J Sched (2011) 14: 291–306
DOI 10.1007/s10951-009-0158-0

Modelling and solving grid resource allocation problem with network resources for workflow applications

Marek Mika · Grzegorz Waligóra · Jan Węglarz

Published online: 24 December 2009

Abstract A problem of allocating resources of a grid to workflow applications is considered. The problem consists, generally, in allocating distributed grid resources to tasks of a workflow in such a way that the resource demands of each task are satisfied. Grid resources are divided into computational resources and network resources. Computational tasks and transmission tasks of a workflow are distinguished. We present a model of the problem, and an algorithm for finding feasible resource allocations. A numerical example is included, showing the importance of the resource allocation phase on a grid. Some conclusions and directions for future research are given.

Keywords Grid · Workflow · Resource allocation · Project scheduling

1 Introduction

Grid resource management systems, being a crucial part of grid environments, have been a subject of intensive studies over the last decade (see, e.g., Nabrzyski et al. 2003). On the other hand, machine scheduling and, especially, project scheduling have been developed since the early 1950s (see Błażewicz et al. 2001, 2007; Leung 2004 for machine scheduling, and Demeulemeester and Herroelen 2002; Józefowska and Węglarz 2006; Węglarz et al. 2010 for project scheduling). However, these two research and application areas of resource allocation problems have been almost completely separated by now. This follows from the separation of the corresponding research communities and, in consequence, from a deep terminological and methodological gap between them. In this paper we attempt to fill this gap by showing under which assumptions the problem of allocating grid computational and network resources to workflow applications can be formulated and solved in the framework of project scheduling.

In order to realize this main objective of the paper, we formulate a model of the problem based on a project scheduling oriented approach, and we propose an algorithm for finding feasible resource allocations for a given workflow. The goal of the paper is also to show how important the phase of resource allocation on a grid really is, in the context of following schedules. Although the model we develop is general and allows the allocation of any set of resources to any set of tasks, we consider so-called workflow applications because of their particular practical importance. Workflow applications can be viewed as complex sets of precedence-related various transformations (tasks) performed on some data. These are mostly scientific, data-intensive applications which, because of the large amounts of computations and data involved, require high computing power to be used efficiently.

The problem of allocating grid distributed resources, existing in many sites, to tasks of a workflow application is very complex, especially when the network capacity varies between the sites. In addition, we often do not possess complete information about the tasks. The process of obtaining a performance model of a task is not trivial. In particular, the processing times of all the tasks on different computer systems (grid resources) are not easy to evaluate. Also other parameters (e.g. bandwidth, resource availability, etc.) may change quite rapidly in grid environments. Thus, generally, many problem parameters are dynamic and/or uncertain. In this work we will show under which simplifying assump-

M. Mika · G. Waligóra (✉) · J. Węglarz
Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland
e-mail: grzegorz.waligora@cs.put.poznan.pl

tions we can formulate the considered problem as a deterministic resource allocation problem. More precisely, under these assumptions we will formulate a model of the problem based on the classical project scheduling models. In the very basic project scheduling problem called RCPSP (resource-constrained project scheduling problem) activities (tasks) are to be scheduled in such a way that the precedence as well as resource constraints are satisfied, and the project duration (or the makespan), i.e. the completion time of a given set of activities, is minimized (see Demeulemeester and Herroelen 2002 for a handbook).

It is easy to see the similarity between a workflow and a project. In both cases we have a set of precedence-related tasks (activities) which are to be executed on a given set of resources. Thus, it is justified to describe the considered problem in terms of project scheduling. In Mika et al. (2003), we formulated the problem of scheduling workflows on a grid as a multi-mode resource-constrained project scheduling problem with schedule-dependent setup times (MRCPSP-SDST). Since in grid environments tasks can be processed on different types of resources, i.e. they can be executed in several different ways (modes), we considered project scheduling in its multi-mode version (MRCPSP). Moreover, we based our approach on an extension of the MRCPSP with so-called setup times, where the setup time was the time necessary to prepare the required resource for processing an activity. Setup times were transmission times of files between tasks. These times depend on which resources the tasks are scheduled on. As a result, we obtained a problem, which we named MRCPSP-SDST.

In the approach proposed in Mika et al. (2003) we only considered computational resources of a grid. We assumed that the network was not overloaded and bandwidth was not a scarce resource. As a result, tasks did not have to compete for network resources, and we treated transmission times simply as setup times easy to calculate. In this paper we modify the model towards a more realistic one. To this end, we consider the network as a resource for which tasks have to apply. Consequently, we distinguish transmission tasks as a separate type of task, which can compete for the same network connection. More precisely, bandwidth is the network resource which can be divided among many transmission tasks. Thus, in this paper we present a more practical model of the problem, where there are network resources along with computational resources on a grid.

As was stressed, e.g., in Kurowski et al. (2008), the majority of centralized grid environments are based on the so-called two-level hierarchy scheduling approach. This is caused by the grid layered architecture. A grid consists of many nodes, each of which is usually managed by some local scheduling system, such as, e.g., Condor,[1] LSF Load Shearing Facility),[2] PBS (Portable Batch System),[3] SGE (Sun Grid Engine),[4] etc. Thus, the grid broker assigns submitted jobs to remote resources in the first step, and then local schedulers generate their schedules for resources they manage. This concept is very natural since a grid broker neither possesses a full knowledge of the local resource load, nor has overall control of the resource. On the other hand, the local scheduler does not know about any other grid jobs and other resources available to these jobs. Examples of two-level hierarchy grids with a central grid broker are, among others, the European EGEE Grid,[5] or Clusterix in Poland.[6] Let us emphasize the difference between resource allocation and scheduling. Resource allocation is just assigning tasks to resources, scheduling is one step further and means allocating resources to tasks over time, i.e. defining the starting time of each task on the resource it has been assigned.

As a result of the two-level hierarchy scheduling approach, the first phase, i.e. grid resource allocation, is of crucial importance. Especially when workflow applications are concerned which require huge computational effort and can run for hours, days, or weeks. Therefore we focus in this paper on resource allocation only. We present an algorithm for finding feasible resource allocations, we analyze its complexity, and show that it never fails to find a feasible allocation if it exists. Of course, any time criterion (e.g. makespan) can be only used to evaluate schedules, not resource allocations. These can be judged, e.g., by cost criteria, if users are charged for accessing the resources. In this paper we show how resource allocation may affect the quality of the following schedule, especially in such a heterogeneous environment as a grid. However, we do not present computational experiments with the scheduling phase in this paper, since it can be performed by local schedulers according to many different scenarios, analyses of which are not the objective of this research.

The paper is organized as follows. In Sect. 2 we describe the problem of allocating grid resources to workflow applications in more detail. Section 3 presents the model of the problem, with the description of all its parameters and comments on the assumptions made in the model. Section 4 is devoted to a numerical example, showing the importance of resource allocation on a grid. In Sect. 5 we discuss an approach to the problem stated, and we present an algorithm for finding feasible grid resource allocations. The last section contains conclusions and some directions for future research.

[1] http://www.cs.wisc.edu/condor/.

[2] http://www.platform.com/.

[3] http://www.pbsgridworks.com/.

[4] http://gridengine.sunsource.net/.

[5] http://www.eu-egee.org/.

[6] http://clusterix.pcz.pl/.

2 Problem description

Let us start with a brief description of workflow applications. In many scientific areas, like high-energy physics, bioinformatics, astronomy and others, we encounter applications involving numerous simpler components that process large data sets, execute scientific simulations, and share both data and computing resources. Such data-intensive applications consist of multiple components (tasks) which may communicate and interact with each other over the course of the application. The tasks are very often precedence-related, and the precedence constraints usually follow from data flow between them, i.e. data files generated by one task are needed to start another task (an output of one task becomes an input for the next one). Although this is the most common situation, the precedence constraints may follow from other reasons as well, e.g., may be arbitrarily defined by the user. Such complex applications, consisting of various precedence-related transformations (tasks) performed on some data, between which data files have to be transmitted very often, are called workflow applications. For example, in astronomy workflows with thousands of tasks need to be executed during the identification of galaxy clusters within the Sloan Digital Sky Survey (Szalay et al. 2000). In Deelman et al. (2005) two types of workflows were distinguished: data-intensive, whose file transfer times dominate task computing times, and compute-intensive, for which it is just the opposite. For both types, because of the large amount of computations and data involved, high computing power is required to execute the workflows efficiently. This power can be delivered by a grid.

In Foster and Kesselman (1999) a grid was defined as an infrastructure for coordinated resource sharing and problem solving in dynamic, multi-instrumental virtual organizations. More recently, the Network of Excellence CoreGRID[7] formulated a definition of a grid as "a fully distributed, dynamically reconfigurable, scalable and autonomous infrastructure to provide location independent, pervasive, reliable, secure and efficient access to a coordinated set of services encapsulating and virtualizing resources (computing power, storage, instruments, data, etc.) in order to generate knowledge".

Since workflow applications are usually very time consuming (even if single tasks can be quite short), and input/output data files for tasks can be quite large, the problem of allocating resources to such applications on a grid has become a great challenge and has great practical importance these days. An allocation of grid resources to a workflow whose component tasks are known but not yet scheduled is an important topic in grid computing, because of its high impact on the efficiency of the workflows, which can generate huge amounts of data and occupy resources for days or weeks.

In general, the considered problem consists in allocating distributed grid resources to heterogeneous tasks. Users submit their jobs (in this case—workflow applications) to a grid. As has been mentioned, workflows consist of multiple tasks. Generically, a task can be anything that needs a resource—a schedulable computation, a bandwidth request, data access or an access to any remote resource, such as remote instruments, databases, humans-in-the-loop, etc. A resource is anything that can be allocated to tasks—a processor, disc space, bandwidth, machine, device, person, etc. In this research, we divide grid resources into two types: computational resources and network resources. Computational resources are all resources needed for computational tasks to be computed; they are not just processors, but also memory, disc space, various devices, etc. Network resources are resources needed for transmission tasks to be executed, i.e. needed for data files to be transmitted. The basic network resource is, obviously, bandwidth.

There are at least several approaches to grid resource allocation. They differ one from another depending on the grid architecture, objectives of a particular grid, and policies for managing the grids. With respect to architecture, two types of grids can be distinguished: peer-to-peer grids, where all services are equal and communicate using a peer-to-peer model, and centralized grids, where a grid resource management system plays a central role and is surrounded by many other grid services structured in a layered architecture. In such grids there is usually one common, central grid broker which serves all users and their jobs. Such a situation is considered in this paper.

A good allocation of distributed grid resources to users' jobs is crucial for an efficient execution. However, in many existing grid toolkits, matchmaking strategies are used which do not care about an overall efficiency of the execution of a set of tasks. They just concentrate on matchmaking individual tasks to resources and do not attempt to find an efficient overall allocation. Because these strategies do not take into account tasks of the workflow that arrive later on, their allocation of resources may result in a poor overall schedule, especially for data-intensive applications when excessive data movement is created. In Deelman et al. (2005), two types of approaches (and algorithms) to grid resource allocation were distinguished. The first approach greedily allocates resources to each ready-to-run task, taking into account only task information. This approach (and algorithms) is called a task-based approach (task-based algorithms). Such algorithms make local decisions about which task to send to which resource. The second approach searches for an efficient allocation for the whole workflow, and may revise the allocation of a task based on subsequent tasks. In

[7] http://www.coregrid.net/.

this approach, all the tasks of a workflow are mapped a priori to resources, in order to minimize the makespan of the whole workflow. However, mapping the entire workflow does not imply scheduling all the tasks ahead of time. If some changes in the environment occur, remapping may be necessary. In other words, in real deployments one would like to find an overall allocation, but one would only release portions of the workflow that are ready to run. If the underlying execution environment subsequently changes, the allocation may be redone (Deelman et al. 2005). Our research corresponds to the latter approach.

Finally, in order to build a formal model of the problem considered, we have to precisely define the assumptions, as well as simplifications that are necessary to model such a complex problem. Too few attributes (i.e. too general an approach) may imply that the model becomes unrealistic, but on the other hand, too many of them can result in too complex a model, which could hardly be solvable by any existing or newly developed method. Therefore, it is crucial to identify these parameters, which allow us to build both a realistic and solvable model of the problem. The next section presents all the assumptions made and the model itself.

3 The model

In this section we formulate the model of the problem considered. The first subsection describes the assumptions concerning the grid environment, the second subsection the assumptions concerning the workflow applications. In the last subsection we summarize all the parameters of the defined problem.

We stress that although we focus on resource allocation in this paper, the model presented is more general and concerns, in fact, the problem of scheduling workflows on a grid. Therefore, some of its parameters, in particular time parameters like processing times of tasks, will not be used in the approach discussed later on in the paper. These parameters will be needed in further research, when we reach the phase of scheduling. As is easy to see, the defined problem is NP-hard in the strong sense, as it reduces to the MRCPSP in the case when all tasks are executed in the same node. The MRCPSP, as well as the RCPSP, are known to be strongly NP-hard problems (see Demeulemeester and Herroelen 2002).

3.1 Grid

We make the following assumptions about the grid environment:

1. A grid is a set of nodes connected by fast network links.
2. There are two types of nodes in the network: resource nodes and non-resource nodes. Resource nodes contain resources for which users' computational tasks compete. These are, in fact, resources of any type for which the tasks can apply (e.g. processors, memory, disc space, devices, etc.), but at this stage we consider the basic computational resources only, i.e. processors. Non-resource nodes are considered only with respect to the network topology. There may exist various resources in such nodes but they are either not available for some reasons, or not constrained (are not scarce resources). In other words, non-resource nodes contain only such resources for which computational tasks do not have to compete.
3. Bandwidth between each two connected nodes is given, and it is identical in both directions.
4. Between two given nodes there can be more than one network link, and these links may have different parameters. We assume that these are alternative links which cannot be merged in order to increase the bandwidth.
5. Bandwidth of each link is discretely divisible, i.e. a minimal portion (quant) of bandwidth is assumed.
6. Bandwidth within a given node is unlimited.
7. Network devices and interfaces operating in the nodes do not cause any bandwidth limitations, i.e. they are able to deal with every data transfer, incoming or outgoing. In other words, the network bandwidth depends only on the bandwidths of the links, and not on the characteristics of the network interfaces.
8. Breakdowns of the physical links are so rare that can be neglected.
9. Communication channels are highly reliable, i.e. network links guarantee that, during transmission, packets are not being lost or duplicated.
10. Delays do not affect the transmission times between nodes, i.e. transmission lateness is so small in comparison to the transmission time itself that it can be neglected.
11. Transmission between nodes does not require computational resources, i.e. each processor used for the execution of a computational task is free immediately after the completion of this task and can be used for the execution of the next computational task.
12. Processors are divided into types, depending on their power. The power (processing speed) of each processor is given by a function of some standard unit. The form of this function is identical for all processors of a given type. At this stage, we assume a linear form of the processing speed function, as "a speed factor multiplied by the standard unit". A processor with a speed factor equal to 1 we call a standard processor.

3.2 Workflows

Below we present the assumptions about the workflows:

1. At this stage only workflow jobs are considered in the model, because of their high practical importance. This is not a limitation of the model, and it is easy to extend it for jobs of other types in the future. For simplicity, we also assume now that there is one workflow job to be scheduled.
2. A workflow consists of many tasks. There are two types of tasks: computational tasks and transmission tasks.
3. The structure of a workflow is represented by a directed acyclic graph (DAG), where each vertex corresponds to a computational task, and each arc represents a precedence relation between two computational tasks.
4. We assume that each arc corresponds to a transmission task, i.e. represents a data transmission between two successive computational tasks. In other words, we assume that precedence constraints represented by arcs of the DAG follow only from data transmissions between computational tasks.
5. Computational tasks are non-preemptable, i.e. once started they have to be completed with no interruptions and no changes in resource allocation. We also assume that each computational task may be executed in exactly one node.
6. Each computational task is characterized by two values: its size, i.e. the execution time on a standard processor (processors), and the number of processors required for its execution. The actual processing time of a task is calculated by dividing its size by the speed factor of the processor (processors) on which this task is scheduled.
7. We assume that computational tasks are not scalable, i.e. the number of processors required for the execution of such a task is given a priori by the user and cannot be changed. Moreover, we also assume that each computational task is executed by the specified number of processors of the same type (it is not possible to assign a task to processors of different types).
8. Transmission tasks are also non-preemptable, i.e. they also have to be executed with once allocated resources and with no interruptions. In this case it means that the data transmission has to be entirely performed using the same path in the grid connection graph, i.e. once a connection is established between two nodes, the whole transmission must be performed over this connection (a sequence of links).
9. Transmission tasks are characterized by two values: the size of the data file (files) to be transmitted, and the required bandwidth between the two nodes between which the transmission is to be performed. The user, submitting a workflow to the system, specifies the minimal bandwidth which is required to transmit data between each two precedence-related computational tasks. At this stage, we assume that a transmission task gets for its execution a connection with bandwidth equal to the minimal value given by the user. As a result, the transmission time (i.e. the execution time of a transmission task) can take one of two values: the data file size divided by the bandwidth, when successive computational tasks are executed in different resource nodes, or zero, when they are executed in the very same resource node.
10. Transmission tasks are not scalable, either. In this case it means that the entire transmission is performed using the assigned bandwidth, which may not be changed during the execution of the transmission task.

Let us comment on points 9 and 10 of Sect. 3.2 in more detail, as they describe important assumptions on network transmission. As is known from network theory, different applications and services have different characteristics and requirements with regard to the underlying network. There are applications like Telephony or Voice Over IP which are very sensitive to delay. On the other hand, there are applications (e.g. compressed video streaming) which are quite sensitive to packet loss. Based on the specification of Quality of Service categories for ATM networks, we may consider the three major QoS classes of service in today's networks:

- Constant Bit Rate (CBR)
- Variable Bit Rate (VBR)
- Available Bit Rate (ABR)

The CBR class of service is characterized by a constant transmission speed of cells. CBR is used when the application needs a static amount of bandwidth continuously available for the duration of the active connection. It is designed for real-time applications, in particular applications involving the streaming of voice and video, for which overall network delay is often critical.

The VBR service class, similarly to CBR, supports real-time applications, such as Voice over IP or video conferencing. Such applications require tightly constrained delays and delay variations. In contrast to CBR, VBR makes a better use of bandwidth for the applications transmitting bursty traffic (when the transmission speed varies during the transmission time).

The ABR class of service is designed for the transmission of files or other non-real-time data for which a minimum amount of bandwidth is specified. When congestion occurs in the network, there is a little available bandwidth for the application. However, when the network is not congested, the whole bandwidth is available for use. ABR uses mechanisms allowing us to make use of any bandwidth available along the end-to-end transmission path at any point of time.

In our model we, in fact, assume the ABR class of service, as the user specifies a minimum bandwidth needed for

his transmission task. However, as has been stated in point 9 of Sect. 3.2, in this formulation of the model we take this minimal value as the one that the task actually gets for its execution, which makes it closer to the CBR class. Moreover, we do not consider the VBR class at all, as is said in point 10. On the other hand, as long as we restrict ourselves to resource allocation, the assumption on the considered class of service does not have a significant influence. It will be important in the phase of scheduling.

In the next extensions of the model, plans are made to take into account all the three classes of service, which results in the possibility of scalable transmission tasks. Moreover, other extensions may concern scalable computational tasks (see point 7), or tasks simultaneously executed in many nodes like, e.g., MPI tasks (see point 5).

3.3 Problem parameters

Below we summarize all the parameters of the model:

(A) Grid

$\Gamma(N, \Psi)$—undirected graph representing the network topology of a grid;
N—set of all (resource and non-resource) nodes in the network, $N = X \cup \Pi$;
X—set of resource nodes;
Π—set of non-resource nodes;
Ψ—set of edges (links) between nodes, i.e. set of couples $(\mu, \nu)_\psi : \mu, \nu \in N, \psi = 1, 2, \ldots$ (ψ denotes alternative links between a given pair of nodes);
X_χ—resource node χ, $\chi = 1, 2, \ldots, |X|$;
Π_η—non-resource node η, $\eta = 1, 2, \ldots, |\Pi|$;
ϖ_κ—speed factor for processors of type κ, $\kappa = 1, 2, \ldots$;
$P_{\kappa\chi}$—the number of processors of type κ, $\kappa = 1, 2, \ldots$ (i.e. processors with the same speed factor ϖ_κ) in resource node X_χ;
$\Psi_\psi^{\mu\nu}$—bandwidth of the link $(\mu, \nu)_\psi \in \Psi$, $\psi = 1, 2, \ldots$;

(B) Workflow

$W(V, E)$—directed acyclic graph (DAG) representing the structure of a workflow;
V—set of vertices of graph W representing computational tasks;
E—set of arcs (ν_i, ν_j) of graph W representing precedence constraints between computational tasks $\nu_i, \nu_j \in V$, i.e. transmission tasks;

Computational tasks

ν_i—computational task i, $i = 1, 2, \ldots$;
p_i—size of computational task ν_i (expressed in assumed computational units, e.g. MIPS or similar), $i = 1, 2, \ldots, |V|$;
r_i—number of processors required for the execution of computational task ν_i, $i = 1, 2, \ldots, |V|$;
ω_i—minimal speed factor of the processor (processors) required for the execution of computational task ν_i, $i = 1, 2, \ldots, |V|$;
$f_i(p_i, \varpi_\kappa)$—function defining the actual execution time of computational task ν_i, $(i = 1, 2, \ldots, |V|)$ on processor (processors) with speed factor ϖ_κ $(\varpi_\kappa \geq \omega_i)$; at this stage we assume that $f_i = f = p_i/\varpi_\kappa$ $(i = 1, 2, \ldots, |V|,\ \kappa = 1, 2, \ldots)$;

Transmission tasks

(ν_i, ν_j)—task of transmission of output data of computational task ν_i, which is at the same time input data for computational task ν_i $(\nu_i, \nu_j) \in E$;
F^{ij}—size of data file (files) transmitted as a result of the execution of transmission task (ν_i, ν_j);
B^{ij}—minimal required bandwidth of the connection between resource nodes in which computational tasks ν_i and ν_j will be executed;
$g(F^{ij}, B^{ij})$—execution time of transmission task (ν_i, ν_j), i.e. the time of data transmission from the resource node in which computational task ν_i will be executed to the resource node in which computational task ν_j will be performed. As mentioned before, at this stage we assume that $g(F^{ij}, B^{ij}) = F^{ij}/B^{ij}$ if tasks ν_i and ν_j are executed in different nodes, or 0 if in the same node.

4 Example

Let us first stress that the model presented in Sect. 3 is a deterministic one. It can be an approximation of a real grid resource allocation problem under the assumptions made and described in the previous section. The model also allows us to build a schedule of a workflow application on a grid, where a schedule is understood not just as a resource allocation, but as an allocation of resources to tasks over time. In other words, the model assumes that resources allocated to tasks according to the constructed allocation will be available for these tasks in time windows sufficient to execute the tasks. Of course, in general, resource re-allocation is still possible during the execution of the schedule created.

In this section we will show the importance of resource allocation in the context of the quality of the schedule built on its basis. As a scheduling criterion, we will assume the makespan, i.e. the total time of execution of a given workflow. However, let us stress here that various performance measures may be considered as the scheduling criterion (e.g. cost, reliability, resource levelling, etc.). Furthermore, a multi-objective approach is well justified which can combine two or even more measures. For multi-criteria approaches to problems of scheduling on a grid, see Kurowski et al. (2001, 2003, 2006, 2008).

We will consider two allocation scenarios. The first scenario assumes that it is not reasonable to send tasks to other

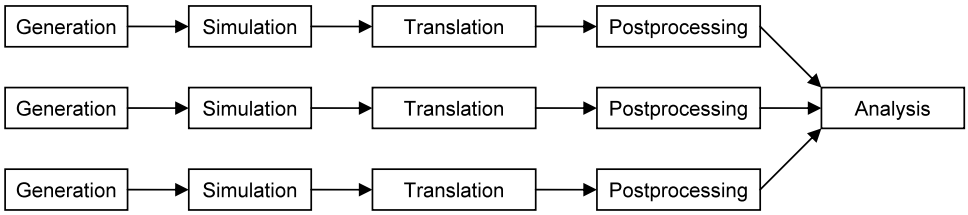

Fig. 1 Generic simulation workflow structure

sites, and the best way to execute a workflow is to do it on local resources (on a local node). Obviously, it is assumed that the set of local resources is capable of executing the given workflow, with respect to computational resources only. We assume (point 6 of Sect. 3.1) that the bandwidth within a given node is unlimited; thus, it is not a scarce resource in this case.

The second scenario, however, shows that assigning tasks of a workflow to distributed grid resources can be profitable, even if there are several transmissions to perform. We will show how a non-local resource allocation can improve the schedule, i.e. give a better makespan. Of course, from among many distributed resource allocations, there will be such of worse and better quality. For the purpose of this example, we only show that assigning tasks to resources over many sites can lead to an improvement of a local allocation.

Let us consider the following workflow.

One of the most common uses of workflows is large-scale scientific simulation. A typical simulation schema may consist of the following steps: generation of data for simulation according to given input parameters (preprocessing), simulation, translation of output data into required format, postprocessing, analysis and comparison of the obtained results. The general structure of such a workflow is presented in Fig. 1.

A specific kind of a workflow that fits to this general schema is a simulation of the Compact Muon Solenoid (CMS). CMS is a multi-purpose particle physics detector constructed at the European Center for Nuclear Research (CERN) in Geneva, Switzerland (Wulz 1998). As stated in Deelman et al. (2003), the CMS detector is expected to record data, produced by high-energy proton–proton collisions occurring within CERN's Large Hadron Collider (LHC), at a rate of 100 MB/s. After the data have been recorded, they will be passed through various filter stages, which transform and reduce the data into formats that are more easily analyzed by physicists. In order to better understand the response of the detector to different input signals, large-scale Monte Carlo simulations are performed which typically involve several different computational stages. These simulations are long-running, parallel, multi-stage processes that are ideally suited for grid computation. Typically, a single workflow creates approximately 1 GB of data and requires 10 to 20 CPU/hours depending on the type of simulation. A typical production run may include thousands of workflows.

In Deelman et al. (2003) the use of workflows for this problem was described. Workflows were generated and executed using the Chimera (Foster et al. 2002) and Pegasus (Deelman et al. 2004) tools. Among others, one particular paper (Deelman et al. 2003) presents a simple simulation use-case known as an "n-tuple-only production", which consists of a five stage computational pipeline that fits to the structure presented in Fig. 1. It consists of the following tasks:

1. In the generation stage, it simulates the underlying physics of each CMS event.
2. In the simulation stage, the CMS detector's response to the events created in the generation stage is modeled.
3. In the translation stage (called formatting stage), it copies the simulated detector data into an object-oriented format.
4. In the postprocessing stage (called reconstruction stage), the obtained data are transformed, producing a "picture" of what a physicist would "see", as if the simulated data were actual data recorded by the experimental apparatus.
5. In the final analysis stage, user-specific information is selected and a convenient, easy-use file that can be analyzed by a researching physicist is created. The results obtained for different input parameters are also aggregated and compared.

In work described in Deelman et al. (2003) these workflows were executed on a computing cluster consisting of 25 dual-processor Pentium (1 GHz) machines. Over the course of 7 days, 678 jobs of 250 events each were executed. From among these jobs, 167 500 events were successfully produced using approximately 350 CPU/days of computing power and generating approximately 200 GB of simulated data.

For the purpose of the numerical example, we assume the following structure of the workflow instance, presented in Fig. 2.

The grid environment considered in this example consists of 3 resource nodes (20 CPUs) and 3 non-resource nodes. Its structure is presented in Fig. 3.

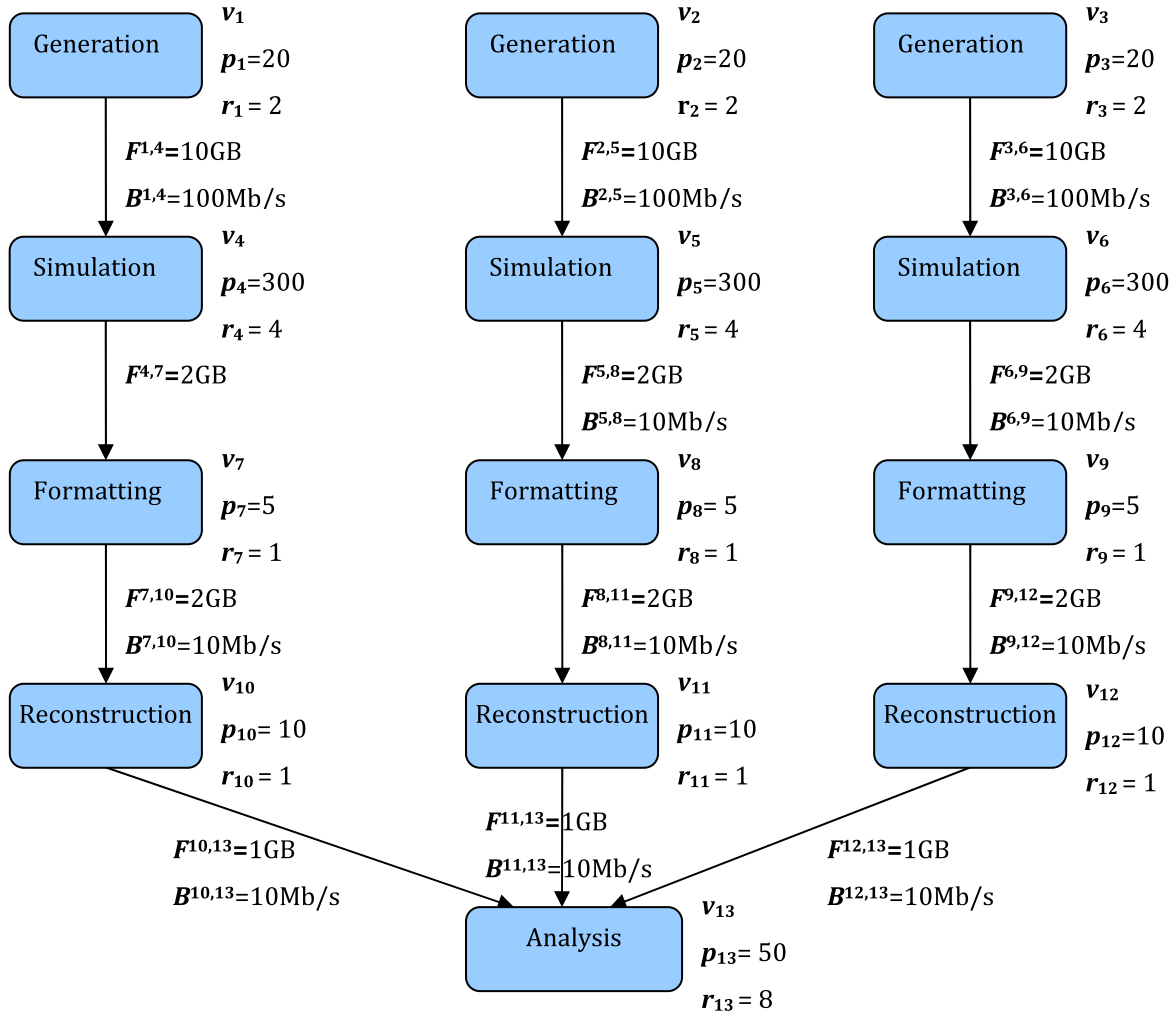

Fig. 2 CMS workflow instance

The processors available in the resource nodes are

Node X_χ	Processor speed ϖ_κ	Number of processors $P_{\kappa\chi}$
X_1	1	8
X_2	2	4
	3	4
X_3	3	2
	4	2

The bandwidths of the links between nodes are

Link (μ, ν)	Bandwidth $\Psi^{\mu,\nu}$
(1, 4)	1 Gb/s
(2, 5)	1 Gb/s
(3, 6)	1 Gb/s
(4, 5)	1 Gb/s
(4, 6)	1 Gb/s
(5, 6)	100 Mb/s

Now, as has been mentioned earlier, the first scenario allocates all computational tasks to one resource node and executes them locally with no transmission between nodes. Since task v_{13} requires 8 processors, the only node capable of executing the entire workflow is X_1 (it is not allowed to assign a task to processors of different types—point 7 of Sect. 3.2). Figure 4 presents the schedule obtained by assigning all tasks of the workflow to node X_1, and scheduling according to a simple rule that each task is started as soon as all its predecessors are finished and processors required by this task are available. In the figure, the vertical axis represents the number of processors, the horizontal axis represents time. The actual processing times of tasks have been calculated as p_i/ϖ_κ. We assume for simplicity that $\omega_i = 1$, $i = 1, 2, \ldots, 13$, i.e. each task can be executed on each type of the processors. The processing times of all transmission tasks are equal to 0, as we assume an unlimited bandwidth within each node.

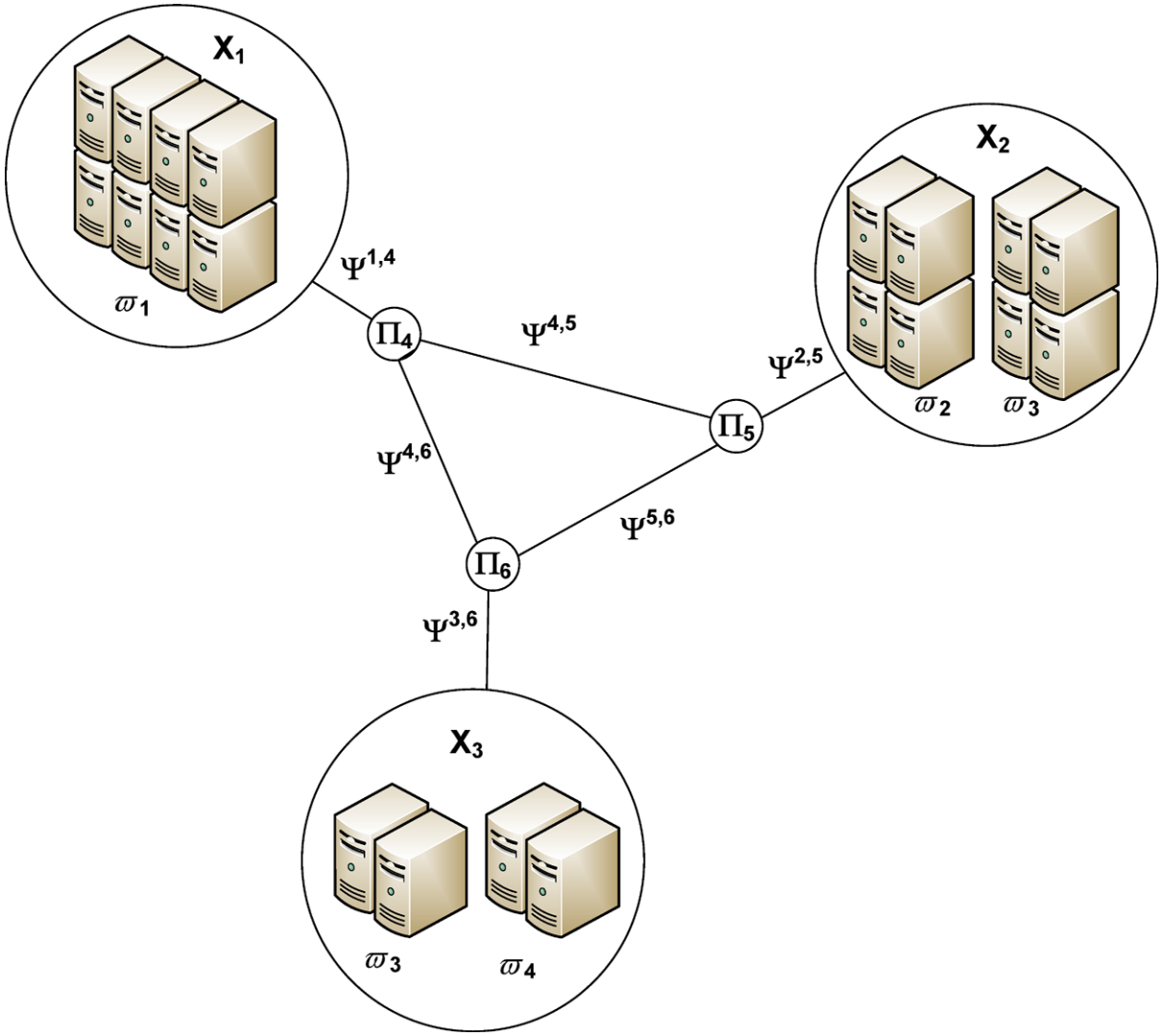

Fig. 3 Grid environment

In the second scenario, generation tasks $v_1 \div v_3$ are executed in node X_1, and then transmission tasks $(v_1, v_4) \div (v_3, v_6)$ send generated files from node X_1 to X_2 over connection $X_1 \to \Pi_4 \to \Pi_3 \to X_2$. In node X_2 simulation, formatting, and reconstruction tasks $(v_4 \div v_{12})$ are executed. Since the analysis task v_{13} requires 8 processors of the same type, it cannot be executed in node X_2 but only in node X_1. Therefore, next transmission tasks $(v_{10}, v_{13}) \div (v_{12}, v_{13})$ are needed to send files required for task v_{13} back to node X_1. The processing times of transmission tasks between different nodes have been calculated as F^{ij}/B^{ij}. The obtained schedule is shown in Fig. 5.

The presented example shows that the distributed schedule achieves a better makespan than the local one. The time benefit from executing computational tasks on faster processors is larger than the time waste following from transmitting files between nodes. This confirms the importance of efficient resource allocation in a grid environment.

5 Resource allocation

We have shown in Sect. 4 that optimization of grid resource allocation might be a very important problem from the practical point of view. Especially this is so when workflow applications are concerned, which are data-intensive applications requiring huge computational power to be used efficiently. In such applications, the time benefit following from scheduling them effectively on grid resources may be quite significant. In this section we discuss the problem of finding a feasible allocation of grid computational and network resources to tasks of a workflow.

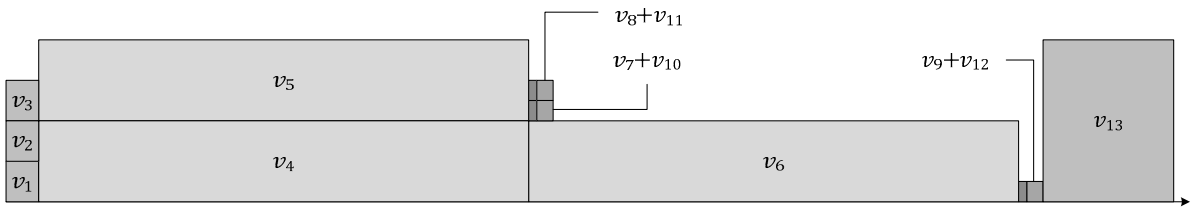

Fig. 4 Local schedule in node X_1

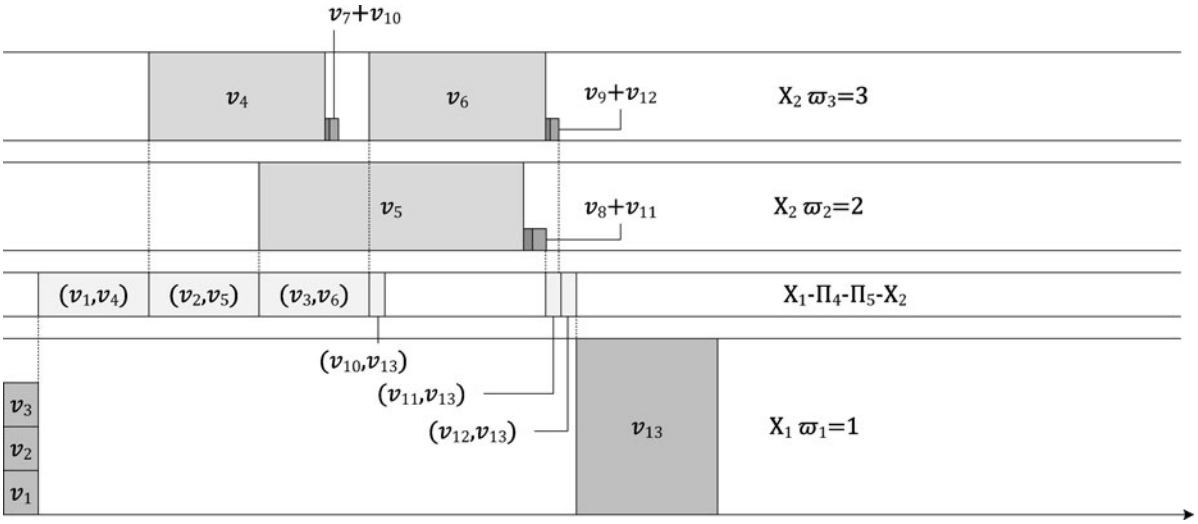

Fig. 5 Distributed schedule

5.1 Definitions and notation

Let us first define the feasibility conditions of a resource allocation in the presented problem. We stress that at this stage we understand resource allocation as assigning computational tasks to resource nodes, and assigning transmission tasks to connections of required bandwidths. We do not assign computational tasks to particular processors in resource nodes, i.e. we do not perform the scheduling phase at this stage. Let us start with definitions and notation.

Definition 1 Resource node $X_\chi \in X$ is *capable* for a computational task $v_i \in V$ if $P_{\kappa\chi} \geq r_i$ and $\varpi_\kappa \geq \omega_i$.

Definition 1 calls a resource node capable for a task if the node has enough computational power to execute this task. Let us recall that r_i is the number of processors required for the execution of computational task v_i, and ω_i is the minimal speed factor of the processors executing task v_i. The condition in Definition 1 means that the number $P_{\kappa\chi}$ of processors with speed factor $\varpi_\kappa \geq \omega_i$ available in resource node X_χ must be greater than or equal to r_i. If a task is assigned to a node possessing the required number of processors of the required type, the task is going to be executed, and at most it will have to wait until the processors are free.

Definition 2 A *Bij-link* is a link $(\mu, \nu)_\psi \in \Psi$ such that its bandwidth $\Psi_\psi^{\mu\nu}$ is at least equal to B^{ij}, i.e. $\Psi_\psi^{\mu\nu} \geq B^{ij}$. A link which is not a Bij-link, i.e. $\Psi_\psi^{\mu\nu} < B^{ij}$, is called a *non-Bij-link*.

Let us recall that B^{ij} is the minimal required bandwidth of the connection between resource nodes in which computational tasks v_i and v_j will be executed. Definition 2 calls a link a Bij-link if it has at least the required bandwidth for transmission task $(v_i, v_j) \in E$.

Assume now that the computational task v_i is assigned to resource node $X_\chi \in X$, and computational task v_j is assigned to resource node $X_\theta \in X$. Let $P^{ij}(\chi, \theta)$ denote a path in graph $\Gamma(N, \Psi)$ from node X_χ to node X_θ.

Definition 3 A *Bij-path* is a path $P^{ij}(\chi, \theta)$ which consists of Bij-links only.

Definition 3 calls a path a Bij-path if it is capable of performing a given transmission task (v_i, v_j).

Now, we define a feasible resource allocation RA_W for workflow W.

Definition 4 Resource allocation RA_W for workflow W is *feasible* if:

(i) each computational task $\nu_i \in V$ is assigned to a capable resource node $X_\chi \in X$, i.e.:

$$\underset{\nu_i \in V}{\forall} \nu_i \otimes X_\chi \quad \Longleftrightarrow \quad \underset{\kappa}{\exists}(P_{\kappa\chi} \geq r_i \wedge \varpi_\kappa \geq \omega_i) \tag{1}$$

(ii) each transmission task $(\nu_i, \nu_j) \in E$ can be performed over a Bij-path, i.e.:

$$\underset{(\nu_i,\nu_j)\in E}{\forall} (\nu_i \otimes X_\chi \wedge \nu_j \otimes X_\theta)$$

$$\Longrightarrow \quad \underset{P^{ij}(\chi,\theta)}{\exists} \; \underset{(\mu,\nu)_\psi \in P^{ij}(\chi,\theta)}{\forall} \Psi_\psi^{\mu\nu} \geq B^{ij} \tag{2}$$

where $\nu_i \otimes X_\chi$ denotes that computational task ν_i is assigned to resource node X_χ.

These two conditions are, of course, interrelated. Let us first discuss them separately.

To satisfy condition (i) is trivial. For each computational task, a list of capable nodes can be constructed by comparing the computational requirements of this task with the computational capabilities of each resource node. Let $Y^i \subseteq X$ denote the set of resource nodes capable of executing computational task ν_i. Each task ν_i then has to be assigned to exactly one node from set Y^i. If for any task ν_i, $Y^i = \emptyset$, then there is no feasible resource allocation for the considered workflow.

Satisfying condition (ii) is more complex. Firstly, graph $\Gamma^{ij} \subseteq \Gamma$ is constructed out of graph Γ by removing from graph Γ all non-Bij-links (i.e. links with bandwidths less than B^{ij}). Then, all connected components in Γ^{ij} are identified, as the whole Γ^{ij} may not be a connected graph anymore. A connected component is, from the definition of graph theory, a maximal connected subgraph. Finding connected components in an undirected graph can be easily done by the Depth-First Search (DFS) method—a very-well known method from graph theory. Let $\Gamma_k^{ij} \subseteq \Gamma^{ij}$, $k = 1, 2, \ldots$, denote the k-th connected component in Γ^{ij}. Such a component we will call briefly a *subgrid*. As each Γ_k^{ij} contains Bij-links only, a transmission task (ν_i, ν_j) can be assigned to any of those subgrids. If we denote by N_k^{ij} the set of nodes in subgrid Γ_k^{ij}, then $X_k^{ij} = N_k^{ij} \cap X$ is the set of resource nodes in Γ_k^{ij}. In order to assure the possibility of execution of the transmission task (ν_i, ν_j), computational tasks ν_i and ν_j have both to be assigned to their capable nodes from the same set X_k^{ij} (possibly even to the same node). Notice that in an extreme case all links from graph Γ may be removed, if they all are non-Bij-links, and each subgrid Γ_k^{ij} consists of one node only (i.e. $k = 1, \ldots, |N|$). However, it is still possible to execute a transmission task (ν_i, ν_j) by assigning tasks ν_i and ν_j to the same node, if it is capable of executing them both. Recall that we assume an unlimited bandwidth within a given node; as a result, each transmission task can be executed in that way.

Definition 5 A *tri-task* $\langle \nu_i, \nu_j \rangle$ is a triple $\{\nu_i, (\nu_i, \nu_j), \nu_j\}$, i.e. two computational tasks and a transmission task between them.

Based on the above considerations, it is now possible to show how to ensure a feasible resource allocation for a tri-task $\langle \nu_i, \nu_j \rangle$.

Definition 6 A *feasible resource allocation* RA^{ij} *for tri-task* $\langle \nu_i, \nu_j \rangle$ is a pair (X_χ, X_θ) such that $\nu_i \in Y^i$ and $\nu_j \in Y^j$, and there exists at least one Bij-path $P^{ij}(\chi, \theta)$ between nodes X_χ and X_θ.

For subgrid Γ_k^{ij}, a feasible resource allocation RA^{ij} for tri-task $\langle \nu_i, \nu_j \rangle$ can be obtained by:

(a) assigning ν_i to any node from the set $Z_k^i = Y^i \cap X_k^{ij}$, and
(b) assigning ν_j to any node from the set $Z_k^j = Y^j \cap X_k^{ij}$.

Of course, if at least one of the sets Z_k^i, Z_k^j is empty (i.e. $Z_k^i = \emptyset$ or $Z_k^j = \emptyset$), the given tri-task cannot be performed in the k-th subgrid. In an extreme situation, if there is no subgrid capable of executing the particular tri-task, it means that there is no feasible resource allocation for the given workflow and, as a result, it cannot be executed on the grid it has been submitted to. If there is at least one tri-task in the workflow for which no feasible resource allocation exists, the whole workflow cannot be executed. In all other cases, we are able to define all possible resource allocations for a given tri-task. This will be shown in Sect. 5.2.

Let us now summarize the notation introduced in this section:

$P^{ij}(\chi, \theta)$—a path in graph $\Gamma(N, \Psi)$ from node X_χ to node X_θ;
$Y^i \subseteq X$—the set of capable nodes for task ν_i;
$\Gamma^{ij} \subseteq \Gamma$—a subgraph of Γ obtained by removing from Γ all non-Bij-links;
$\Gamma_k^{ij} \subseteq \Gamma^{ij}$—the k-th connected component (subgrid) in Γ^{ij};
N_k^{ij}—the set of all nodes in subgrid Γ_k^{ij};
$X_k^{ij} = N_k^{ij} \cap X$—the set of resource nodes in subgrid Γ_k^{ij};
$Z_k^i = Y^i \cap X_k^{ij}$—the set of capable nodes for task ν_i in subgrid Γ_k^{ij};
$\langle \nu_i, \nu_j \rangle$—a tri-task denoting computational tasks ν_i and ν_j, and transmission task (ν_i, ν_j);
$RA^{ij} = (X_\chi, X_\theta)$—a feasible resource allocation for tri-task $\langle \nu_i, \nu_j \rangle$;
RA_W—feasible resource allocation for workflow W.

5.2 Algorithm RA-TT

In this section we will show how to find all feasible resource allocations for a given tri-task with respect to Definition 6.

This simply can be done by defining sets Z_k^i, Z_k^j for each subgrid Γ_k^{ij}, $k = 1, 2, \ldots$, and a feasible resource allocation RA^{ij} for $\langle v_i, v_j \rangle$ is any combination (X_χ, X_θ) such that $X_\chi \in Z_k^i$ and $X_\theta \in Z_k^j$. In other words, the sets Z_k^i, Z_k^j define the resource nodes of subgrid Γ_k^{ji} to which computational tasks v_i, v_j (respectively) of tri-task $\langle v_i, v_j \rangle$ may be assigned. These sets will be different for different subgrids.

Now, enumerating all such combinations (X_χ, X_θ) within a given subgrid gives possible resource allocations in this particular subgrid. Collecting all those combinations over the whole workflow (all subgrids) gives the set of feasible resource allocations for tri-task $\langle v_i, v_j \rangle$.

Let A^{ij} be the set of all feasible resource allocations RA^{ij} for tri-task $\langle v_i, v_j \rangle$ of a workflow W. The following algorithm finds the set A^{ij}:

Algorithm RA-TT

1. $A^{ij} = \emptyset$.
2. Find sets Y^i and Y^j. If $Y^i = \emptyset$ or $Y^j = \emptyset$, then no feasible RA^{ij} exists and STOP.
3. Construct graph Γ^{ij}.
4. Use the DFS method to find all connected components (subgrids) Γ_k^{ij}, $k = 1, 2, \ldots$, of graph Γ^{ij}.
5. For each $k = 1, 2, \ldots$, define sets $Z_k^i = Y^i \cap X_k^{ij}$ and $Z_k^j = Y^j \cap X_k^{ij}$, where $X_k^{ij} = N_k^{ij} \cap X$. If $Z_k^i = \emptyset$ or $Z_k^j = \emptyset$, then tri-task $\langle v_i, v_j \rangle$ cannot be executed in subgrid Γ_k^{ij}.
6. For each $k = 1, 2, \ldots$, substitute A^{ij} by $A^{ij} = A^{ij} \cup (Z_k^i \times Z_k^j)$. If $A^{ij} = \emptyset$, then no feasible RA^{ij} exists.

Algorithm RA-TT shows two cases when the considered tri-task, and in consequence the entire workflow, cannot be executed on the considered grid:

- if there is at least one computational task for which no capable resource node exists (point 2 of Algorithm RA-TT), or
- if each computational task has at least one capable node, but there is at least one transmission task which cannot be executed because no connection with the required bandwidth between given nodes exists (point 6 of Algorithm RA-TT).

The first situation is trivial. The second situation may be illustrated by a simple example, when task v_i can only be executed in node X_χ, task v_j can only be executed in X_θ, and there is no Bij-path between X_χ and X_θ.

Let us now analyze the complexity of Algorithm RA-TT.

The complexity of step 2 is $\mathcal{O}(|X|)$, since all computational nodes must be checked, whereas the complexity of step 3 is $\mathcal{O}(|\Psi|)$, since all links of the grid have to be examined. The DFS in step 4 has a complexity of $\mathcal{O}(|N| + |\Psi|)$. The intersections in step 5 give it the complexity of $\mathcal{O}(|X|^2)$. Finally, the Cartesian products for each subgrid make step 6 have a complexity of $\mathcal{O}(|X|^3)$. As a result, the complexity of the RA-TT algorithm is $\mathcal{O}(\max\{|X|^3; (|N| + |\Psi|)\})$.

After the execution of Algorithm RA-TT for each tri-task $\langle v_i, v_j \rangle$ of a workflow, we obtain set A^{ij} of all feasible resource allocations RA^{ij} for $\langle v_i, v_j \rangle$, i.e. a set of pairs (X_χ, X_θ). Now, in order to find a feasible resource allocation for the whole workflow, feasible resource allocations for all of its tri-tasks have to be found. This will be discussed in Sect. 5.4. However, first, in Sect. 5.3, the case of dependent tri-tasks will be analyzed.

5.3 Dependent tri-tasks

As has been mentioned in Sect. 5.2, it is possible to define set A^{ij} for each tri-task $\langle v_i, v_j \rangle$ of a workflow. However, at this stage the most difficult phase begins. Notice, namely, that in a workflow there exist many chains of dependent tri-tasks.

Definition 7 Two tri-tasks $\langle v_i, v_j \rangle$ and $\langle v_k, v_m \rangle$ are called *dependent*, if one of the following three cases occurs (Fig. 6):

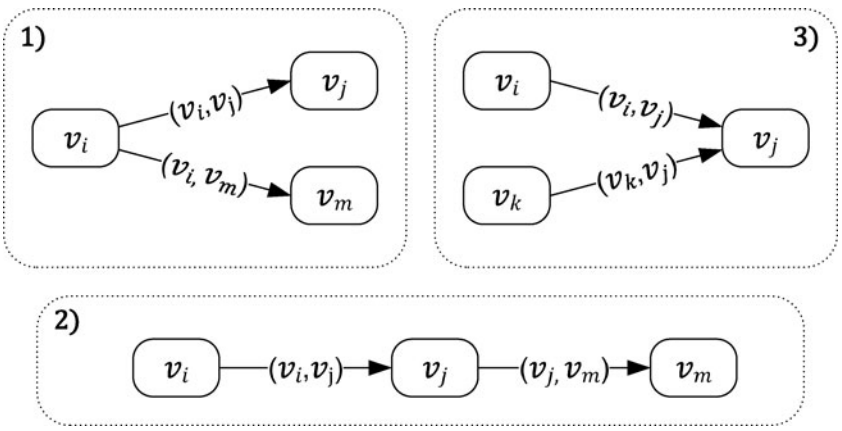

Fig. 6 Three cases of dependent tri-tasks

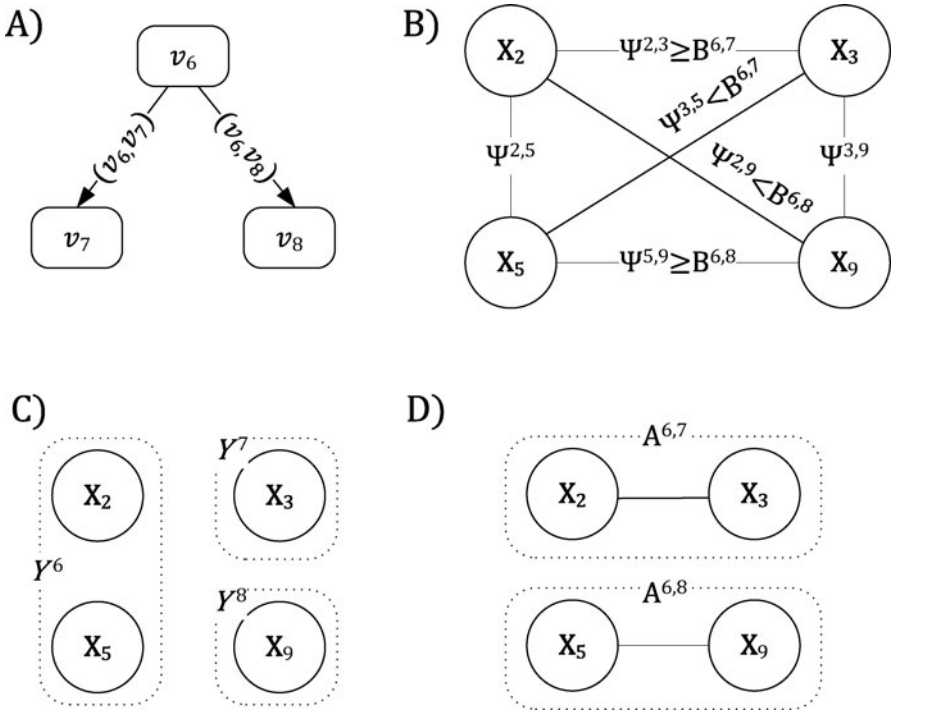

Fig. 7 Illustration for Example 1

(1) $i = k$, or
(2) $j = k$, or
(3) $j = m$.

For example, if we have two tri-tasks $\langle \nu_i, \nu_j \rangle$ and $\langle \nu_j, \nu_k \rangle$ (case 2), then we cannot make resource allocations for these two tri-tasks independently, since task ν_j binds them—it is the finishing element of the first tri-task and, simultaneously, the starting element of the second one. As a result, after choosing a pair $(X_\chi, X_\theta) \in A^{ij}$ as a resource allocation for tri-task $\langle \nu_i, \nu_j \rangle$, such a resource allocation (X_θ, X_ρ) has to be chosen from the set A^{jk} whose first element is X_θ. Similar dependencies occur for cases 1 and 3. The condition of a common node in resource allocations for two dependent tri-tasks has to be satisfied for each two dependent tri-tasks over the whole workflow.

Thus, having defined feasible resource allocations for all tri-tasks a workflow is not a sufficient condition for the existence of a feasible resource allocation for the entire workflow. Let us show it on a simple example, regarding case 1 of Definition 7.

Example 1 Assume a part of a workflow W with 3 computational tasks ν_6, ν_7, ν_8 and 2 transmission tasks (ν_6, ν_7), (ν_6, ν_8), as shown in Fig. 7A. The required bandwidths for transmission tasks (ν_6, ν_7), (ν_6, ν_8) are $B^{6,7}$ and $B^{6,8}$, respectively. Next assume a part of the grid structure with resource nodes X_2, X_3, X_3, X_9 and corresponding bandwidths $\Psi^{\mu\nu}$ between nodes presented in Fig. 7B. Figure 7C presents sets Y^6, Y^7, Y^8 of capable nodes for tasks ν_6, ν_7, ν_8, respectively. In such a case, after the execution of Algorithm RA-TT for tri-task $\langle \nu_6, \nu_7 \rangle$, we will obtain the only feasible resource allocation $RA^{6,7} = (X_2, X_3)$ whereas for tri-task $\langle \nu_6, \nu_8 \rangle$ the only feasible resource allocation is $RA^{6,8} = (X_5, X_9)$. Thus, the resulting sets of all feasible allocations for these tri-tasks are $A^{6,7} = \{(X_2, X_3)\}$ and $A^{6,8} = \{(X_5, X_9)\}$, as shown in Fig. 7D. Since task ν_6 cannot be executed in nodes X_2 and X_5 at the same time, there is no feasible resource allocation for workflow W.

Let us now pass to the problem of finding a feasible resource allocation for a given workflow.

5.4 Algorithm RA-W

In this section we present an algorithm for finding feasible resource allocations for the entire workflow.

Let us first represent a feasible resource allocation RA_W for workflow W with respect to Definition 4 as a function $w : V \to X$, where $(w(i) = \chi) \Leftrightarrow (\nu_i \otimes X_\chi)$. Then, according to Definition 7, for each pair $(\langle \nu_i, \nu_j \rangle, \langle \nu_k, \nu_m \rangle)$ of dependent tri-tasks the following conditions must hold:

$$i = k \implies \exists_{X_\chi} \left[(X_\chi, X_\theta) \in A^{ij} \wedge (X_\chi, X_\rho) \in A^{im} \right] \quad (3)$$

$$j = k \implies \exists_{X_\theta} \left[(X_\chi, X_\theta) \in A^{ij} \wedge (X_\theta, X_\rho) \in A^{jm} \right] \quad (4)$$

$$j = m \implies \exists_{X_\rho} \left[(X_\chi, X_\rho) \in A^{ij} \wedge (X_\theta, X_\rho) \in A^{kj} \right] \quad (5)$$

In other words, function w assigns each computational task to a resource node in such a way that there is no conflict between dependent tri-tasks.

We will now show how to find a feasible resource allocation RA_W (i.e. satisfying conditions (3)–(5) for each pair of tri-tasks) for a given workflow W. Let us define the sets A_L^{ij} and A_R^{ij} for each tri-task $\langle v_i, v_j \rangle$ by:

$$\begin{aligned} A_L^{ij} &= \left\{ X_l : \exists_{(X_l, X_r)} (X_l, X_r) \in A^{ij} \right\} \\ A_R^{ij} &= \left\{ X_r : \exists_{(X_l, X_r)} (X_l, X_r) \in A^{ij} \right\} \end{aligned} \tag{6}$$

i.e. A_L^{ij} is the set of resource nodes occurring as left elements of pairs RA^{ij} in A^{ij}, and A_R^{ij} as right elements. Then, for each computational task v_j, $j = 1, 2, \ldots, |V|$, we define set A^j as:

$$A^j = \prod_i A_R^{ij} \cap \prod_j A_L^{jk} \tag{7}$$

which is a set of nodes where task v_j can only be executed with respect to both computational and transmission requirements of all tri-tasks in which v_j occurs.

And finally, a new set A^{ij}, denoted A_W^{ij}, is constructed thus:

$$A_W^{ij} = A^{ij} \cap \left(A^i \times A^j \right) \tag{8}$$

Thus, for each tri-task $\langle v_i, v_j \rangle$ the set A_W^{ij} contains all resource allocations $RA_W^{ij} = (X_\chi, X_\theta)$ which enable to maintain conditions (3)–(5) over the entire workflow W. Consequently, if none of the sets A_W^{ij} is empty, there must exist a feasible resource allocation RA_W for workflow W. Otherwise (i.e. at least one $A_W^{ij} = \emptyset$), there is no feasible RA_W.

Now, since each set A_W^{ij} may contain more than one element, the problem of choosing one $RA_W^{ij} \in A_W^{ij}$ for each tri-task $\langle v_i, v_j \rangle$ appears. This is necessary for finding a particular feasible resource allocation RA_W for workflow W. Each function w satisfying the following conditions:

$$\forall_{v_j \in V} \; \exists_{\theta \in \{1, \ldots, |X|\}} \; w(j) = \theta \tag{9}$$

$$(X_\chi, X_\theta) \in A_W^{ij} \quad \Longrightarrow \quad w(i) = \chi \tag{10}$$

$$(X_\theta, X_\rho) \in A_W^{jk} \quad \Longrightarrow \quad w(k) = \rho \tag{11}$$

defines a feasible resource allocation RA_W with respect to Definition 4.

In other words, each node j, $j = 1, 2, \ldots, |V|$ of the workflow representing a computational task must be assigned a number θ indicating a resource node of the grid. Then its incoming arcs (transmission tasks) must be covered by pairs (resource allocations) with the right element equal to θ, whereas its outgoing arcs must be covered with the left element equal to θ. The number θ does not need to be unique, i.e. more than one node of the workflow may be assigned to the same resource node.

Let us give the following illustration. We can treat pairs $RA_W^{ij} = (X_\chi, X_\theta)$ from sets A_W^{ij} as domino bones with χ spots on the left end, and θ spots on the right end (the orientation is important). The problem consists in covering all arcs of workflow W with those domino bones in such a way that arc (v_i, v_j) is covered with a bone from set A_W^{ij} (left end to start of the arc, right end to finish of the arc) and, of course, for each node every attached bone must have the same number of spots on the node-adjacent end.

Summarizing, the following algorithm finds a feasible resource allocation RA_W for workflow W:

Algorithm RA-W

1. Execute Algorithm RA-TT for each tri-task $\langle v_i, v_j \rangle$ to find the sets A^{ij}. If at least one $A^{ij} = \emptyset$, then no feasible RA_W exists and STOP.
2. Find sets A_L^{ij} and A_R^{ij} from (6) for each tri-task $\langle v_i, v_j \rangle$.
3. Find set A^j from (7) for each task v_j, $j = 1, 2, \ldots, |V|$. If at least one $A^j = \emptyset$, then no feasible RA_W exists and STOP.
4. Find set A_W^{ij} from (8) for each tri-task $\langle v_i, v_j \rangle$.
5. Find a function w satisfying conditions (9)–(11).

After the execution of Algorithm RA-W, a feasible resource allocation RA_W for the considered workflow W with respect to Definition 4 is found, defined by the function w found in step 5 of the algorithm. Obviously, there can exist many functions w satisfying conditions (9)–(11) and, as a result, many feasible resource allocations. Algorithm RA-W finds just one of them. However, the important thing is that finding any function w is very easy. It is simply enough to choose any pair of precedence-related computational tasks (i.e. an arc of workflow W) (v_j, v_m) as the starting one, and to cover it by any $RA_W^{jm} \in A_W^{jm}$. That first step defines computational nodes for tasks v_j and v_m. Then, all dependent arcs (v_i, v_j), i.e. coming into the node where v_j is executed, and all dependent arcs (v_j, v_k), i.e. going out of the node where v_j is executed, are covered by proper elements $RA_W^{ij} \in A_W^{ij}$ and $RA_W^{jk} \in A_W^{jk}$, respectively. The same has to be done for the node where v_m is executed. The process continues until all arcs of workflow W are covered. Of course, it may be simplest to set $v_j = v_1$ in the first step, take any $RA_W^{1m} \in A_W^{1m}$ in order to assign node to tasks v_1 and v_m, then to all other successors of task v_1, and to go on as described above. But it is not obligatory to start with task v_1. Below we will show that starting with any arc (v_j, v_m) is enough to find a function w in step 5 of Algorithm RA-W, provided that steps 1–4 have been performed correctly. In other

words, defining sets A_W^{ij} according to RA-W guarantees that the covering will continue until all computational tasks have assigned nodes, and there is no danger of getting stuck.

Proposition 1 *Algorithm RA-W always finds a feasible resource allocation RA_W for workflow W if it does exist. Moreover, a feasible allocation can always be found regardless of which pair of precedence-related computational tasks is allocated first.*

Proof Assume that we have just covered an arc (ν_j, ν_m), i.e. some nodes X_θ, X_γ have been assigned to precedence-related computational tasks ν_j and ν_m, respectively. According to step 4 of RA-W, there must exist a non-empty set A_W^{jm} containing a pair (X_θ, X_γ). Taking into account (8) two conditions must hold simultaneously: (i) $(X_\theta, X_\gamma) \in A^{jm}$ and (ii) $(X_\theta, X_\gamma) \in A^j \times A^m$. From (ii) it follows directly that $X_\theta \in A^j$. Thus, in step 3 of RA-W, the set A^j must have been defined such that $X_\theta \in A^j$, which, according to (7) and (6) means that $\forall_{\langle \nu_i, \nu_j \rangle} \exists_{(X_\chi, X_\theta)} (X_\chi, X_\theta) \in A^{ij}$ and $\forall_{\langle \nu_j, \nu_k \rangle} \exists_{(X_\theta, X_\rho)} (X_\theta, X_\rho) \in A^{jk}$.

Since equations (6) have been used in step 2 of RA-W, $A^{ij} = \emptyset$ and $A^{jk} = \emptyset$ must hold, otherwise step 2 would never have been reached according to the stop criterion in step 1. In consequence, task ν_j may be assigned to node X_θ since it is always possible to assign any of its predecessors to some node X_χ, as well as any of its successors to some node X_ρ. The same argument can be applied to task ν_m of the arc (ν_j, ν_m). Thus, the covering based on sets A_W^{ij} defined by Algorithm RA-W can always continue until a feasible resource allocation (function w) is obtained. □

Let us now briefly analyze the complexity of Algorithm RA-W.

The complexity of the first step is $\mathcal{O}(|V|^2 \cdot \max\{|X|^3; (|N| + |\Psi|)\})$. $\mathcal{O}(|V|^2 \cdot |X|^2)$ is the complexity of steps 2, 3, and 5, whereas $\mathcal{O}(|V|^2 \cdot |X|^4)$ is the complexity of step 4. As a result, the complexity of the RA-W algorithm is $\mathcal{O}(|V|^2 \cdot \max\{|X|^4; (|N| + |\Psi|)\})$.

Thus, finding a feasible resource allocation for a given workflow can be done in polynomial time. Of course, Algorithm RA-W can be used as a full enumeration scheme, i.e. an exact algorithm finding all possible feasible resource allocations. To this end, all possible functions w have to be found in step 5. This, however, results in exponential complexity of the algorithm, equal to $\mathcal{O}((|X|^2)^{|V|^2})$, as all possible allocations have to be examined.

6 Conclusions and future research

In this paper we have attempted to fill the gap between the fields of grid resource management and project scheduling. The problem of allocating grid resources to workflow applications has been considered. Grid resources have been divided into two types: computational resources and network resources. Accordingly, computational tasks of a workflow as well as transmission tasks have been distinguished. The problem consists in allocating grid computational resources to computational tasks, as well as grid network resources to transmission tasks in such a way that resource demands of all tasks are satisfied. Processors of different types constitute computational resources considered in this paper, whereas the bandwidth is the network resource for which transmission tasks have to apply. We have shown under which assumptions the defined grid resource allocation problem can be formulated, and we solved in the framework of project scheduling.

The main contribution of this paper lies in two features. Firstly, a mathematical model of the problem considered, including network as a resource, has been presented. The model is very general, and may serve as a basis to the problem of scheduling workflows on a grid. Secondly, an approach to the problem of finding feasible resource allocations for a given workflow has been discussed. The approach is formalized as the presented RA-W algorithm. The algorithm finds a computation- and transmission-feasible allocation of resource nodes of a grid to computational tasks of a workflow in polynomial time. It may also be used, under exponential complexity, to find all possible feasible resource allocations.

In future research we plan to pass to the scheduling phase. In that context, Algorithm RA-W may be used as an enumeration scheme, and different algorithms may be tested as scheduling policies on grid local resources. Moreover, heuristic algorithms for resource allocation can be proposed and tested along with various strategies for local scheduling. Finally, further extensions of the model presented can be considered, as has been mentioned in Sect. 3.2.

Acknowledgements This work has been funded by the Polish Ministry of Science and Higher Education as a research project in the years 2010–2012.

We express our gratitude to two anonymous referees for their valuable comments.

References

Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Węglarz, J. (2001). *Scheduling computer and manufacturing processes* (2nd ed.). Berlin: Springer.

Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Węglarz, J. (2007). *Handbook on scheduling: from theory to applications*. Berlin: Springer.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., & Korranda, S. (2003). Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, *1*(1), 25–39.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M. H., Vahi, K., & Livny, M. (2004). Pegasus: mapping scientific workflows onto the grid. In *Proceedings of the 2nd European across grids conference*, Nicosia, Cyprus (pp. 11–20).

Deelman, E., Blythe, J., Jain, S., Gil, Y., Vahi, K., Mandal, A., & Kennedy, K. (2005). Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the 5th IEEE international symposium on cluster computing and grid*, Cardiff, UK (pp. 759–767).

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project scheduling—a research handbook*. Boston: Kluwer.

Foster, I., & Kesselman, C. (1999). Computational grids. In I. Foster & C. Kesselman (Eds.), *The grid: blueprint for a new computing infrastructure* (pp. 15–52). San Mateo: Morgan Kaufmann.

Foster, I., Voeckler, J., Wilde, M., & Zhao, Y. (2002). Chimera: a virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th conference on scientific and statistical database management*, Edinburgh, UK (pp. 37–46).

Józefowska, J., & Węglarz, J. (2006). *Perspectives in modern project scheduling*. New York: Springer.

Kurowski, K., Nabrzyski, J., & Pukacki, J. (2001). User preference driven multiobjective resource management in grid environments. In *Proceedings of cluster computing and the grid conference*, 15–18 May 2001, Australia (pp. 114–122).

Kurowski, K., Nabrzyski, J., Oleksiak, A., & Węglarz, J. (2003). Multicriteria aspects of grid resource management. In J. Nabrzyski, J. Schopf & J. Węglarz (Eds.), *Grid resource management: state of the art and future trends* (pp. 271–293). Boston: Kluwer.

Kurowski, K., Nabrzyski, J., Oleksiak, A., & Węglarz, J. (2006). Grid multicriteria job scheduling with resource reservation and prediction mechanisms. In J. Józefowska & J. Węglarz (Eds.), *Perspectives in modern project scheduling* (pp. 345–373). New York: Springer.

Kurowski, K., Nabrzyski, J., Oleksiak, A., & Węglarz, J. (2008). Multicriteria approach to two-level hierarchy scheduling in grids. *Journal of Scheduling*, *11*(5), 371–379.

Leung, J. Y.-T. (2004). *Handbook of scheduling: algorithms, models, and performance analysis*. London, Boca Raton: Chapman & Hall/CRC.

Mika, M., Waligóra, G., & Węglarz, J. (2003). A metaheuristic approach to scheduling workflow jobs on a grid. In J. Nabrzyski, J. Schopf & J. Węglarz (Eds.), *Grid resource management: state of the art and future trends* (pp. 295–318). Boston: Kluwer.

Nabrzyski, J., Schopf, J., & Węglarz, J. (2003). *Grid resource management: state of the art and future trends*. Boston: Kluwer.

Szalay, A. S., Kunszt, P. Z., Thakar, A., Gray, J., Slutz, D., & Brunner, R. J. (2000). Designing and mining multi-terabyte astronomy archives: the Sloan Digital Sky Survey. *SIGMOD Record*, *29*, 451–462.

Wulz, C.-E. (1998). CMS concept and physics potential. *Proceedings of the American Institute of Physics Conference*, *444*(1), 467–478.

Węglarz, J., Józefowska, J., Mika, M., & Waligóra, G. (2010). Project scheduling with finite or infinite number of activity processing modes—a survey. *European Journal of Operational Research* (to appear).