

Chapter 2

Workflow Scheduling Techniques for Big Data Platforms

Mihaela-Catalina Nita, Mihaela Vasile, Florin Pop and Valentin Cristea

2.1 Introduction

Today, almost everyone is connected to the Internet and uses different Cloud solutions to store, deliver, and process data. Cloud computing assembles large networks of virtualized services, such as hardware and software resources [1]. The use of cloud resources by end users is made in an asynchronous way and in many cases using mobile devices over different types of networks. Interoperability for such type of systems with the main aim to ensure dependability and resilience is one of the major challenges for heterogeneous distributed systems.

While cloud computing optimizes the use of resources, it does not (yet) provide an effective solution for processing complex applications described by workflows. Some example of such applications is hosting multimedia content-driven, and process tsunami (often in real-time) of content from heterogeneous sources, such as surveillance cameras, medical imaging devices, etc. The current need is an optimal and validated middleware framework and that can support end-to-end life-cycle operations of different multimedia content-driven applications on more standard cloud infrastructures [2].

Many scientific applications are defined as a set of ordered tasks that are linked by data dependencies. A workflow management system is used to define, manage, and execute these workflow applications on cluster, grid, cloud environments. In this context, a workflow scheduling strategy is used to map the task on the different resources [3, 4].

We live in the data age, and a key metric of present times is the amount of data that is generated anywhere around us. The largest scientific institution of present times, CERN near Geneva, Switzerland, produces in the Large Hadron Collider project over 30 PB of data per year (as of 2013) [5]. Thus the notion of Big Data,

M.-C. Nita · M. Vasile · F. Pop (✉) · V. Cristea
Computer Science Department, University Politehnica of Bucharest,
Bucharest, Romania
e-mail: florin.pop@cs.pub.ro

© Springer International Publishing AG 2016

F. Pop et al. (eds.), *Resource Management for Big Data Platforms*,
Computer Communications and Networks, DOI 10.1007/978-3-319-44881-7_2

a commonplace in all business discussions involving technology. Yet, its definition is not that clear. Big Data represents data sets of sizes larger than the common ability of traditional technologies to process given a certain service level agreement. And this last part brings us to a more meaningful definition: Big Data is the process of delivering decision-making insights. But, rather than focusing on people, this process uses a much more powerful and evolved technology, given the latest breakthroughs in this field, to quickly analyze huge streams of data, from a variety of sources, and to produce one single stream of human-level knowledge [6]. Nevertheless, in 2001, META Group (now Gartner), proposed a three-dimensional view regarding data growth challenges and opportunities, taking into consideration the increasing volume (the amount of data), the velocity (the speed of data in and out) and variety (the range of data types and sources) [7]. In 2012, Gartner updated this report as follows: Big Data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision-making, insight discovery and process optimization.¹ Lately, another key point—veracity is added by some organizations to make a strong case for the high need of accuracy of Big Data. We can extend this model to 8-V dimensions of Big Data: volume, velocity, variety, veracity, variability, visualization, volatile, and value [8]. We consider Grids and Clouds as suitable systems for Big Data platforms.

In this context, we face with the following assumptions for workflow scheduling. The challenges came from dynamic systems affected by faults. In this context of variety, the stimulating relationship between users, who require better computing services, and providers, who discover new ways to satisfy them, is the motivation to introduce future trends oriented on self-* capabilities [9]. For multimedia applications, real-time scheduling and processing are done on reliable and unreliable resources (an example is based on mixed processing based on satellite images [10, 11], live data streaming and sensor data for video surveillance). The main challenges are to ensure deadlines (very important in real-time interaction), budget (pay-per-use Cloud model), energy consumption (battery saving for mobile devices), and QoS (to guarantee SLA) for complex workflow applications.

When addressing the problem of adaptive workflow scheduling we first need to investigate the current solutions for managing such workflow applications. With specific focus on scientific workflows we can find some tools designed by the research community to help the scientists address/investigate/simulate issues from every area like: astronomy, bioinformatics, earth science, chemistry. It is important to have a global picture of the current tools in order to properly choose the one that suits perfectly in our research area [12, 13].

The chapter is organized as follow. First, we discuss the workflow scheduling algorithms and techniques in grid and cloud. We present the strengths and weaknesses of several existing scheduling strategies and we make a critical analysis of workflow management systems.

¹<http://www.gartner.com/it-glossary/big-data/>.

2.2 Workflow Scheduling in Distributed Systems

We are facing with a strong development of various technologies leading to complex applications, which are able to process Big Data sets and execute different experiments/tasks on distributed systems. These applications are covering important aspects of everyday life: health, education, astronomy, research engineering, etc. and are described by a number of interdependent tasks called workflows. Scientific workflows represent the automation of a scientific process in which tasks are organized based on their control and data dependency.

In this context, distributed systems are offering several advantages, such as: utilizing geographically distributed resources, increasing throughput, reducing costs by not investing in proprietary resources and using the shared ones, engaging various scientific teams with different expertise.

A workflow is described by connecting multiple tasks according to their dependency (execution dependency or data dependency). This pool of interconnected tasks may take two shapes: that of a directed acyclic graph (DAG) or a non-DAG, the difference consisting in the existence of repeating tasks.

The DAG structure can be categorized as sequential, parallel and choice, while the non-DAG structure is adding one more structure type: iteration. In a sequential structure, the tasks are ordered in a serial manner and one task starts only after the previous one ended. In a parallel structure, the tasks may be computed simultaneous and in a choice structure, the decision of following a certain path in the graph is done at runtime. In the iteration structure, several tasks of the workflow may be repeated. A workflow may be described by all the presented structure types simultaneously [14].

2.2.1 Workflow Scheduling Algorithms and Techniques in Grid

There are two main classes of workflow scheduling: best effort and QoS constraints-based scheduling [15]. The best-method class wants to minimize the makespan and ignores other constraints, like budget, energy, etc. On the other hand the QoS constrained class, as the name suggests, attempts to minimize cost/time under some QoS metrics.

2.2.1.1 Best-Effort-Based Workflow Scheduling

Best-effort-based workflow scheduling algorithms are more specific for community-based environments, like grids. In this type of environments, there are factors like elasticity or cost that does not count and the main focus is on the execution time. This class of scheduling has as target to complete execution at the earliest time. Best-effort-based scheduling algorithms have two different approaches: heuristics

based or meta-heuristics based. The heuristic approach is to develop a scheduling algorithm for a particular type of problem, while the meta-heuristic-based approach is to develop an algorithm based on a meta-heuristic method which provides a general solution for a specific class of problems.

Individual task scheduling method, as the name suggests it makes scheduling choices based on a single task at a time. It is not aware of the general context. The Myopic algorithm [16] implements this method and it schedules an unmapped ready task to the resource that is expected to complete the task earliest, until all tasks have been scheduled.

List scheduling prioritizes workflow tasks and schedules the tasks based on their priorities. It includes two phases: task priority and resource allocation. The list scheduling models can be categorized in three groups: batch mode scheduling, dependency mode scheduling, and a hybrid version: Batch-Dependency Mode scheduling.

Batch mode scheduling algorithms intent to schedule parallel independent tasks on a pool of resources. Since the number of resources is much less than the number of tasks, the tasks need to be scheduled on the resources in a certain order [17]. This priority is made through the following algorithms: Min-Min, Max-Min, and Suffer.

Min-Min heuristic schedules sets of independent tasks iteratively. In each iterative step, it computes the ECT (Early Completion Time) of each task on its every available resource and obtains the MCT (Minimum Estimated Completion Time). The task with minimum MCT is chosen to be scheduled first. The task is assigned on the resource which is expected to finish it at first.

Max-Min heuristic is similar to the Min-Min heuristic, but it sets high scheduling priority to tasks which have long execution time.

Sufferage sets high scheduling priority to tasks whose completion time by the second best resource is far from the first which can complete the task at earliest time. This method may have optimal results in heterogeneous environments.

Dependency mode intends to provide strategies to map workflow tasks on heterogeneous resources based on analyzing the dependencies of the entire task DAG. Unlike batch mode algorithms, it ranks the priorities of all tasks based on the whole application context.

The Heterogeneous Earliest Finish Time (HEFT) algorithm proposed in [18] by Topcuoglu et al. has been applied by the ASKALON project and it first calculates average execution time for each task and average communication time between resources of two dependent tasks. Then, each task receives a rank value which is computed in a recursive manner based on the rank value of the following dependent tasks. So, the exit task in the graph will have the smallest rank value, as being the average execution time. The tasks previous the exit task will have their average execution time + the maximum ([communication time from a resource to another resource] + [the rank value of the successor]). The task with the highest priority will be scheduled first.

Sakellariou and Zhao [19] proposed a *hybrid heuristic for scheduling DAGs* on heterogeneous systems. The heuristic combines dependency mode and batch mode. It first computes rank values of each task and ranks all tasks in the decreasing order of their rank values. Then it creates groups of independent tasks. Each group will

have a group number based on the rank values of the group tasks. Then it schedules tasks group by group and uses a batch mode algorithm to reprioritize the tasks in the group.

Cluster-based scheduling and duplication-based scheduling aim to avoid the communication time of the data for interdependent tasks, such that it is able to reduce the overall execution time. The cluster-based scheduling clusters tasks and assign tasks of the same cluster to the same resource. The duplication-based scheduling use the idling time of a resource to duplicate some parent tasks and it schedules them on other resources. Bajai and Agrawal [20] proposed a task duplication-based scheduling algorithm for network of heterogeneous systems (TANH). This algorithm combines both cluster-based scheduling and duplication-based scheduling. It first traverses the DAG to compute parameters of each node including earliest start and completion time, latest start and completion time, critical immediate parent task, best resource, and the level of the task. Afterwards, it clusters tasks based on these parameters and it scales down the number of clusters until it is less or equal than the number of resources. In case of number of clusters being less than the number of resources, it utilizes the idle times of resources to duplicate tasks and rearrange tasks in order to decrease the overall execution time.

Genetic Algorithms (GAs) [21] provide robust search techniques that allow an optimal solution to be derived from a large search space by applying the principle of evolution. It first creates an initial population consisting of randomly generated solutions and then it applies genetic operators (selection, crossover and mutation). Then the individuals are selected based on their fitness values and included in the next selection steps. These steps are repeated until an optimal solution is found. The art of scheduling consists in finding the proper fitness function. As an example of such function designed for scheduling [22] is $f(x) = C_{max} - FT(I)$, where C_{max} is the maximum completion time observed so far and $FT(I)$ is the completion time of the individual I . While trying to minimize the completion time, the individuals with a larger fitness value will be selected for the next steps.

Simulated Annealing (SA) [23] is inspired by the Monte Carlo method for statistically searching the global optimal between several local optimal. The concept is taken from the annealing process, which repeats the heating and slowly cooling of a structure. The input of the algorithm is an initial solution which is constructed by assigning a resource to each task randomly. Then, based on an acceptance rate, the solutions are selected for the next step. At each iterative step the acceptance is decreased.

2.2.1.2 QoS-Constraint-Based Workflow Scheduling

When talking about QoS Constraints algorithms, in general, we are talking about two different perspectives: user perspective and scheduler perspective. If the user perspective refers to QoS of the entire workflow, the scheduler perspective refers to the QoS of the task. The scheduler may assure a QoS for the entire workflow only if the QoS of each individual task is assured.

Many workflow applications have a tight deadline so the deadline constrained scheduling algorithms are designed mainly for these types of applications. However being in pay-per-use environment such as cloud, it is also important to minimize the cost. In this context, this class of scheduling classes pays attention to the time framework but without ignoring the cost.

The *backtracking heuristic* developed by Menasce and Casalicchio [24] assigns available tasks to least expensive computing resources (in case of many available tasks, it assigns the most CPU intensive task to the fastest resource). The procedure is repeated until all tasks are mapped. After each iterative step, the execution time of current assignment is computed and in case of the execution time exceeding the time constraint, the heuristic backtracks the previous step and remove the least expensive resource from its resource list and reassigns tasks with the reduced resource set.

Another deadline constraint heuristic is the *deadline distribution* [25] heuristic which partitions a workflow and distributes the overall deadline into each task based on their workload and dependencies. After deadline distribution, the entire workflow scheduling problem has been divided into several subtask scheduling problems. A sub-deadline can be also assigned to each task based on the deadline of its task partition.

The *Budget Constrained* scheduling puts accent on the cost constraints while minimizing the execution time. LOSS and GAIN scheduling approach [26] adjusts a schedule which is generated by a time optimized heuristic and a cost optimized heuristic to respect budget constraints. There are two situations:

1. Total execution cost generated by time optimized schedule is greater than the budget; the LOSS approach is applied: gain a minimum loss in execution time for the maximum money savings by amending the schedule to satisfy the budget.
2. Total execution cost generated by a cost optimized scheduler is less than the budget, the GAIN approach is applied in order to use the surplus for decreasing the execution time: gain the maximum benefit in execution time for the minimum monetary cost, while amending the schedule.

In [27] a *scheduling data-intensive workflows onto storage-constrained distributed* method is proposed. Important improvement in storage use for workflow data is to add a cleanup job algorithm to erase data files when they are not longer in use or required by current task or any other task in the workflow process. If the compute tasks are mapped to many resources then the data file is also replicated on all resources thus the cleanup jobs/tasks are added per resource basis. In some situations the data file required by a task comes from another resource and must not be deleted by the algorithm at the source before its transferred to the child resource or task. Cleaning up files on the workflow when having multiple file dependencies is challenging as the algorithm inserts cleanup jobs along the executable workflow and cleans up along the executions. This can generate for complex workflows greater number of cleanup jobs than the compute tasks itself. More efficient is to have an algorithm for storage aware when workflow is mapped considering the overall storage requirements, minimizing also the requirements. Thinking at an efficient execution of the tasks, these

must be mapped around the workflow taking in consideration the execution time and using a resource with ample disk space. Main idea is to consider space requirement and then performance of the resource when allocating the tasks. The algorithm can be split in a 3-phase algorithm:

1. identification of the resources capable to accommodate the data files;
2. task allocation based on the shortest run time for that task; and
3. cleanup any unnecessary files indicated by the cleanup jobs.

In [28] the problem of *scheduling multiple workflows* is addressed: how to better plan multiple workflows instead of merging them as previous solutions proposed. The proposed solution has three components:

- (a) DAG Planner: assigns each job local priority (HEFT-based priority), manages the job interdependence and submits the jobs to the Job Pool;
- (b) Job Pool an unsorted list with all jobs waiting to be scheduled;
- (c) Executor reprioritizes the job into the Job Pool.

Each workflow has its own DAG Planner. Each DAG planner sends only independent jobs at a time. Once it has finished the execution, the DAG planner is notified and it will send the successor of that job. At the Executor level there are two types of priority for two different cases:

- (a) If jobs are from the same DAG planner, the Highest Rank first rule will be applied (HEFT);
- (b) If jobs are from the different DAG planners, the Lowest Rank first rule will be applied in order to avoid the starvation of the exit jobs of some DAG planner.

2.2.2 *Workflow Scheduling Algorithms and Techniques in Cloud*

One of the main advantages of moving to the cloud is application scalability, which allows real-time provisioning of resources to respect service level agreements (SLAs)/application requirements. This enables workflow management systems (WMS) to support real-time provisioning instead of advanced reservations. In this context, workflow scheduling algorithms will need to adapt and assure the agreed level of QoS.

As concluded in [29] the main requirements for Cloud workflow scheduling are: satisfaction of QoS requirements for individual workflow instances, minimization of the running cost, ability of assigning fine-grained QoS to facilitate SLA management and good scalability. These conclusions lead to a two level workflow scheduling: service-level scheduling and task-level scheduling. The service level is responsible for the first and third objective while the task-level is responsible for the task VM optimization process (second and forth objective). The service level is a global scheduler that identifies the resources needs, makes provisioning and the task-level is a local

scheduler which implements the optimal scheduling plan for the given workflow on the resources obtained.

In [30] Meng Xu et al. propose a multiple QoS constrained scheduling strategy of multiple workflows for cloud computing (MQMW) which has a similar architecture with the one proposed in [28], but different names: *Preprocessor*, *Scheduler* and *Executor*.

The *Preprocessor* will compute the following attributes for the received tasks:

- the available service number;
- the covariance for time and cost;
- the time quota: the time limit when the task is executed;
- the cost quota: the cost limit when the task is executed;
- the time surplus of the workflow: the difference between the time attribute of QoS and the finish time of the workflow if all the resources are available;
- the cost surplus of the workflow: the difference between the cost attribute of QoS and the cost of the workflow if all the resources are available.

Afterwards, the *Preprocessor* inserts the ready tasks into the queue and for the first time only entry tasks will be submitted. After the notification given by the *Executor* that a certain job has finished, the dependent tasks will be also submitted. The *Scheduler* will reorder the jobs in the list and it will allocate a job to the optimal resource. When a task will be finished the *Executor* will notify the *Preprocessor* about the task completion status.

The *Scheduling Strategy* is the following:

1. the task with minimum available service number should be scheduled first (that the task would have not available services, if other tasks are scheduled first);
2. the tasks which belong to the workflow with minimum time surplus and cost surplus should be scheduled first;
3. the tasks with minimum covariance should be scheduled first. The covariance describes the strength of the correlation between time and cost. The minimum covariance means when the time decreasing a definite value, the cost will increase mostly. So the task should be scheduled first. Otherwise, we should pay more or the time would increase more.

This algorithm was evaluated in parallel with the one described in [28] and the results shows that the previous one improves the execution time of the workflow no matter of the costs. However MQMW respects all the QoS constraints.

In [31] a multi-objective heterogeneous earliest finish time algorithm. The method called MOHEFT is an extension of the Heterogeneous Earliest Finish Time (HEFT) algorithm and intends to provide an optimal solution to the problem of makespan and energy reduction. In this context, it is not always possible to find a solution that minimizes both makespan and energy consumption so they introduced the concept of dominance. A solution $\times 1$ dominates a solution $\times 2$ if the makespan and energy consumption of $\times 1$ are smaller than those of $\times 2$.

In this context, two solutions are said to be nondominated whenever none of them dominates the other (i.e. one is better in makespan and the other in energy

Scheduling Method	Algorithm
Heuristics Based Approach	
Individual Task Scheduling	Myopic
List Scheduling	Min-Min Max-Min Sufferage HEFT Hybrid
^ Batch Mode	
^ Dependency Mode	
^ Batch – Dependency Mode	
Cluster Based Scheduling	
Duplication Based Scheduling	THAN
Meta-Heuristics Approach	
Genetic Algorithms	
Simulated Annealing	

Fig. 2.1 Overview of the best-effort workflow scheduling models

consumption). The set of optimal nondetermined solutions are called Pareto Front. Similar to HEFT, MOHEFT ranks first the tasks and then instead of creating an empty solution as HEFT does, it creates a set *S* of *K* empty solutions. Afterwards, the mapping phase begins in which MOHEFT iterates first over the list of ranked tasks. The idea is to extend every solution by mapping the tasks onto all possible resources. This strategy results in an exhaustive search if there is any restriction taken into consideration. Only the best *K* trade-off solutions from the temporary set are kept. A solution belongs to the best trade-off if it is not dominated by any other solution and if it contributes to the diversity of the set. The diversity of the set is described as the highest crowding distance (the area surrounding a solution where no other trade-off solution is placed nearby).

An overview of the best-effort workflow scheduling models is presented in Fig. 2.1.

2.2.3 Scheduling Methods

In the following table, we present a critical analysis of existing methods for workflow scheduling, by highlighting the strengths and weaknesses and a short description for each presented method.

A Planner-Guided Scheduling Strategy for Multiple Workflow Applications and Dynamic scheduling multiple DAGs [28]

<i>Description</i>	Dynamic scheduling multiple DAGs; Poisson distribution of the arrival jobs; Custom ranking system (DAG path for example); Highest rank first between jobs from the same DAG; Lowest rank first between jobs from different DAGs, resulting that, applications that are closing to finish are not starving for resources (the last jobs will have lower priorities)
<i>Strengths</i>	Multiple DAGs scheduling; hybrid priority (improvement of highest rank first in case of multiple DAGs)
<i>Weaknesses</i>	If lower rank workflows are coming continuously, the higher rank task scheduling will be postponed; not focused on deadline and budget constraints; not SLA awareness at the Executor level

Immediate Mode: Individual Task Scheduling [16]

<i>Description</i>	Myopic algorithm; The best-effort scheduling strategy (more suitable for grids); Individual task scheduling; Scheduling decision is made only for one task at a moment; Task is mapped at the resource that is expected to finish the task first
<i>Strengths</i>	Works fine for short tasks and a small number of requests/second
<i>Weaknesses</i>	Not suitable for cloud environments; It does not apply any logic in tasks priority (FIFO rule); which will generate starvation in case of longer tasks

Batch Mode Scheduling : Min-Min, Max-Min, and Sufferage strategies [17]

<i>Description</i>	Batch mode scheduling strategy (best effort/list scheduling); Provides a strategy to map a number of tasks (T) on a number of resources R , where $T \gg R$; MIN-MIN a task having a min MECT (Minimum Estimated Completion Time) will be scheduled first; MAX-MIN task with max MECT will be executed first; SUFFERAGE priority based on the suffrage value (the difference between 1st ECT (earliest completion time) and 2nd ECT)
<i>Strengths</i>	Experimental results shows that generally MIN-MIN outperforms MAX-MIN; MAX-MIN may be better in cases of having much more short tasks than longer tasks (it will not generate starvation for longer tasks); Suffrage performs better in heterogeneity environments where there is a remarkable difference between resources performance
<i>Weaknesses</i>	Application type-dependent strategy; best effort; suitable only for grid

Dependency Mode Scheduling HEFT [18]

<i>Description</i>	Graph dependency is analyzed before scheduling; It ranks the priorities of all tasks at a time 2 metrics: average execution time and average communication time; All the tasks are ordered based on a rank (computed recursively based on previous ranks)
<i>Strengths</i>	Based on heterogeneous resources
<i>Weaknesses</i>	Mean value is the only best practice; Best effort; Suitable only for grid

Batch-Dependency Mode [19]

<i>Description</i>	Hybrid method between batch and dependency methods; Independent tasks are added into separate groups; Group rank ordering
<i>Strengths</i>	Parallel execution of multiple different jobs
<i>Weaknesses</i>	Not SLA aware; Suitable for grid environments

Cluster-based scheduling [20]	
<i>Description</i>	Avoid transfer communication cost; tasks are grouped in clusters; one cluster is scheduled on the same resource
<i>Strengths</i>	Avoids data transfer cost
<i>Weaknesses</i>	Difficult to respect deadline constraints
Duplication-based scheduling [20]	
<i>Description</i>	Avoid transfer communication cost; Use the idle time of an resource to schedule parent tasks (task duplication)
<i>Strengths</i>	Good in a heterogeneous environment where the machine performance is unknown
<i>Weaknesses</i>	Parallel job scheduling.
Hybrid method of cluster and duplication scheduling [20]	
<i>Description</i>	Cluster tasks based on earliest start and completion time, latest start and completion time, critical parent task and best resource; If the number of clusters $< R$ (resources): duplication; If nr. of clusters $> R$, scaling down by merging some clusters
<i>Strengths</i>	Avoids data transfer cost; Good in a heterogeneous environment where the machine performance is unknown
<i>Weaknesses</i>	Difficult to respect deadline constraints; Merging metrics
Genetic Algorithms meta-heuristics [21]	
<i>Description</i>	Generates new solutions by modifying currently known good solutions
<i>Strengths</i>	Optimized solution
<i>Weaknesses</i>	Its limitations depends on the fitness valuation function
Simulated Annealing SA meta-heuristics [23]	
<i>Description</i>	Generates new solutions by modifying currently known good solutions
<i>Strengths</i>	Optimizes the solution
<i>Weaknesses</i>	Its limitation depends on the acceptance function reduction value
Backtracking [24]	
<i>Description</i>	Assign available tasks to least expensive resources; Largest computational demand to fastest resource; After each step, the execution time of the current assignment is computed; if its exceed the time constraint, the task will be reassigned
<i>Strengths</i>	Deadline assurance
<i>Weaknesses</i>	SLA awareness; Multiple constraints; Dynamic environments

Deadline distributions [25]	
<i>Description</i>	Synchronization task (multiple parents) versus simple task; Grouped in sub-workflows with different deadlines; Distributed deadline
<i>Strengths</i>	Assures the deadline commitment at a granular level (the smallest unit task)
<i>Weaknesses</i>	SLA awareness; Multiple constraints; Dynamic environments
LOSS and Gain [26]	
<i>Description</i>	Gain a minimum loss in the execution time while maximizing the cost savings; First is applied a time optimization heuristics
<i>Strengths</i>	Budget constraint; assurance
<i>Weaknesses</i>	SLA awareness; Multiple constraints; Dynamic environments
MQMW [30]	
<i>Description</i>	Scheduling based on the following metrics: service available number, time surplus, workflow surplus, the covariance for time and cost
<i>Strengths</i>	Multiple workflow; Multiple QoS successfully respected
<i>Weaknesses</i>	It is not very clear what is the algorithm behavior in case of having a number of tasks respecting the conditions for prioritization greater than the number of available resources; Speed; Scalability
MOHEFT—Multi-objective energy-efficient workflow scheduling using list-based heuristics [31]	
<i>Description</i>	Extends HEFT by finding the optimal solution and respecting multiple objectives
<i>Strengths</i>	Multiple objectives: energy and time; Interesting to scale for multiple objectives: cost, energy and time
<i>Weaknesses</i>	Multiple workflows problem

2.3 Workflow Modeling and Existing Platforms

This section presents a detailed picture of the major tools used for workflow management.

Pegasus [29] is a Workflow system that can take a workflow description, transform it in an executable sequence of jobs and map it on a local machine, cluster, Condor Pool, or a cloud (Amazon EC2, Google Cloud Storage). It is developed since 2001 and the last release was in May 2015. Pegasus has been used in several scientific areas including bioinformatics (DNA sequencing, Epigenomics, etc.), astronomy (Galactic Plane—NASA Collaboration; Montage—Caltech, etc.), earthquake science, gravitational wave physics, and ocean science. Pegasus major components (Fig. 2.2):

1. *Mapper*—transforms the abstract workflow definition into an executable set of dependent jobs. The final will find the appropriate software, data, and computational resources required for workflow execution.
2. *Execution Engine*—executes in the specific order the tasks defined in the workflow. This component relies on the compute, storage, and network resources defined in the executable workflow to perform the necessary activities.

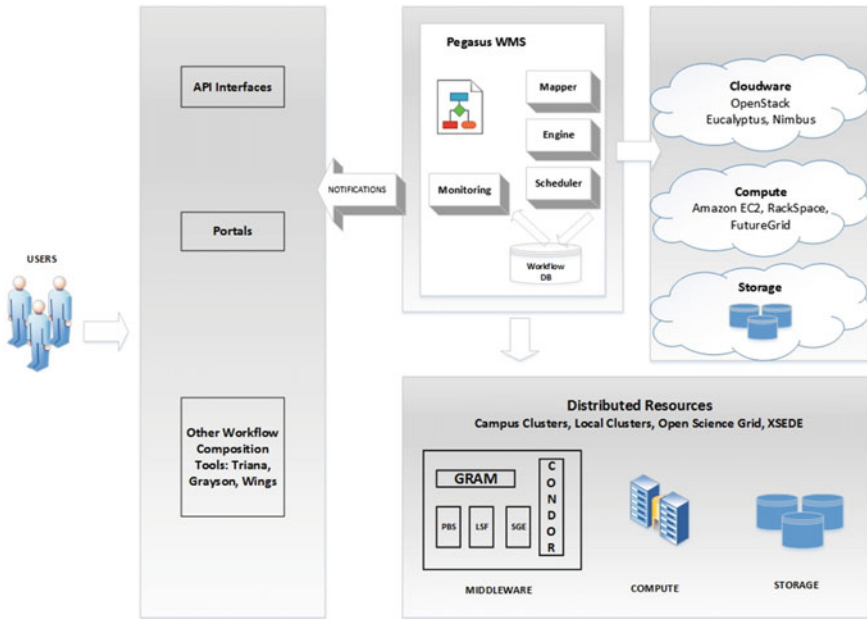


Fig. 2.2 Pegasus components

3. *Task Manager*—manages single workflow tasks, by supervising their execution on local or remote resources.

Workflows successfully execution is based on the information gathered from the following components:

1. Replica Catalog: looks for input and output data locations;
2. Transformation Catalog: looks for executables locations: binary files;
3. Site Catalog: looks for the environment infrastructure.

Pegasus is also able to make decisions in order to improve the overall performance: cluster small jobs together, data reuse (identical jobs, different workflows). Regarding the subject that is treated in this report, the scheduling problem, Pegasus by default implements the following policies:

1. Random—the sites are randomly chosen; default option.
2. Round Robin—jobs will be assigned in a round robin manner amongst the sites that can execute them. A site cannot execute all types of jobs so the round robin scheduling will be applied on a sorted list of sites. The sorting is done based on the number of jobs a particular site has been assigned in that job class so far. If a

job cannot be run on the first site in the queue (due to no matching entry in the transformation catalog), it goes to the next one and so on.

3. **Group**—jobs are grouped and a specific group will be assigned to the site that can execute them. A job will be put into a specific group based on a profile key group from the DAG. The jobs that do not have the profile key associated with them, will be put in the default group. The jobs in the default group are handed over to the “Random” Site Selector for scheduling.
4. **HEFT**—HEFT processor scheduling algorithm is used to schedule jobs in the workflow to multiple grid sites. The implementation assumes default data communication costs when jobs are not scheduled on to the same site. The runtime for the jobs is specified in the transformation catalog by associating the Pegasus profile key runtime with the entries. The number of processors in a site is picked up from the attribute `idle-nodes` associated with the job manager of the site in the site catalog.
5. **NonJavaCallout**—it will call out to an external site selector. A temporary file is prepared containing the job information that is passed to the site selector as an argument while invoking it. The path to the site selector is specified by setting the property `pegasus.site.selector.path`. The environment variables that need to be set to run the site selector can be specified using the properties with a `pegasus.site.selector.env.prefix`. The target sites used in planning are specified on the command line using the `sites` option to `pegasus-plan`. If not specified, then it will pick up all the sites in the Site Catalog as candidate sites and it will map a job on a specific site only if it finds an installed executable on that site.

Taverna [32] is an open-source Java-based workflow management system developed at the University of Manchester with the main target in supporting the life sciences community (biology, chemistry, and medicine) to design and execute scientific workflows and support research experiments. However, it can be applied to a wide range of fields since it can invoke any web service by simply providing the URL of its WSDL document. In addition to web services, Taverna supports the invocation of local Java services (Beanshell scripts), local Java API (API Consumer), R scripts on an R server (Rshell scripts), and imports data from a CVS or Excel spreadsheet. Taverna main components are:

- **Taverna Engine**—enacting workflows.
- **Taverna Workbench**—client application; users graphically create, edit, and run workflows on a desktop computer.
- **Taverna Server**—users set up a dedicated server for executing workflows remotely.
- **A Command Line Tool**—quick execution of workflows from a command prompt.

Triana [33] is a Java-based scientific workflow system, developed at the Cardiff University, which combines a visual interface with data analysis tools. It can con-

nect heterogeneous tools (e.g., web services, Java units, and JXTA services) in one workflow. Triana uses its own custom workflow language, although it can use other external workflow language representations, such as Business Process Execution Language (BPEL), available through pluggable language readers and writers.

One of the most powerful aspects of Triana on the other hand is its graphical user interface. It has evolved in its Java form for over 10 years and contains a number of powerful editing capabilities, wizards for on-the-fly creation of tools and GUI builders for creating user interfaces. Triana's editing capabilities include: multilevel grouping for simplifying workflows, cut/copy/paste/undo, the ability to edit input/output nodes (to make copies of data and add parameter dependencies, remote controls or plug-ins), zoom functions, various cabling types, optional inputs, type checking, and so on. Triana may generate Pegasus input files.

Kepler [34] is a Java-based open-source software framework providing a graphical user interface and a run-time engine that can execute workflows either from within the graphical interface or from a command line. It is developed and maintained by a team consisting of several key institutions at the University of California and has been used to design and execute various workflows in biology, ecology, geology, chemistry, and astrophysics.

Askalon [35] is an application development and runtime environment, developed at the University of Innsbruck, which allows the execution of distributed workflow applications in service-oriented Grids. Its SOA-based runtime environment uses Globus Toolkit as Grid middleware. Workflow applications in Askalon are described at a high level of abstraction using a custom XML-based language called abstract grid workflow language (AGWL).

The Askalon architecture includes the following components:

- *Resource broker*—responsible of the resources negotiation and reservation in Grid.
- *Resource monitoring*—rule-based monitoring; monitors Grid resources.
- *Information service*—discovery, organization, and maintenance of resources and data.
- *Workflow executor*—dynamic deployment and fault-tolerant execution of activities in the Grid nodes.
- *Metascheduler*—workflow applications mapping in the Grid.
- *Performance prediction*—estimates execution time of atomic activities and data transfers; Grid resource availability.
- *Performance analysis*—unifies the performance monitoring, instrumentation, and analysis for Grid applications; supports the interpretation of performance bottlenecks.
- *Askalon Scheduler*.

One of the Askalon architecture components that represents interest for us is the MetaScheduler and its architecture described in Fig. 2.3. The Scheduler consists of two major components: Workflow Converter and Scheduling Engine. Event Generator is a future extension for increasing dynamicity in workflow processing.

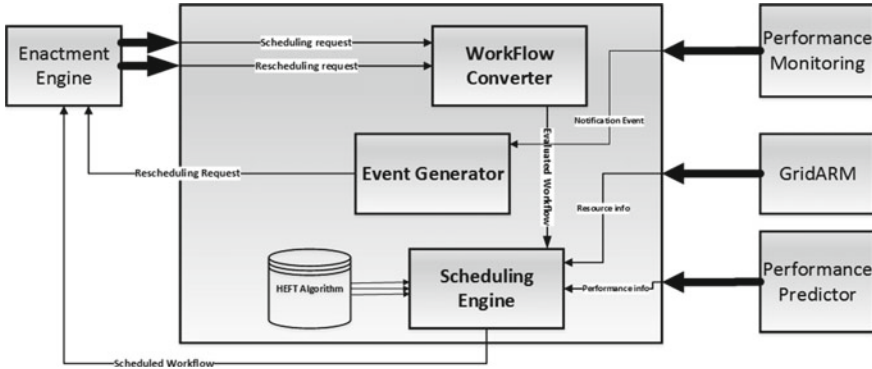


Fig. 2.3 Askalon components

The Workflow Converter is responsible for transforming all the sophisticated workflow graphs to simple DAGs. The Scheduling Engine for scheduling workflows into specific resources. It is based on a plug-in architecture, where different scheduling algorithms can be implemented. By default the HEFT algorithm is chosen as the primary scheduling algorithm for Askalon.

Karajan [36] is a JAVA written system that allows users to compose workflows through an XML scripting language and a custom language, called K, which is more user friendly. Both languages support hierarchical workflow descriptions based on DAGs and have the ability to use instructions such as if/while order to easily express concurrency. Also, it can be based on Grid tools such as Globus GRAM for distributed/parallel execution of workflows. The architecture of the Karajan framework contains the following components:

- *Workflow engine*—interacts with high-level components (GUI module for describing the workflows) and monitors the execution.
- *Checkpointing subsystem*—checkpoints the current state of the workflow.
- *Workflow service*—allows the execution of workflows; specific libraries enables the workflow engine to access specific functionalities.

2.4 Analysis of Workflow Management Systems

In the following table, we present a critical analysis of existing workflow management systems, by highlighting the strengths and weaknesses and a short description for each presented method.

WMS	Short description	Strengths	Weaknesses
Taverna	DAG based; Graphical specification	User-friendly interface for workflow description; easy to use by nontechnical scientists in their simulations	Relies on the user to make the choices of resources for mapping; It doesn't offer integration with public cloud environments; Adaptive workflows
Triana	Non-DAG based; Graphical specification	Graphical User Interface; Pegasus interoperability; Tool for editing/creating workflows; Workflow rewriting: creating sub-workflows that execute and feed back into the main workflow	Job submission made through GridLab GAT, which can make use of GRMS, GRAM or Condor for the actual job submission; Passive approach in case of a failure (it informs the users)
Pegasus	DAG based; Language specification	Focus on the mapping and execution capabilities and leave the higher level composition tasks to other tools; Amazon EC2/Google Cloud Support; Support for optimization decisions	It does not offer support for adaptive workflows
Kepler	Non-DAG based; Graphical specification	Graphical workflow specification; Both workflow specification support and execution engine; Local/Web/Grid services; Fault tolerance—smart rerun; Adaptive workflows—workflows can modify themselves during execution	Relies on the user to make the choices of resources for mapping; Adaptive Workflows; It doesn't offer integration with public cloud environments
Askalon	DAG; Language and graphical specification	Graphical User Interface; Support for optimization decisions; Complex Fault tolerance mechanism (checkpointing, task-level recovery)	Custom description language (AGWL); Common Public Cloud integration; Adaptive Workflows
Karajan	DAG; Language and graphical specification	Graphical User Interface; Support hierarchical workflows; Checkpoint and rollback assurance	Low support for interoperability between workflow management system (only XML and custom workflow specification); Cloud integration; Adaptive Workflows

2.5 Conclusions

In this chapter, we investigate the current solutions for managing workflow applications in grids and clouds, offering a critical analysis on existing scheduling algorithms and management systems. We presented an overview of the best-effort workflow scheduling models.

Acknowledgments The research presented in this paper is supported by projects: *DataWay*: Real-time Data Processing Platform for Smart Cities: Making sense of Big Data—PN-II-RU-TE-2014-4-2731; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow—PN-II-PT-PCCA-2013-4-0321; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms—PN-II-PT-PCCA-2013-4-0870.

References

1. Pop, F., Zhu, X., Yang, L.T.: Midhdc: Advanced topics on middleware services for heterogeneous distributed computing. part 1. *Future Gener. Comput. Syst.* **56**, 734–735 (2016)
2. Pop, F., Potop-Butucaru, M.: Armco: Advanced topics in resource management for ubiquitous cloud computing: An adaptive approach. *Future Gener. Comput. Syst.* **54**, 79–81 (2016)
3. Simion, B., Leordeanu, C., Pop, F., Cristea, V.: A hybrid algorithm for scheduling workflow applications in grid environments (icdpd). In: OTM Confederated International Conferences “On the Move to Meaningful Internet Systems”, pp. 1331–1348. Springer (2007)
4. Vasile, M.A., Pop, F., Tutueanu, R.I., Cristea, V., Kołodziej, J.: Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Gener. Comput. Syst.* **51**, 61–71 (2015)
5. Lynch, C.: Big Data: How do your data grow? *Nature* **455**(7209), 28–29 (2008)
6. Pop, F., Iacono, M., Griboado, M., Kołodziej, J.: Advances in modelling and simulation for big-data applications (amsba). *Concurrency Comput. Practice Experience* **28**(2), 291–293 (2016)
7. Chen, M., Mao, S., Liu, Y.: Big Data: a survey. *Mob. Networks Appl.* **19**(2), 171–209 (2014)
8. Erl, T., Khattak, W., Buhler, P.: *Big Data Fundamentals: Concepts*. Prentice Hall Press, Drivers & Techniques (2016)
9. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.* **25**(5), 528–540 (2009)
10. Muresan, O., Pop, F., Gorgan, D., Cristea, V.: Satellite image processing applications in mediogrid. In: 2006 Fifth International Symposium on Parallel and Distributed Computing, pp. 253–262. IEEE (2006)
11. Gorgan, D., Bacu, V., Rodila, D., Pop, F., Petcu, D.: Experiments on esipenvironment oriented satellite data processing platform. *Earth Sci. Inf.* **3**(4), 297–308 (2010)
12. Masdari, M., ValiKardan, S., Shahi, Z., Azar, S.I.: Towards workflow scheduling in cloud computing: a comprehensive analysis. *J. Network Comput. Appl.* **66**, 64–82 (2016)
13. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: *Workflows for e-Science: Scientific Workflows for Grids*. Springer Publishing Company, Incorporated (2014)
14. Pop, F., Dobre, C., Cristea, V.: Performance analysis of grid dag scheduling algorithms using monarc simulation tool. In: 2008 International Symposium on Parallel and Distributed Computing, pp. 131–138. IEEE (2008)
15. Yu, J., Buyya, R., Ramamohanarao, K.: Workflow scheduling algorithms for grid computing. In: *Metaheuristics for Scheduling in Distributed Computing Environments*, pp. 173–214. Springer (2008)

16. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of scientific workflows in the askalon grid environment. *ACM SIGMOD Rec.* **34**(3), 56–62 (2005)
17. Maheswaran, M., Ali, S., Siegal, H., Hensgen, D., Freund, R.F.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pp. 30–44. IEEE (1999)
18. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **13**(3), 260–274 (2002)
19. Sakellariou, R., Zhao, H.: A hybrid heuristic for dag scheduling on heterogeneous systems. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004, p. 111. IEEE (2004)
20. Bajaj, R., Agrawal, D.P.: Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans. Parallel Distrib. Syst.* **15**(2), 107–118 (2004)
21. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison Wesley **1989**, 102 (1989)
22. Hou, E.S., Ansari, N., Ren, H.: A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.* **5**(2), 113–120 (1994)
23. YarKhan, A., Dongarra, J.J.: Experiments with scheduling using simulated annealing in a grid environment. In: *International Workshop on Grid Computing*, pp. 232–242. Springer (2002)
24. Menasce, D.A., Casalicchio, E.: A framework for resource allocation in grid computing. In: *MASCOTS*, pp. 259–267. Citeseer (2004)
25. Yu, J., Buyya, R., Tham, C.K.: Cost-based scheduling of scientific workflow applications on utility grids. In: *First International Conference on e-Science and Grid Computing (e-Science'05)*, pp. 8–pp. IEEE (2005)
26. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: *Integrated Research in GRID Computing*, pp. 189–202. Springer (2007)
27. Ramakrishnan, A., Singh, G., Zhao, H., Deelman, E., Sakellariou, R., Vahi, K., Blackburn, K., Meyers, D., Samidi, M.: Scheduling data-intensiveworkflows onto storage-constrained distributed resources. In: *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pp. 401–409. IEEE (2007)
28. Yu, Z., Shi, W.: A planner-guided scheduling strategy for multiple workflow applications. In: *2008 International Conference on Parallel Processing-Workshops*, pp. 1–8. IEEE (2008)
29. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berman, G.B., Good, J., et al.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Prog.* **13**(3), 219–237 (2005)
30. Xu, M., Cui, L., Wang, H., Bi, Y.: A multiple qos constrained scheduling strategy of multiple workflows for cloud computing. In: *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 629–634. IEEE (2009)
31. Durillo, J.J., Nae, V., Prodan, R.: Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Future Gener. Compu. Syst.* **36**, 221–236 (2014)
32. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., et al.: Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* **20**(17), 3045–3054 (2004)
33. Taylor, I., Shields, M., Wang, I., Rana, O.: Triana applications within grid computing and peer to peer environments. *J. Grid Comput.* **1**(2), 199–217 (2003)
34. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S.: Kepler: an extensible system for design and execution of scientific workflows. In: *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, 2004, pp. 423–424. IEEE (2004)
35. Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.L., Villazon, A., Wieczorek, M.: Askalon: A grid application development and computing environment. In: *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pp. 122–131. IEEE Computer Society (2005)
36. von Laszewski, G., Hategan, M.: Java Cog Kit Karajan/Gridant Workflow Guide. Tech. rep, Technical Report, Argonne National Laboratory, Argonne, IL, USA (2005)

Resource Management for Big Data Platforms
Algorithms, Modelling, and High-Performance
Computing Techniques

Pop, F.; Kołodziej, J.; Di Martino, B. (Eds.)

2016, XIII, 516 p. 138 illus., 57 illus. in color., Hardcover

ISBN: 978-3-319-44880-0