems I/O performance has been studied by [14] that proposes the use of Ramdisk as a storage system with data location techniques on a section of the memory system.

When considering the data usage of common bioinformatics workflows for common data analysis cases as variant analysis, read mapping and sequence alignment, we find some common special characteristics:

- Large volume of input data to be processed, starting at 2 GB for bioinformatics data files.
- Large volume of data are sequentially processed in their entirety, like input files for read mapping and data format transformation.
- Important amount of data being shared by similar applications: typically files like human genome indexes.
- Amount and volume of temporal files generated by some applications. Input files of 2 GB can generate between 4 and 6 GB of temporal files.

Due to the peculiarity of bioinformatics workflows that are composed of many applications that share data files, we can take advantage of keeping a cached version of input and temporal files. Then, these files are kept in a highly accessible storage such as ramdisk built in the main memory of the system or in a solid state disk. From here, we propose an extension of the classic model to a shared input file policy of execution and mapping of workflow applications on this kind of system architectures.

In summary, we proposed at [15] a scheduling algorithm for multiworkflows in a cluster environment called CPFL (Critical Path File Location) that is a continuation of the work presented at [16]. Our objective is to improve the effective use of High Performance Computing (HPC) platforms for the execution of Data Intensive Applications (DIC) by extending the multiworkflow model on to store-aware scheduling. Multiworkflow such as bioinformatic and Epigenomics use shared input and temporal data files. Typical bioinformatic workflow has input files starting at 2 GB that generates temporal files of 6 GB. We realize that other workflows such Montage [17] generate several shared temporal data files. For a small Montage, 200 input files of 2 GB generate over 1000 shared temporal data files of up to 500 MB each. Due to that, store-aware scheduling approach on cluster environment helps to improve overall makespan. Keeping shared files on a storage hierarchy system we reduce the time access to disk on regards to network access.

We evaluate the effect of moving the execution of certain tasks to nodes where needed data items are previously located. For bioinformatics applications with input files in common, we move those files to a fast memory storage level. As a result, we increase locality and reduce the number of disk accesses. We also want to support the execution of different workflows considering their main resource limitations like Input/Output (I/O), memory or CPU. To consider new technologies and different kind of workflows we introduce a simulator and the extension needed to deploy the scheduler on it.

This approach has been evaluated with an initial set of experimental environments: a batch of workflows statically merged into a meta-workflow and then applied a classic List Scheduling like Heterogeneous Earliest Finish Time (HEFT). This heuristic is typically used to schedule a set of dependent tasks onto a network of heterogeneous workers taking computation time into account. In our case, we have introduced the use of a Network File System (NFS) as the storage system for all the data files. This is compared against a new List Scheduling heuristic for data-aware multiworkflows with a critical path using a local disk and local Ramdisk as storage hierarchy. In our experiments, we use synthetic bioinformatics workflows as a benchmark to test our proposals as well as Montage and Epigenomics benchmarks because they typically produce plenty of temporal files. The results show performance improvements up to 70% against HEFT modified for multiworkflow with better usage of storage hierarchy such as local disk and ramdisk.

The rest of the paper is organized as follows. State of the art is discussed in Section 2. Then, we give an overview of the scheduler architecture in Section 3. Section 4 describes WorkFlowSim simulation environment and introduces its use to validate schedulers scalability. Section 5 elaborates the experiment design and evaluates the performance of proposed algorithm in the experimental platforms presented. Finally, in Section 6 we summarize the results obtained and lay out the future work.

2. Related work

The scheduling problem, understood as the task of allocating computational jobs to processors to define their order of execution without restrictions, is NP-complete [2]. As a relevant problem, many heuristics have been proposed for its resolution [18]. Among the most important we can find: clustering heuristics [19], duplication based heuristics [20], meta heuristics (Genetic Algorithms, Simulated Annealing, tabu search) and list scheduling heuristics [21] based in assigning priorities depending on the critical path length associated to each node [3].

In our work, we are considering a generalization of the problem taking scientific workflow tasks as jobs to manage. This case of workflow scheduling has been widely studied and we can find many algorithms based on DAG list scheduling heuristics. Some

of the most relevant proposals are: Predict Earliest Finish Time (PEFT) [22], Lookahead [23] and Heterogeneous Earliest Finish Time (HEFT) [24]. The HEFT algorithm proposes to select the application with the highest rank at each level of the graph. This rank is the position of the application along with the critical path of the whole workflow, using computation time as a cost. Then it assigns applications to processors in order to minimize the execution time. The HEFT algorithm is the best proposal in terms of robustness and schedule length. Its also very relevant considering its temporal complexity $O(n^2 * m)$ where n is the number of jobs and m is the number of processors.

When we consider the increase in computational resources available at today's servers, the execution of a single workflow might not need all computational resources in the system, such as CPU cores. This opens the door for attending multiple workflows simultaneously. From here, we need to re-evaluate the classical scheduling solutions for the concurrent execution of multiple workflows.

Workflow scheduling methods have been well studied [25]. The majority of proposals apply a similar principle of transforming a multi-workflow into a single one by connecting a given number of dummy nodes at both extremes of the graph. Then, we find different strategies of managing the obtained group of connected tasks.

It is very common to find list-based heuristics for solving multiworkflow scheduling problems. List-based heuristics maintain a list of all tasks of a specific DAG according to their priorities. Every task of the graph is given a priority. Then, a task list is built considering a decreasing order of priority. Last, following the list of precedence given by the graph, the best computational resource is selected for the task with the highest priority, considering a cost function previously defined.

We are concentrating our work in analyzing bioinformatics workflow jobs. These usually make extensive use of certain types of applications that typically share data files between them. Our proposal applies a well-known list-based heuristic (HEFT), adapting its use to the multiworkflow context by merging workflows into a meta-workflow.

Finally, we want to improve the performance of bioinformatics workflows by carefully analyzing their I/O behavior. Bioinformatics applications usually process input and output temporal files sequentially. That is, input files are entirely processed from the beginning to the end. Moreover, some of those files are shared by several applications, so they are accessed several times during the execution. Here, our objective is to make good use of the memory hierarchy to cache common data files and executing applications in nodes where data files are located. We are interested in evaluating the performance of a current system with different levels of storage hierarchy such as distributed file system, local disk, local ramdisk and main memory.

Initially, we modify most relevant list scheduler, HEFT, to consider communication costs in the implementation of the rank formula of the critical path. We calculate new communication costs by finding the cost of accessing files being placed at any folder of the storage hierarchy. Then, we obtain our CPFL policy that considers computation and communication costs.

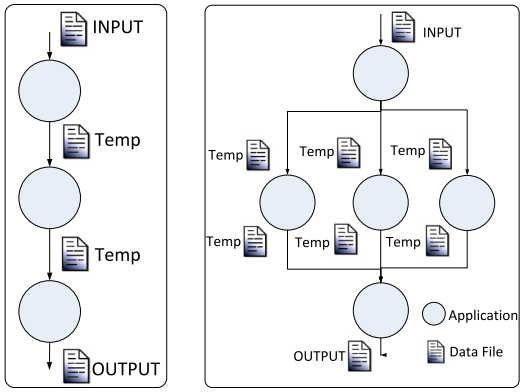
Although we can find many studies that make reference to workflow scheduling methods [25], there is room for improvement in the multi-workflow scheduling case [8] that group individual applications of multiple workflows into a ready-application pool and then apply a simple HEFT to that pool. For [6] the approach is to compose workflows with the objective of combining multiple DAGs in one before applying static algorithms, considering that all representative parameters of the different graphs are previously known. In the same way, Hönig et al. [7] describe a meta-scheduler for multiple DAGs that implies the fusion of these multiple DAGs in just one, to improve global parallelism and minimize the inactivity time of resources. Studies like [26] extend HEFT to the multiworkflow context in a grid environment. In this way, they generate a single aggregated workflow composed of the composition of all workflows given.

As an initial approach, we considered a batch of workflows as the workload, defined as a group of workflows ready to be scheduled on the system. When new workflows arrive, they are treated as a new batch to be scheduled. This approach transforms the scheduling of dynamically arriving new workflow tasks into a static problem. This static approach gives us the ability to ensure that applications of multiple workflows are going to be grouped in batches according to how many resources are needed such as storage capacity and computation time. The admission function ensures that waiting time will not exceed execution time of the applications. Many studies [27–29] show that the static strategy can potentially obtain a nearly optimal result for synthetic and real workflows [30]. Simulation studies such as [27] also suggest that the static algorithms provide good results for intensive workflow applications even when future job information is not complete.

The relevance of data storage for scheduling has already been introduced by Bent et al. [31] when presenting the batch-aware distributed file system (BAD-FS). Its objective was to propose that the file system could export some architecture and behavior features that could be considered when designing the scheduler. Moreover, [32] introduces the concept of an indexing framework, focuses on data distribution for big data applications over a distributed file system. The approach of SmallClient and others nevertheless refers to the idea of splitting data files into smaller chunks of data, stored in a key–value format on a distributed file system. Finally, we do not need to split data files into chunks, but we do need to retrieve the complete data files several times in case of shared input files and temporal files to read once and used several times. In that case, even when we are working with a high-speed network we want to avoid the need of reading from distributed file system or a local disk where I/O bottlenecks are present. We look for exploiting the cached data file in the main memory using a local ramdisk and submitting applications that share data file to be executed in the same execution node. Due to the inability of having unlimited ramdisk, we introduce the use of hierarchical storage levels with Distributed File System, Local Disk, and Local ramdisk, which SmallClient and other frameworks do not take into account.

Data transfers between nodes of a system can become the major overhead when network bandwidth limits are met. Stork [33], describes a scheduler that focus the problem of transferring k files from m sources to n destinations. Stork is focused on finding the best protocols, the parameters to tune the transferences and the order of the transfer requests, among others. In our case, our limiting factor is the disk, not the network, so we need to study alternative techniques to keep data close to the computational nodes in multi-level fast storage systems.

In a similar scenario, there are models like [34] that evaluate the performance of cloud and clusters attending their storage systems. In this case, a global storage backup procedure is used to compare the execution time between cloud and cluster systems when both systems process the same amount of data requests. This work states the relevance of adequately manage the requests when the network is the limiting resource. In contrast, our objective is to benefit from the locality patterns commonly shown by the scientific applications in study. In this sense, our limiting resource is the disk and we need to optimize memory access to common files to improve the performance of the applications. Nevertheless, [35] propose the use of cloud computing for data analysis of weather. Taking advantage of data distribution on private clouds to improve velocity on thousands of data to be processed, analyzed and



(a) Sequential pattern.

(b) Parallel pattern.

Fig. 2. Bioinformatics workflow patterns.

visualized simultaneously. Contributing to a affordable technology and to the society by inform the general public about the impacts to their schedules due to the sudden change of the weathers. In this approach our main contribution is a scheduler that allow the use of existing storage technology and prepare to deal with new ones by reducing makespan of bioinformatic multiworkflow on cluster. The possibility of using clusters, allows the bioinformatic society, where most of the data has privacy, nontransferable and secret policy to take advantage of the available computational resources. Finally, another contribution is to provide solutions to similar technologies such as CloudSim to extend the mechanism of simulating workflow execution in order to reduce cloud storage hierarchy costs and to provide a method on WorkflowSim to researchers that wants to probe or develop new store-aware heuristics.

Few previous works have considered the location of input, output and temporal data files in shared clusters environments. All these aspects have proven to be critical in the management of workflows of data intensive applications (DIC) [36,37] as is the case of this work.

In our case, we want to present a scheduler that is able to coordinately deal with multiple workflows for shared cluster environments and that takes into account the localization of input, output and temporal files.

3. Scheduler architecture

In this section we are going to describe our scheduler proposal to improve the performance of bioinformatics workflows. First, we are going to give some details on the application data dependencies found in the workflows. Then, we describe how meta-workflows are built and how we can use data dependencies information to locate files in the storage hierarchy. Finally, we provide a basic description of how data files are stored in the different hierarchy layers using a prefetching algorithm.

As shown in Fig. 2, based on the type of real bioinformatics data analysis executed in the cluster, we found two main patterns of workflow execution: sequential and parallel branch. We can see that one input data file for one application could generate several temporal shared files. Then, we could have a depending tree of different applications for the case of a parallel branch. In this work, we are going to study a list of bioinformatics workflows such as genome alignment, variant analysis, and data file format transformation. In these workflows, applications can be well characterized and we can determine which data files are shared.

To achieve our objective, we first create a single meta workflow to evaluate a priority list according to a critical path based heuristic using computation time and communication cost. Second, with the information exposed from the meta workflow, we know which applications are using the same data file as shared input, and shared temporal files. Information about shared input files helps us to locate data files at local ramdisk, local hard disk or distributed file system as levels of the hierarchical storage system. Lastly, we keep a list of locations of data input files and use a priority list to execute applications that share files at the nodes where the files are located.

The main idea of the solution, first, is to determine the location of the data files and use a bioinformatics application characterization, that provides execution time, I/O reads, I/O writes, RSS, and CPU utilization metrics. The algorithm will retrieve information from the workflow pattern about shared input files as their sizes and current location. Initially, all data files are located at the Distributed File System. In this moment, we apply a prefetching algorithm that moves data files to ramdisk when the data file is bigger than 1024 MB and it is shared as input or temporal file from more than one application. The ramdisk has 6 GB at each computational node. If there is no enough space, it will use a lower storage level as local disk or the distributed file system. Any other data files are stored directly in the distributed file system. This prefetching is useful to minimize data loading times and allows the distribution of needed data files to network file systems that provide better latency than actual disks.

We present the design of the architecture that composes the system that implements our solution. For practical purposes, we are going to describe each of the 3 layers of the architecture shown in Fig. 3 User-Level, Prescheduling, Scheduler and their modules.

3.1. User-level

Each scientist can design their workflows in a scientific workflow management tool such as graphical web-based applications, scripting or any other workflow composition tool independent from our scheduler. In this layer, workflow characteristics metadata is defined and is encapsulated in an XML file to provide to the next layers.

3.1.1. Workflow characteristics

For our initial approach we prepare a set of characteristic application of workflows. The workflow management tool log feeds the scheduler with a Workflow Characteristics set. The metadata generated contains Application Names, Application Version, Command Line, Data Files Used, Data Files Sizes, Execution Time.

3.1.2. Submit WF

By the user-level workflow management tool it can submit the workflow created to the system.

3.2. Prescheduling

We evaluate the attributes of the workflow pattern design received from the user-level and define where to locate data files in the storage hierarchy previous to create the list of applications to schedule with priority based on the critical path.

3.2.1. WF merge

In this module all applications from the studied workflows that arrive at the system are merged into a single meta workflow. The meta workflow generated is implemented with two dummy initial and final node to evaluate a priority list according to a critical path based heuristic. We use computation and communication times obtained from user-level workflow characteristics.

3.2.2. Data placement

This module is in charge of locating the data files to a specific storage hierarchy level. With the information exposed from merging the workflows at WF Merge Module, we know how many applications are using the same data files as 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 the very beginning, input data files are located in the distributed file system and copied to the local disk of the cluster node where the 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.

3.2.3. The App Controller

We have a group of workflow batches to schedule. Initially, we apply critical path to the first batch of workflows and prepare their execution. When a new batch of workflows is scheduled, the App Controller re-runs the critical path algorithm to the new applications in the system, but without changing the state of those applications that are already executing. No applications are assigned to any computational unit if there are previous applications in the workflow dependency not yet finished.

3.2.4. Order App List

Once the App Controller finishes the critical path of a batch of workflows, the Order App List is in charge of keep a queue of applications to submit to the execution node.

3.3. Scheduler

The scheduler assigns applications to specific nodes of the cluster following the priority list calculated at App Controller module and provided by the Order App List module. The assignments consider which node holds the input data for those applications.

3.3.1. Resource manager system

This module takes care of the status of computational resources of the system and the applications running on the platform.

3.3.1.1. App scheduling. This module manages the interdependent applications of workflows and is where the scheduling is done. The list of applications provided by Order App List Module is sorted in one unique list of ready applications. The highest priority application will be on the top of the list waiting to be mapped to a resource. It will be sent to the Application Executor when all the parents of the application are done. The priority of an application can change dynamically while it remains in the queue without being executed, no matter if it 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. Branch factor is calculated at WF Merge Module at Prescheduling level.

3.3.1.2. App executor. This module submits the first application of the list provided by the Order App List Module. In this module, the application is finally submitted through the DRM to a specific computational node that contains all data files needed for the execution.

3.3.1.3. DRM. For regular HPC applications, the management of resources is done through a Distributed Resource Management (DRM) system that provides information about the cluster and applications, so this can schedule at the internal application queue. We take advantage of the DRM by processing information logs of the system and scripts provided by a workflow engine. Instructions about which node and resource to use is provided here to the App Controller. In the meanwhile, the Data Placement use the information provided by the DRM to move data files through the different levels of the storage hierarchy system.

Finally, we introduce some implementation details of the module implementation with the objective of providing more details on how the algorithm works on each stage.

The user-level module has no algorithm because it depends on the workflow management tool that user wants to use to create the patterns. Workflows characteristics in provided by an XML historical log.

In the pre-scheduling stage, we use algorithm 1. We use lines 1–3 we initialize values for the workflow characterization provided by user definition. In lines 5–9, we merge all workflows into one meta-workflow adding two dummy nodes for start and end nodes. Following line 10 applies a critical path analysis to this new meta-workflow.

Data placement implementation is described from line 14. Here we classify data files to keep small files ($<$1 MB) in the distributed file system. Also, we decide which larger shared files such as medium ($>$512 MB) and big ($>$1028 MB) need to be copied to the local storage system. In line 21, 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 local storage space. Finally, in line 33, we describe how all output files are going to be written into the NFS file system.

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 ;   // Characteristics provided at the
   |  User Level Layer
3  |  User Application Characterization ;                  // Application
   |  characteristics, Data Files with sizes, and Application
   |  Parents and Branch Factors are provided by the User Level
4  end
   /* The initial batch of Workflows arrive at the system and
      are merged at prescheduling layer through the WF Merge
      Module here                                                    */
5  while New Workflow Applications are not in Meta Workflow do
6  |  Add New Applications to Meta Workflow;
7  end
8  List = CriticalPath on Meta Workflow ; // The App Controller execute a
   CriticalPath algorithm to the Meta Workflow to generate the
   Priority List
   /* Data Placement Module, locate each data file of the
      Priority List Applications                                     */
9  if fileInSize = SMALL then
10 |  keep file on Distributed File System ;     // Third Level of Storage
   |  Hierarchy
11 end
   /* BranchBrothers used to determine Application using
      shared files                                                   */
12 if (fileInSize = MEDIUM or BIG) and Shared then
   |  /* Ramdisk or Local Disk is selected as storage              */
13 |  Select Storage Hierarchy according to storage availability;
14 end
   /* Output Data Files goes to the Distributed File System
      always                                                         */
15 if FileOut is from Last Application then
16 |  Copy FileOut to Distributed File System;
17 end
```

Next stage, for Scheduler level, we apply a scheduling heuristic to the list of applications described in algorithm 2. Initially, all applications arrive as READY to be schedule. In line 1 we receive resource status from DRM and process log analysis. 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. In this way, we describe which applications are ready to

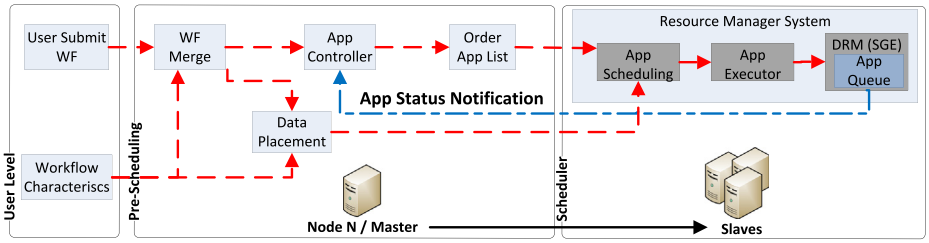


Fig. 3. Architecture modules.

run but are not allocated yet to a computational unit. We select the target computational resources in line 5 considering the maximum branch of the workflow of the current application. From lines 6 to 17 we make effective the resource assignment and change application status to RUNNING which means that the application is correctly assigned to a computational unit. If all parents of the application are DONE, that is, these applications finish execution correctly, we move them 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 to be free. 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.

Algorithm 2: Scheduler

```
   input : Meta Workflow, List Scheduling, Available Resources
1  Available Resources = Resource Status through the Distributed Resource
   Management and read process log;
2  while List of Priorities not empty do
3      upgrade application status to STANDBY ;
4      Sort List of Priorities ;          // Internal List of Applications
5      Select Resources = Max(BranchBrothers of first on List) ;      // Maximum
       cores according to parallelism
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
                   /* If precedence is complete then send
                      application to resource                          */
9                  execute Application on Selected Resource;
10             end
11         end
12         else
13             add to List of Priorities at first;
14         end
15     end
       /* State machine should be always updated                       */
16     for all applications in List do
17         if application status is DONE then
18             remove application from List;
19         end
20         upgrade application status to READY/STANDBY/RUNNING/DONE;
21     end
22 end
```

4. Policy scalability using WorkflowSim extension

To verify if the Data-Aware Multiworkflow Cluster Scheduler [15] is scalable in larger clusters, we decided to use a well-known workflow simulator. WorkflowSim [38] is an open source simulation tool developed at the University of Southern California and is an extension of previous CloudSim simulator. Is used to simulate workflows that have been modeled by DAGs defined through XML files and implements various classic scheduling algorithms like HEFT, Min–Min, Max–Min, FCFS.

In the Fig. 4 as we can see, the simulator has two main groups of modules, Submit host and Execution Site. We highlight those modules that we extended to scale our proposal.

Table 1
Cluster simulation specs.

Specs	Standard simulator	Simulator (IBM-like)
Nodes p/Cluster	4	32
Processors p/Node	1	4
Processors Freq.	1000 MIPS	6000 MIPS
RAM p/Node	2 GB	12 GB
Disk capacity p/Node	1 TB	10 TB
Ramdisk	–	6.2 GB
Net latency	0.2 ms	0 ms
Internal latency	0.05 ms	0.05 ms

At Submit host, the workflow mapper is responsible for importing several DAGs which are concatenated for multiworkflow execution. Then, it creates a list of applications to be assigned to available resources. In any case, we must enforce applications original dependencies to respect the workflow natural execution order of predecessors.

When needed, the clustering engine is responsible for encapsulating multiple applications within a single job. A workflow scheduler, according to user-defined criteria, effectively adds every job or application to a queue of ready applications to be assigned to worker nodes.

We introduced CPFL (Critical file Path Location) in the workflow Scheduler component following the guide for extensions of WorkflowSim by adding a Java CPFLScheduler file implementing the CPFL data-aware scheduler described above.

Our objective again is to take advantage of data locality to reduce the cost of communication in the storage hierarchy (distributed file system, local disk, and Ramdisk). When a job reaches the remote scheduler, it has to wait until the assigned worker node is free but with the required data already there.

We modified the Execution Site module of WorkFlowSim, specifically the representation of the worker nodes.

Infrastructure parameters like a number of processors, RAM capacity, local storage capacity, types of local storage are defined at worker node. We needed to add a new storage type, the Ramdisk, to evaluate our proposal. Once we had the new storage type, a new cluster was created with a given amount of workers, with defined parameters like operating system and communication costs between the storage hierarchy levels.

The next step was to adjust the parameters of the simulator to behave as close as possible to our local IBM cluster, both are compared in Table 1.

5. Experiment design and evaluation

In this section, we introduce an experimental design to evaluate the CPFL MultiWorkflow Data-Aware Scheduler proposal. First, we comparatively evaluate our approach against the HEFT classic list scheduling on a distributed file system (NFS) of our local cluster. The impact of the hierarchical storage system when we do not use critical path is done with a random scheduling approach of applications and just allocating data files in the Ramdisk and/or Local Disk, this comparative is done under Without Critical Path

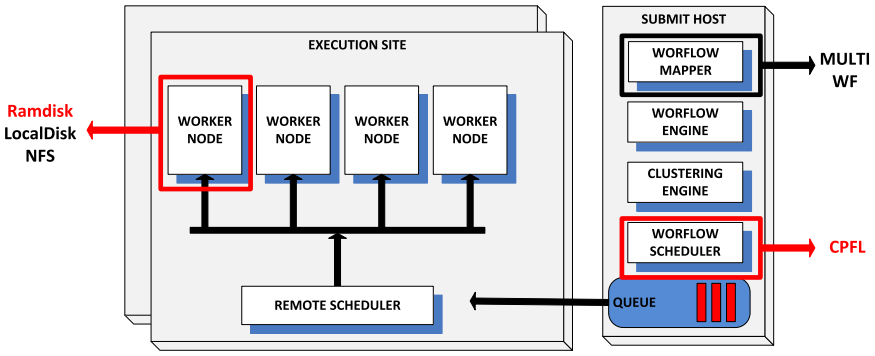


Fig. 4. WorkflowSim components [38].

File Location (SCPFL) name. Additionally, we also evaluate CPFL on Local Ramdisk and Local Disk storage with Shared Input File in our experimental cluster.

The cluster environment has 32 nodes, and each of the nodes has a CPU with 4 cores at 2.0 GHz, 12 GB of main memory, and a local Ramdisk of 6.2 GB. The file systems that we are using are NFS as distributed file system, an EXT4 local disk format, and a TMPFS Ramdisk.

In the cluster that we use, a large amount of data analysis is done by running bioinformatics applications. In the example provided by Fig. 5 we describe a workflow where a representative trait of workflow structures found in bioinformatics applications is in use of the same data input for different applications. Most workflow work is applied to different bioinformatics data analysis as genome alignment, variant analysis, and common data file format transformations.

We use a list of commonly used and well characterized bioinformatics applications to test the workflow management system and then analyze a repository of historical execution times to develop a synthetic workflow that behaves as a bioinformatic workflow.

Considering these applications as nodes and their dependencies as inputs, we have designed a synthetic workflow pattern that we will use for our experiments. Applications shown in Table 2 were selected as well to extrapolate relevant execution times, communication costs and resource usage in Table 3. They are the elements that compose the synthetic workflows for our experimentation.

For validation purposes, we have selected well-known experimental data intensive application public workflows [39]. In this case, we have selected Montage [17], an I/O-bound workflow used in astronomy to generate mosaics of the sky, and Epigenomics CPU-bound bioinformatics workflows.

In Table 3 we describe a list of metrics ranges applications that compose Montage and Epigenomics as well as the synthetic workflow application develop according to 5 and their application analysis on Table 2.

We show in Fig. 6, the graph schemas corresponding to the described workflows. The first is a synthetic workflow based on the type of bioinformatics data analysis done in the cluster. The second is the Epigenomics and third is the Montage workflow schemas obtained from the Workflow Gallery [40].

Synthetic workflow at Fig. 6(a) helps us to relate Fig. 5 and Table 2 with the synthetic workflow generation. We need to represent 3 types of data files (Input, Shared Input/Temporal, and Output) and their possible location on the storage hierarchy system. At first, input data files will be at Distributed File System. Shared Input Files and Temporal Data files will be located at Local Ramdisk or Local Disk according to space availability. Output Data files are directly located at Distributed File System. The real applications are represented with synthetic applications as shown in the same figure to simplify the installation of applications and

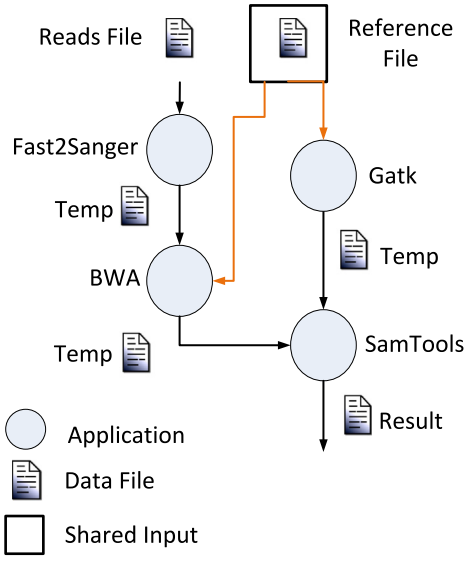


Fig. 5. Shared input bioinformatics workflow example.

the excessive execution time by extrapolating the computation time. We define a branch factor of 6 because bioinformatic applications usually have input files of 2 GB, generating temporal files of up to 6 GB that will be enough to stress our 6 GB on Local Ramdisk with 32 files of 2 GB, one for each node (192 GB is the total size of ramdisk combining the 32 nodes of the cluster). The fallowing 18 input files has no space on the Ramdisk and they are located on Local Disk. On each synthetic workflow, we introduce 2 shared input files that produce 6 more temporal files of up to 6 GB, provoking that the local Ramdisk and local disk becomes stressed, forcing to use NFS.

We analyzed the workflow execution makespan times as the main result of the policies in the study using our experimental cluster defined above. Then, we are proposing the use of WorkflowSim Simulator to evaluate how CPFL scales on larger clusters.

We considered the execution of each of the workflows presented using different shared input file sizes. Our experimentation considered 512, 1024, and 2048 MB input file sizes. We found that the results obtained were very similar. Hence, we are showing the figures for one case for each combination of data size and computational resources. For initial scheduler heuristic profile which support experimental scenarios, tabular information is provided in the Appendix.

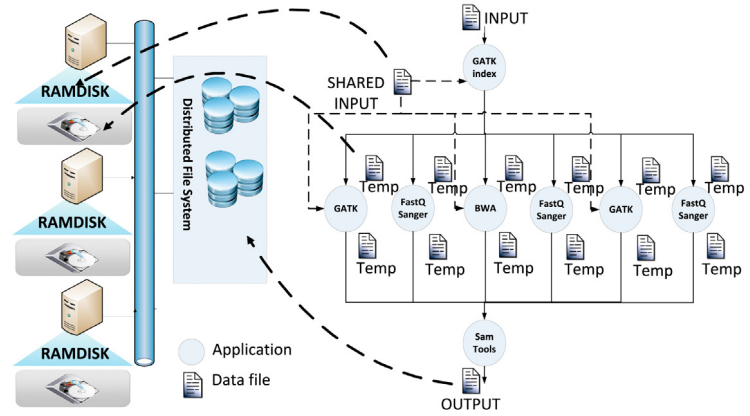
For all experimental cases, we considered the need of scheduling multiworkflows. To do so, we used a list of combinations as workloads: (A) 50 synthetic workflows (B) A group of 10 Epigenomics workflows, (C) A group of 10 Montage Workflows. For real

Table 2
Workflow applications considered.

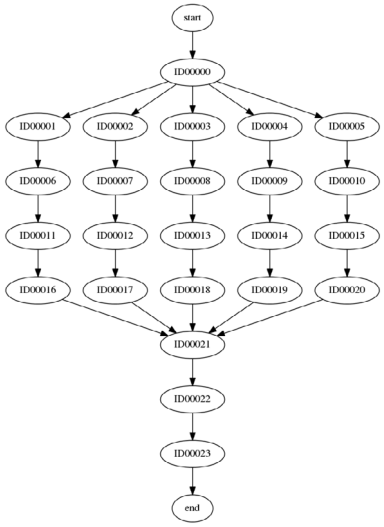
Apps	Exec time (s)	IO (Mb)		RSS (Mb)	CPU util (%)	Resource bound	Objective
		Read	Write				
BWA (b,c,d)	11 400	197	304	800	45	CPU	Alignment
Fast2Sanger (a)	1 440	67	69	180	98	I/O	Format transformation
Sam2Bam (h)	1 020	160	54	480	99	I/O	Format transformation
Gatk (e,f,g)	1 380	10	47	300	99	CPU	Variant analysis

Table 3
Montage/Epigenomics/Synthetic execution metrics.

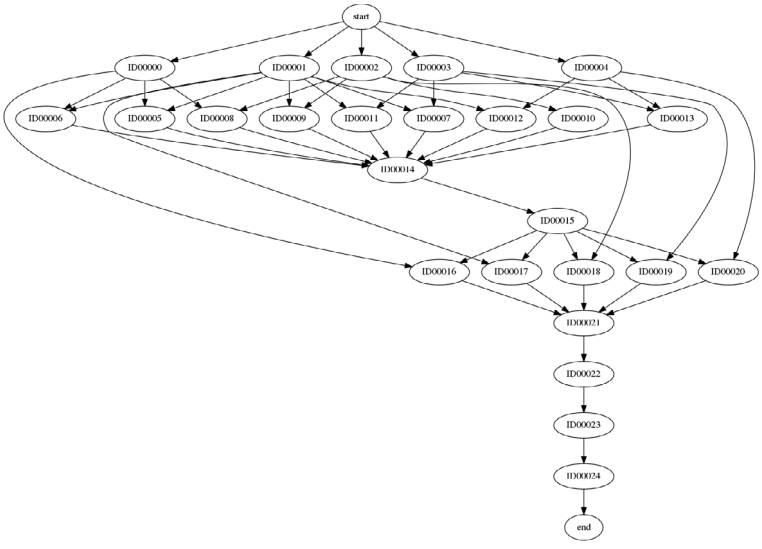
Workflow	Exec time (s) Min–Max	IO read (Mb) Min–Max	IO write (Mb) Min–Max	CPU util (%) Min–Max
Montage	1–5	1–29	1–83	2–100
Epigenomics	1–4065	12–14 676	10–103 821	4–100
Synthetic	7–179	2–201	0–305	3–98



(a) Synthetic workflow pattern.



(b) Epigenomics benchmark.



(c) Montage benchmark.

Fig. 6. Workflow patterns considered in the experimentation.

cluster case, we made up to 10 executions with a deviation of no more than 6%. For case B and C the execution made in a simulator, the results were the same always due to the hard definition of attributes.

5.1. Real cluster experimental scenario

We found that executing type (A) 50 synthetic workflows due to resource assignment for parallel branch factors. We

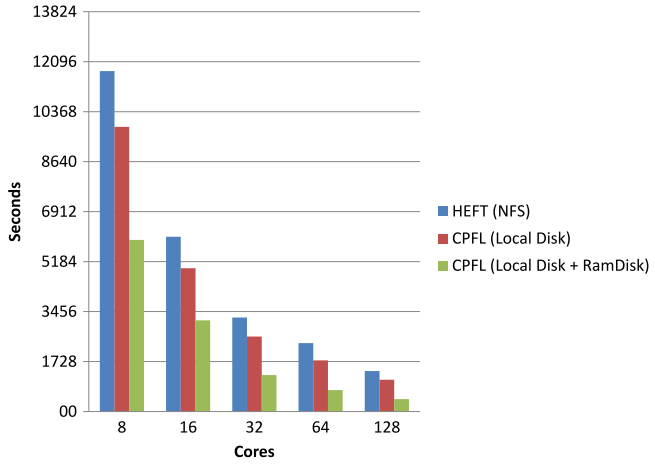


Fig. 7. Synthetic workflow makespan with 2048 MB of shared input files.

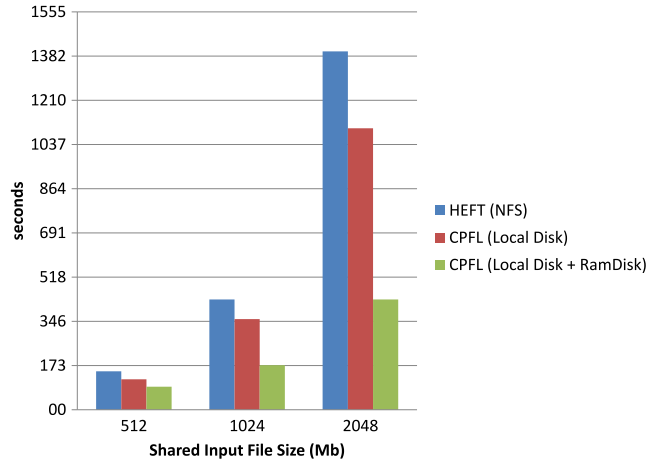


Fig. 8. Synthetic workflow makespan on 128 cores.

prepare a workload of 5 batches of 10 workflows each with the corresponding 2 shared files on each batch. Initially for 2 different shared input files of 512 MB, 1024 MB, and 2048 MB sizes each. We decide to initiate the experiment with 2 shared files to correspond a bioinformatic reference and query files.

The results on next experiments were similar, so we just show the last case in Fig. 7. Considering the use of HEFT algorithm on a shared file system (NFS) using between 8 and 128 cores, the CPFL obtained up to 50% better makespan when 2 storage levels were used (local disk + Ramdisk). When only one storage level was used (local disk), CPFL was 15% faster.

Continuing with type (A) 50 synthetic workflows and only on real IBM cluster in Fig. 8 once again the results were similar considering 8, 16, 32, 64 and 128 cores. We present the results for 128 cores where the gain of makespan is up to 70% when shared input files is 2048 MB and 40% for 512 MB using 2 storage levels (Ramdisk + Local Disk), and gains of 20% for 2048 MB files and 12% for files of 512 MB when we use just local disk.

In Fig. 9 we can evaluate the impact of reading the same data file many times in the makespan of the multiworkflow execution. We extend the initial number of shared data files to each of the 5 batches of 10 workflows. Each batch now has from 2 to 4 and 8 shared files. For 128 cores and data files size of 2048 MB, we have a gain of up to 78% for 8 shared files on CPFL using Local Disk and Local Ramdisk approach versus a HEFT on NFS storage.

As we have found so far, the profits on synthetic workflows are between 12% and 78%, so it is necessary to contrast this

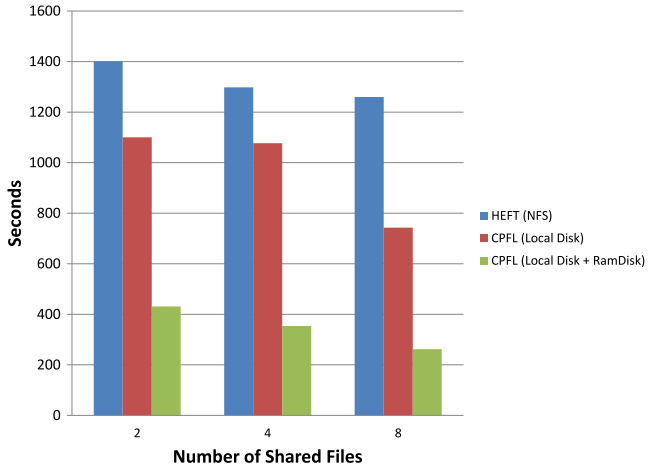


Fig. 9. Synthetic workflow makespan on 128 cores with many 2048 MB shared input files.

performance with well-known benchmarks as Montage and Epigenomics.

Using a set of workflows defined as (B) 10 Epigenomics workflows and (C) 10 Montage workflows. The workload was divided into 5 batches of 2 workflows each. We compared the execution of HEFT vs. CPFL algorithms. We also consider the impact of not using critical path (SCPFL) for the allocation of files in the storage hierarchy. In SCPFL case applications are assigned randomly to a resource but still using the storage hierarchy.

The results from Epigenomics and Montage were similars. We will present on Fig. 10 the results from Epigenomics. As we can evaluate the gain against NFS and 8 cores is 7.5% from 15 458 to 14 300 s and using Ramdisk and Local Disk as storage hierarchy on 128 cores gain is up to 68% reducing from 1209 to 384 s with CPFL and the gain is up to 38% from 649 to 1069 s by using local disk to local disk + ramdisk when we used SCPFL.

Of the 68% gain, CPFL has a 2% of the impact, due to the previous allocation of data files to the correct computation node avoiding waiting times while files are copied or moved from one storage to another. The local disk has an impact of 11%. Ramdisk has 41%. The combination of Local disk + Ramdisk has 46%, because of the high-speed storage at a higher level of the storage hierarchy systems.

Montage as well with 8 to 128 cores on real IBM cluster has a gain on NFS and 8 cores of 9.5% and using storage hierarchy combining Local disk and Ramdisk on 128 cores gain is up to 71%. SCPFL has a gain up to 19% from using local disk and using local disk + ramdisk on 128 cores, because not using a scheduler to determine which node has the data file, produce a slowdown at the gain even when the storage hierarchy is in use.

5.2. WorkflowSim cluster scenario

For our next experiment, we want to consider a 512-core cluster architecture. For that, we propose the use of WorkflowSim to evaluate CPFL scalability for hundreds of cores and for a large number of workflows executing at the same time in the system.

First, we validate WorkflowSim platform parameters, using the same workload of our previous real cluster scenario.

In Fig. 11 we show the behavior of type (B) group of 10 Epigenomics workflows on WorkflowSim. We can evaluate the error percentage on the simulation has a maximum 0.9% for CPFL. We believe that this error is due to the implementation of CPFL on the simulator because some parameters such weight of communication are not applied correctly to the scheduler comparing to CPFL implementation. The accuracy is calculated

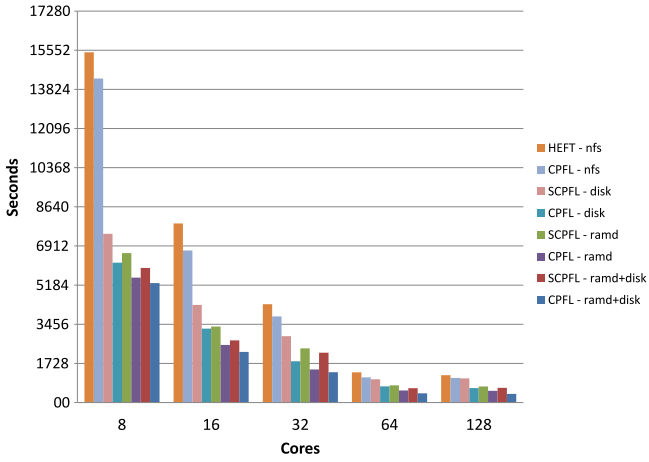


Fig. 10. Epigenomics makespan results in experimental cluster.

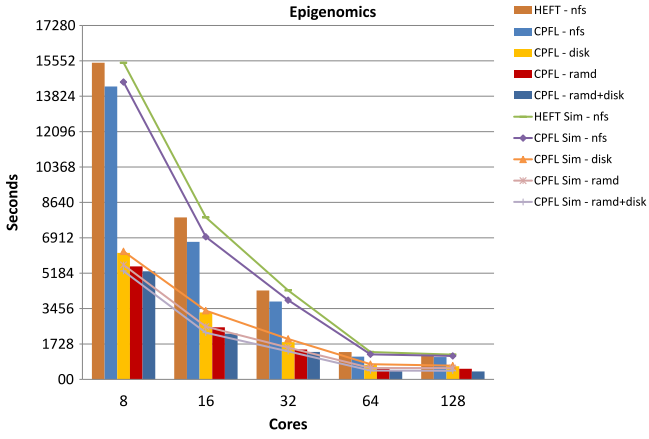


Fig. 11. Epigenomics WorkflowSim makespan simulation vs. real.

according to WorkflownSim formula Simulation Time over Real Time.

In Montage case, WorkflowSim has an error percentage of 2% for CPFL.

After we have set the parameters and validated the simulation conditions on WorkflowSim, we use the same platform for larger experiments.

Our objective is to evaluate CPFL scalability on a simulated modern cluster environment. We increased the amount of Epigenomics workflows from 10 to 20 to test how CPFL works with an increasing number of workflows and resources. We define a new workload of 20 workflows to provide a large task load for the simulated cluster. Additionally, the maximum parallel degree of Epigenomic was 5 and increasing the number of cores without increasing the number of workflows would not show any effect. Fig. 12 finally presents results of executing 20 Epigenomic workflows on a simulated cluster of 8, 16, 32, 64, 128, 256 and 512 cores. We compare against other states of art heuristics such as HEFT. As a result, we obtained a gain of 1.69% when we compare the use of HEFT and CPFL on NFS on 64 cores from 9099 to 8945 s. Using CPFL on local disk + Ramdisk the makespan has been reduced to 4210 s. Due to the new improvement, the gain is 53% against HEFT.

6. 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.

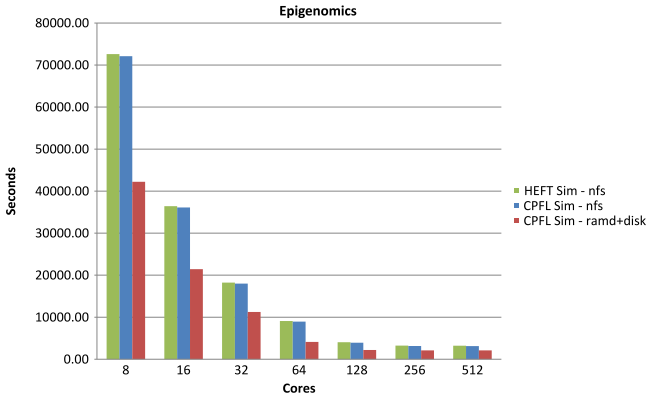


Fig. 12. Epigenomics CPFL simulation scaling to 512 cores versus HEFT algorithm.

We concentrated our efforts in studying disk I/O cluster bottlenecks. We characterized bioinformatics applications where some of them use the 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 presented a set of synthetic workflows based on bioinformatics applications and a set of well-known workflow benchmarks to test CPFL against other well-known scheduling techniques.

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

When data files grow to 2048 MB as we see on Section 5.1 the makespan starts to increase. This is due to the need of CPFL to find the location on storage hierarchy for large files. To improve the makespan we can introduce different types of storage such as operating system cache, SSD disks, and other new storage technology.

To evaluate the behavior of CPFL in larger systems, we used WorkflowSim. We configured the simulator to use a larger number of cores and storage levels. Increasing the number of cores will help us to determine the behavior of CPFL but currently, we are having some problems with the WorkflowSim code, preventing us from scaling further than 512 cores.

We are working to extend WorkflowSim to other several storage hierarchies such as cache Linux, SSD disks, distributed disks and distributed ramdisk.

Looking forward, as [41] propose disaster recovery for all data processing and storage with a "multi-purpose" approach to ensure that restored data can be fully recovered from multiple sites on. This could be integrated into the workflow management system. Due to our approach of keeping copies of data files on several storage hierarchy, we could ensure data redundancy on a future work. Currently we are using DRMAA [42] as an API to implement our scheduler independently of the distributed resource manager, this allow the use of a implement a state machine to ensure disaster recovery over several computation technology.

Finally, in Chang [43] they discuss security since hacking has been common in HPC, scheduling, and Cloud. Integration with classic security technology such as LDAP is ongoing and ready to integrate into the system, with LDAP technology we move the responsibility to the system admin that is in charge of the CCAF security layers. The CPFL scheduler is ready to be integrated into a scientific workflow manager like Galaxy [44] which is a web-based workflow manager widely used in the bioinformatics community that allows the use of LDAP security technology.

Table A.4
Scheduling policies profiling.

Policy	CPU (%)		Net/Total (MB)		Disk/Total (MB)		Mem usage (MB)	
	Util	Wait	Recv.	Send	Read	Write	Used	Free
HEFT NFS	94	6.32	13 255	3650	0.074	4.023	257–789	9981–10 337
CPFL NFS	96	5.43	12 848	3893	0.058	4.106	232–803	9923–11 004
CPFL Disk	98	2.42	2 373	239	1.838	7742	242–817	9846–10 801
CPFL Disk + Ramdisk	99	0.08	712	57	0.269	51.068	9674–11 325	122–1849

Table A.5
Batches profiling by nodes on CPFL Disk + RamDisk.

Batch	Nodes	Net/Total (MB)		Disk/Total (MB)		Mem usage (MB)	
		Recv.	Send	Read	Write	Used	Free
0	5, 9, 11, 16, 20, 21, 23, 27, 30	180	15	276	14 379	9674–11 280	123–1849
1	4, 9, 12, 15, 17, 18, 19, 24, 25, 31	203	17	275	16 071	9675–11 325	122–1848
2	1, 2, 7, 10, 14, 21, 22, 26, 29, 30	231	18	0.3	16 435	9674–11 247	270–1838
3	3, 4, 5, 6, 8, 13, 18, 23, 25, 28	231	18	275	17 442	10464–11 325	123–1086
4	9, 11, 12, 15, 17, 19, 20, 21, 24, 31	231.5	19	0.32	16 725	9675–11 211	281–1838

Acknowledgments

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.

Appendix

In order to substantiate the comparative effectiveness of the proposed heuristic, in Table A.4 we present the results of profiling the system according to the execution of workload of 5 batches of 10 workflows each with the corresponding 2 shared files of 2048 MB on each batch.

We show an example of how storage hierarchies helps to improve makespan by allocating data files previous to execution. That is, an initial profile of the storage performance. For CPFL compared to HEFT a small gain has been found when both are using NFS as main storage. Looking forward, when CPFL uses only local Disk, the network access is reduced on 70%. If CPFL uses Local Disk and Ramdisk the access to disk is reduced on up to 80%, from 13 255 MB to 712 MB on network data transfer at execution time. According to the experiment 2 shared data files of 2048 MB each generates at least 12 GB of output data file. The Mem Usage column reflects the total size of data files on a period of time corresponding to the makespan.

Table A.5 illustrates the case when 5 batches of 10 workflows are executed on a 32 nodes cluster. The table shows how CPFL Disk + Ramdisk uses resources. The whole 32 nodes are busy. We have parallel factor of 6 on each workflow, that is, 300 applications for 128 cores. Once again, two shared input files of 2048 MB generate 12 GB of shared temporal files. We cannot see significant network activity because the relevant resource access on CPFL are local storages such Local Disk and Ramdisk. Column Disk/Total, shows that in some cases the total size of data write is near 17 GB for each batch. Mem Usage column shows that each batch allocate 12 GB of data files to be accessed reducing the access time. As Ramdisk size is 6 GB and we do not have a distributed Ramdisk, this for sure, generate paging between Disk and Ramdisk, but we do not have the impact factor here.

Finally, Fig. A.13 has initial results that provides the idea of using a general purpose storage system. The impact of caching two shared input data files on local disk to be used by 10 applications. Caching shared data files previous to execution reduces the access time to disk and or low level storage systems. We get better performance when shared data files are bigger.

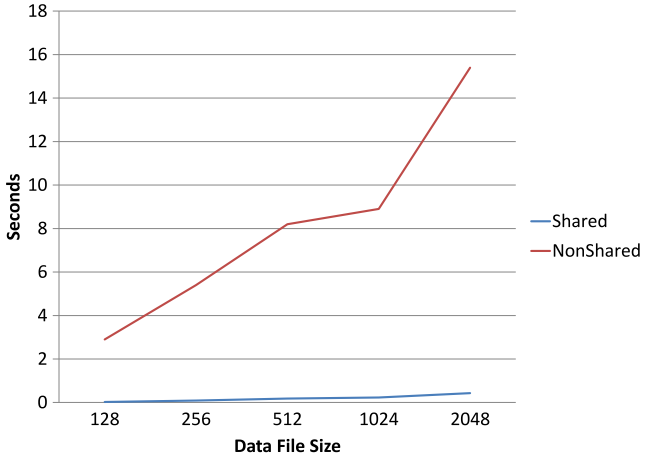


Fig. A.13. Impact of caching 2 shared data files on local disk.

References

[1] 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, J. Grid Comput. 13 (1) (2015) 95–113.

[2] G. Weiss, M. Pinedo, Scheduling: Theory, algorithms, and systems, 1995.

[3] H. Topcuoglu, S. Hariri, M.-Y. Wu, Task scheduling algorithms for heterogeneous processors, in: Heterogeneous Computing Workshop, 1999. HCW'99, Proceedings. Eighth, IEEE, 1999, pp. 3–14.

[4] L.F. Bittencourt, E.R. Madeira, Towards the scheduling of multiple workflows on computational grids, J. Grid Comput. 8 (3) (2010) 419–441.

[5] G.L. Stavrinides, H.D. Karatza, Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques, Simul. Model. Pract. Theory 19 (1) (2011) 540–552. http://dx.doi.org/10.1016/j.simpat.2010.08.010, URL http://www.sciencedirect.com/science/article/pii/S1569190X10001863.

[6] H. Zhao, R. Sakellariou, Scheduling multiple dags onto heterogeneous systems, in: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, IEEE, 2006, p. 14.

[7] U. Hönig, W. Schiffmann, 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, 2006, pp. 38–39.

[8] Z. Yu, W. Shi, A planner-guided scheduling strategy for multiple workflow applications, in: 2008 International Conference on Parallel Processing - Workshops, IEEE, 2008, pp. 1–8. http://dx.doi.org/10.1109/ICPP-W.2008.10, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4626773.

[9] T. N'Takpe, F. Suter, Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations, in: IEEE International Symposium on Parallel & Distributed Processing, 2009, IPDPS 2009, IEEE, 2009, pp. 1–8.

[10] F. Afrati, C.H. Papadimitriou, G. Papageorgiou, Scheduling dags to minimize time and communication, in: Aegean Workshop on Computing, Springer, 1988, pp. 134–138.

[11] M. Rahman, S. Venugopal, R. Buyya, A dynamic critical path algorithm for scheduling scientific workflow applications on global grids, in: IEEE International Conference on e-Science and Grid Computing, IEEE, 2007, pp. 35–42.
[12] Y. Gu, Q. Wu, Optimizing distributed computing workflows in heterogeneous network environments, in: International Conference on Distributed Computing and Networking, Springer, 2010, pp. 142–154.
[13] X. Wang, C. Olston, A.D. Sarma, R. Burns, Coscan: cooperative scan sharing in the cloud, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, 2011, p. 11.
[14] T. Wickberg, C. Carothers, 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, ACM, 2012, p. 5.
[15] C. Acevedo, P. Hernandez, A. Espinosa, V. Mendez, A data-aware multiworkflow cluster scheduler, in: Proceedings of the 1st International Conference on Complex Information Systems, SCITEPRESS, 2016, pp. 95–102.
[16] Proceedings on International Conference on Complex Information Systems Complexis 2016, (April 2016). URL http://eprints.soton.ac.uk/397888/.
[17] G.B. Berriman, E. Deelman, J.C. Good, J.C. Jacob, D.S. Katz, C. Kesselman, A.C. Laity, T.A. Prince, G. Singh, M.-H. Su, Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand, in: SPIE Astronomical Telescopes+ Instrumentation, International Society for Optics and Photonics, 2004, pp. 221–232.
[18] Y. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. (CSUR) 31 (4) (1999) 406–471.
[19] T. Yang, A. Gerasoulis, Dsc: Scheduling parallel tasks on an unbounded number of processors, IEEE Trans. Parallel Distrib. Syst. 5 (9) (1994) 951–967.
[20] G.-L. Park, B. Shirazi, J. Marquis, Dfrn: A new approach for duplication based scheduling for distributed memory multiprocessor systems, in: Parallel Processing Symposium, 1997. Proceedings., 11th International, IEEE, 1997, pp. 157–166.
[21] E. Ilavarasan, P. Thambidurai, Low complexity performance effective task scheduling algorithm for heterogeneous computing environments, J. Comput. Sci. 3 (2) (2007) 94–103.
[22] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682–694.
[23] L.F. Bittencourt, R. Sakellariou, E.R. Madeira, Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm, in: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, IEEE, 2010, pp. 27–34.
[24] H. Topcuoglu, S. Hariri, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274. http://dx.doi.org/10.1109/71.993206, URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=993206.
[25] J. Yu, R. Buyya, A taxonomy of scientific workflow systems for grid computing, ACM SIGMOD Rec. 34 (3) (2005) 44–49.
[26] R. Bolze, F. Desprez, B. Isnard, Evaluation of Online Multi-Workflow Heuristics based on List-Scheduling Algorithms, Gwendia report L. URL http://scholar.google.es/scholar?hl=es&q=+Evaluation+of+Online+Multi-Workflow+Heuristics+based+on+List-Scheduling+Algorithms&btnG=&lr=#0.
[27] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in grids, in: CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005, Vol. 2, IEEE, 2005, pp. 759–767.
[28] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.
[29] L.-C. Canon, E. Jeannot, R. Sakellariou, W. Zheng, Comparative evaluation of the robustness of dag scheduling heuristics, in: Grid Computing, Springer, 2008, pp. 73–84.
[30] M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of scientific workflows in the askalon grid environment, ACM SIGMOD Rec. 34 (3) (2005) 56–62.
[31] J. Bent, D. Thain, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, M. Livny, Explicit control in the batch-aware distributed file system, in: NSDI, Vol. 4, 2004, pp. 365–378.
[32] A. Siddiqa, A. Karim, V. Chang, Smallclient for big data: an indexing framework towards fast data retrieval, Cluster Comput. (2016) 1–16.
[33] T. Kosar, M. Livny, Stork: Making data placement a first class citizen in the grid, in: 24th International Conference on Distributed Computing Systems, 2004. Proceedings, IEEE, 2004, pp. 342–349.
[34] V. Chang, G. Wills, A model to compare cloud and non-cloud storage of big data, Future Gener. Comput. Syst. 57 (2016) 56–76. http://dx.doi.org/10.1016/j.future.2015.10.003, URL http://www.sciencedirect.com/science/article/pii/S0167739X15003167.
[35] V. Chang, Towards data analysis for weather cloud computing, Knowledge-Based Systems, 2017.
[36] K. Ranganathan, I. Foster, Design and evaluation of dynamic replication strategies for a high performance data grid, in: International Conference on Computing in High Energy and Nuclear Physics, Vol. 2001, 2001.
[37] K. Ranganathan, I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, in: 11th IEEE International Symposium on High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings, IEEE, 2002, pp. 352–358.
[38] W. Chen, E. Deelman, WorkflowSim: A toolkit for simulating scientific workflows in distributed environments, in: 2012 IEEE 8th International Conference on E-Science, e-Science 2012, 2012.
[39] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: 2008 Third Workshop on Workflows in Support of Large-Scale Science, IEEE, 2008, pp. 1–10.
[40] R.F. da Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community resources for enabling research in distributed scientific workflows, in: 2014 IEEE 10th International Conference on e-Science, Vol. 1, (e-Science), IEEE, 2014, pp. 177–184.
[41] V. Chang, Towards a big data system disaster recovery in a private cloud, Ad Hoc Networks 35 (2015) 65–82. special Issue on Big Data Inspired Data Sensing, Processing and Networking Technologies.. URL http://www.sciencedirect.com/science/article/pii/S157087051500147X.
[42] P. Troger, H. Rajic, A. Haas, P. Domagalski, Standardization of an API for distributed resource management systems, in: Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on, IEEE, 2007, pp. 619–626.
[43] V. Chang, Y.-H. Kuo, M. Ramachandran, Cloud computing adoption framework: A security framework for business clouds, Future Gener. Comput. Syst. 57 (2016) 24–41. http://dx.doi.org/10.1016/j.future.2015.09.031, URL http://www.sciencedirect.com/science/article/pii/S0167739X15003118.
[44] J. Goecks, A. Nekrutenko, J. Taylor, T.G. Team, Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.

César Acevedo was a DBA for Bussines Intelligence for Telecom companies, Paraguay. Studied Computer Science at Universidad Catolica de Asuncion. He received a M.Sc. in High Performance Computing, Information Theory and Security at Universitat Autonom de Barcelona in September 2013. Since 2012 working as a research fellow at the department of Computer Architecture and Operating Systems of the Universitat Autonoma de Barcelona.

Porfidio Hernández received the B.S., M.S. and Ph.D. in Computer Science from the Autonomous University of Barcelona (UAB) in 1984, 1986 and 1991, respectively. He is currently an Associate Professor of Distributed Systems at UAB. His research interests include Distributed Systems, prediction and scheduling in Cluster and Cloud environments.

Antonio Espinosa is an assistant professor the Computer Architecture and Operating Systems Department at the Universitat Autonoma de Barcelona. During the last 10 years, he has participated in several European and national projects related to bioinformatics and high-performance computing, in collaboration with a number of biotechnology companies and research institutions.

Víctor Méndez has a main background in big data management and distributed systems. He participated in scientific international projects in physics and bioinformatics like AGATA, LHCb, ENRM and EISCAT, collaborating with several more. Currently, he is Head of Software Development at Mind the Byte, a bioinformatics company specializing in computational drug discovery using a pay-per-use SaaS platform (Software as a Service). He also works in the Universitat Autònoma de Barcelona (UAB) as a part time professor. He holds the program chair in INSTICC conferences in the fields of cloud computing, big data, complexity and the internet of things, as well as editorial collaboration in some journals of high impact.