Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

# Comparison of selected algorithms for scheduling workflow applications with dynamically changing service availability*

Paweł CZARNUL

(*Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Gdansk 80-233, Poland*)

E-mail: pczarnul@eti.pg.gda.pl; pczarnul@gmail.com

Received Sept. 29, 2013; Revision accepted Mar. 7, 2014; Crosschecked May 4, 2014

**Abstract:** This paper compares the quality and execution times of several algorithms for scheduling service based workflow applications with changeable service availability and parameters. A workflow is defined as an acyclic directed graph with nodes corresponding to tasks and edges to dependencies between tasks. For each task, one out of several available services needs to be chosen and scheduled to minimize the workflow execution time and keep the cost of service within the budget. During the execution of a workflow, some services may become unavailable, new ones may appear, and costs and execution times may change with a certain probability. Rescheduling is needed to obtain a better schedule. A solution is proposed on how integer linear programming can be used to solve this problem to obtain optimal solutions for smaller problems or suboptimal solutions for larger ones. It is compared side-by-side with GAIN, divide-and-conquer, and genetic algorithms for various probabilities of service unavailability or change in service parameters. The algorithms are implemented and subsequently tested in a real BeesyCluster environment.

**Key words:** Dynamic scheduling of workflow applications, Workflow management environment, Scheduling algorithms
**doi:**10.1631/jzus.C1300270 **Document code:** A **CLC number:** TP302

## 1 Introduction

Integration of services has become one of the most important challenges in current and future information technology systems. In a service oriented environment, independent providers offer services capable of executing particular tasks. Providers define conditions including execution time and cost on which other users can access their services.

Usually complex tasks are modeled as workflows composed of smaller tasks (Yu *et al*., 2005; Yu and Buyya, 2006b; Wieczorek *et al*., 2009). Services need to be found and executed for each task so that a certain quality of service (QoS) goal is optimized and possibly other constraints are met. Services have several parameters assigned to them including cost and execution time. For instance, a user running a workflow may be interested in the selection of such services so that the workflow execution time is minimized and the cost of the selected services does not exceed a given threshold. In real open service-based environments, services may appear or disappear at any time due to decisions made by their providers or accessibility issues due to hardware failure. Furthermore, QoS parameters such as execution time and cost may change dynamically in time. If this is the case, reselection of services is needed to choose an alternative best service or to optimize the goals since new services have appeared or conditions have changed. The total execution time of a workflow application needs to consider the execution of both particular services selected by a scheduling algorithm and the latter, possibly several times if rescheduling is needed. There might be a

* Project partially supported by the Polish National Science Center (No. DEC-2012/07/B/ST6/01516)

trade-off between the running time of an algorithm and its quality which may impact the total workflow execution time. For this reason it is important to compare algorithms because it can give knowledge as to which algorithm would yield the best workflow execution times including running times of the algorithm for dynamic environments and particular workflow types. This paper aims at comparing several algorithms in a dynamic environment with dynamically changing service availability including adopting integer linear programming (ILP) (Zeng *et al*., 2003; Aggarwal *et al*., 2004a; 2004b; Genez *et al*., 2012) for dynamic scheduling, the genetic algorithm (Yu and Buyya, 2006a), divide-and-conquer (Yu *et al*., 2005), and GAIN (Sakellariou *et al*., 2007).

## 2 Related work and algorithms

There are three different, albeit, related approaches and problems considered in this paper.

1. Task scheduling considers mapping of workflow tasks to available resources so that workflow execution time is minimized (Blythe *et al*., 2005; Yu and Buyya, 2005). The workflow is usually considered as a directed acyclic graph (DAG) in which nodes correspond to tasks and edges to communication (Yu *et al*., 2008). In the classic model, task execution times are known in advance, and neither data flows nor other metrics such as costs are usually considered. This is also called best-effort based workflow scheduling (Yu *et al*., 2008). Mapping of workflow nodes to services installed on particular resources to optimize makespan also falls into this category (Chin *et al*., 2010). The general problem of assignment of tasks to nodes is NP-complete (Blazewicz *et al*., 1993; Braun *et al*., 1999). Several algorithms were proposed for task scheduling such as min-min (Braun *et al*., 1999; Deelman *et al*., 2005), round-robin, and random allocation methods (Deelman *et al*., 2005). The heterogeneous earliest finish time (HEFT) algorithm is preferred over the genetic algorithm (GA) (Wieczorek *et al*., 2005), and both of them are preferred over a myopic algorithm using locally optimal decisions for large and complex workflows (Wieczorek *et al*., 2006). HEFT and CPOP were also presented by Topcuoglu *et al*. (2002) and demonstrated as giving high performance and short scheduling time for a network of heterogeneous processors. Simulated annealing (SA) is preferred over an approach based on game theory, random, and best of *n* random choosing (Young *et al*., 2003). Several algorithms were listed by Yu *et al*. (2008), including: individual scheduling based on scheduling an individual task, list scheduling with prioritizing tasks and resource selection for the tasks based on their priorities, batch mode algorithms such as min-min and max-min, along with metaheuristics such as GAs and SA. For optimization of workflow makespan, adaptive list scheduling for service (ALSS) is favored over AHEFT, SLACK, max-min, min-min, and myopic algorithms (Chin *et al*., 2010). Mika *et al*. (2011) considered workflows with both computational and transmission types of tasks as well as a grid model with resource and non-resource nodes. Tasks compete for resource nodes such as processors while non-resource nodes reflect the network topology. The authors considered task parameters such as the size, the execution time function for a processor with certain parameters, and the number of processors required. Then feasible resource allocations were found with assignment of computational tasks to resource nodes and transmission tasks to links with required bandwidth. Actual scheduling was listed as future research. Optimization of workflow makespan in cloud environments was discussed by Lin and Lu (2011) who proposed scalable-HEFT (SHEFT) that allows resources to scale at runtime. Rahman *et al*. (2013) presented a dynamic critical-path-based adaptive workflow scheduling algorithm for grid environments that recalculates the critical path at successive steps. The proposed DCP-G algorithm compares favorably to others such as min-min, myopic, HEFT, GRASP, and GA.

2. For scheduling in utility grids (Yu and Buyya, 2006b) or workflow scheduling in grids (Yuan *et al*., 2007) for each task $t_i$, we distinguish a set of services $S_i$ out of which only one is to be chosen to execute $t_i$. Other attributes such as service costs have been considered by Yu and Buyya (2006a). If we consider the execution time of a service only, the problem would be similar to task scheduling in the sense of resource allocation for minimization of the total execution time. As considered by Yu *et al*. (2005) and Yu and Buyya (2006b), the goal is to find the best schedule. This is defined as an assignment of $t_i \rightarrow (s_k, t_{ik}^{\mathrm{st}})$ where $s_k$ is a

service able to execute task $t_i$ and $t_{ik}^{st}$ is the starting time for the execution of task $t_i$ on service $s_k$. Execution of $t_i$ and $t_j$ on one $s_k$ must not overlap and the workflow execution time should be minimized while keeping the cost of the selected services below a predefined minimum. Both deadline and budget, which constrain scheduling with bounds on the execution time and cost, respectively, were considered by Yu *et al*. (2008). Several service levels (pairs of cost/execution time) for each task of the workflow and several heuristic approaches were considered by Yuan *et al*. (2007). Choosing a service level corresponds to an assignment of a service to a task as above and leads to a trade-off between time and cost. A divide-and-conquer (DaC) approach for solving the workflow mapping problem by grouping workflow nodes into partitions was proposed by Yu *et al*. (2005), distributing the deadline and using local solutions for the partitions. It is compared to a greedy algorithm in which all tasks have the same deadline and a deadline algorithm in which deadlines are the same for tasks on the same level. It is demonstrated that the proposed algorithm is similar in execution time while being less costly for pipeline, parallel, and hybrid applications compared to the others. LOSS and GAIN algorithms for budget constrained scheduling were discussed by Sakellariou *et al*. (2007). LOSS and GAIN tune the schedule resulting from optimization of only time or only cost to meet the cost constraint and optimize the execution time. Similarly, an algorithm was proposed by Kyriazis *et al*. (2008) that maps workflow processes to service instances so that the user's requirements are met concerning availability level or cost. Then, replacements of the initial mapping with a better level of QoS were found. Yu *et al*. (2008) compared GA, deadline distribution among partitions, and back-tracking algorithms for a deadline constrained scheduling problem. For budget constrained scheduling, LOSS and GAIN were compared to GA. Yao *et al*. (2010) minimized the execution cost and kept the execution time before the deadline. Integer programming (IP) was proposed to distribute the deadline among windows and apply local optimization within the windows. The same goal was optimized by Abrishami *et al*. (2010) using the concept of partial critical paths. This approach schedules critical nodes first to meet the deadline and then recursively all other nodes using the scheduled nodes as deadlines. Recently, Genez *et al*. (2012) proposed to use ILP for scheduling workflows in a cloud environment for minimization of computing cost with an upper bound on makespan. Genez *et al*. (2012) considered scheduling at discrete time steps, while this paper allows scheduling execution of a service at any moment expressed as a real value (variables $t_i^{st}$). This approach has the benefit of being more accurate and flexible, because in reality the services can start at any time. Yassa *et al*. (2013) presented a multi-objective solution for workflow scheduling in cloud environments that considers performance, cost, as well as energy consumption to obtain a good trade-off between these metrics. The authors proposed multi-objective particle swarm optimization combined with dynamic voltage and frequency scaling. Bittencourt and Madeira (2011) presented the algorithm HCOC used for workflow scheduling in hybrid clouds, i.e., both private and public clouds. The goal of the algorithm is to reduce makespan to fit the imposed deadline and maintain reasonable cost. If resources in the private cloud are not sufficient, others from a public cloud can be engaged at a cost. Varalakshmi *et al*. (2011) presented the optimal workflow based scheduling (OWS) algorithm that focuses on scheduling cloud workflows and achieves better CPU utilization than FCFS and back-filling.

3. In the context of typical business interactions, the QoS service selection/workflow composition problem is stated. Similar to the scheduling problem, a graph of tasks with dependencies is considered. Additionally, a set of services is distinguished for each task, out of which only one should be selected for execution. Compared to the scheduling problem in utility grids, there are usually two differences in the formulation. Firstly, in QoS service selection, many more quality attributes important in business are considered, such as execution time, cost, availability (Patel *et al*., 2003; Zeng *et al*., 2003; 2004; Canfora and Penta, 2004), accessibility (Patel *et al*., 2003), fidelity (Cardoso *et al*., 2002) or conformance (Patel *et al*., 2003), security (Patel *et al*., 2003), and reputation (Zeng *et al*., 2003). Secondly, usually no overlapping of services executing different tasks is considered. The goal is to select proper services so that a function of the QoS metrics is minimized, possibly also with additional conditions imposed on some of them (Canfora and Penta, 2004; Canfora *et al*.,

2005b), e.g., the cost not to exceed the given budget. A service selection/workflow composition problem is NP-hard (Zeng *et al*., 2003). Traditionally, service selection/workflow composition ILP (Zeng *et al*., 2003; Aggarwal *et al*., 2004a; 2004b) is used to solve the problem. Integer programming was compared to global enumeration by Gao *et al*. (2005) for service selection in a workflow. As for workflow application scheduling, market-based approaches (Geppert *et al*., 1998; Stricker *et al*., 2000) were suggested as well.

GAs can be used for all task scheduling (Wieczorek *et al*., 2005), workflow application scheduling in utility grids (Yu and Buyya, 2006a), and service selection (Canfora and Penta, 2004). GAs are preferred for a large number of concrete services per abstract service (Canfora *et al*., 2005a); otherwise, integer programming is better.

## 3 Motivations and contribution

In this paper, we consider an environment in which existing services may become unavailable and their key parameters, such as cost and execution time, change dynamically, as well as new services being capable of executing particular tasks may appear. In high performance computing (HPC) and grid environments, resources in various administrative domains may become unavailable at any time or new ones may be added. Processor usage of machines in environments without queuing systems, such as laboratories, may change unpredictably, possibly affecting the execution time of services that are installed there. In business environments, providers are free to change availability or the conditions in which services are offered, and to withdraw or advertise new services. Many of them do this on a daily basis as indicated by search engines.

As discussed by Yu *et al*. (2008), there are several approaches to this problem. Following Deelman *et al*. (2004) for task scheduling, the workflow can be divided into parts which are scheduled subsequently as the workflow is running. However, it was suggested by Wieczorek *et al*. (2006) that full-graph analysis would be a preferred approach, instead of the partitioning strategy, for this problem. In this work, the iterative rescheduling concept (Yu *et al*., 2008) is adopted for a more complex workflow scheduling of services. Several algorithms are compared in this context. We assume that services are first selected and scheduled before the execution of the workflow actually starts (Fig. 1). This is done by considering that all services are available at that moment. Before each task is executed, the environment checks whether the service selected as the best one for the given task is available or if the conditions of the services have changed considerably or new services for the task have become available. If availability or conditions have changed, a rescheduling algorithm is spawned and a new service is selected out of those available. The workflow application shown in Fig. 1, with various numbers of steps and parallel paths, has many applications in the following areas:

1. Business area: tasks $t_1$ and $t_2$ may correspond to purchases of components for a production company, $t_3$ to integration of components and the added value by the company and finally tasks $t_4$, $t_5$, and $t_6$ to distribution of the product on various markets. This scenario was analyzed by Czarnul (2013a).

2. Scientific area: tasks $t_1$ and $t_2$ may refer to parallel mathematical computations, parallel recognition of images against various templates, $t_3$ to integration of results, while $t_4$, $t_5$, and $t_6$ to further parallel recognition against selected sets of templates based on the results of the first stage. Such workflows for analysis of multimedia streams were considered by Czarnul *et al*. (2012).

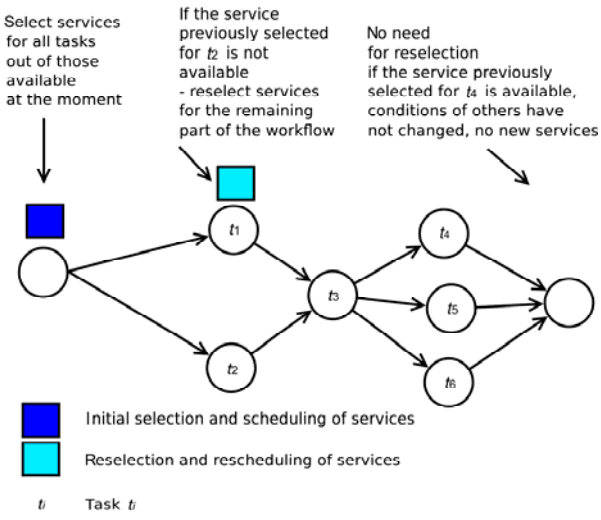


**Fig. 1 Workflow application scheduling with changing service availability and parameters**

Thus, such workflows composed of stages with parallel paths are considered as typical of a variety of workflow applications.

There are several trade-offs involved in this solution. Solving the problem using global knowledge (GLOBAL) of the system takes more time than solving using local data (LOCAL) by partitioning the problem into subproblems and solving each one individually. However, the former can potentially optimize the given goal better. Using a more time-consuming, possibly optimal (OPT) algorithm may yield a better schedule than a heuristic algorithm (HEU), but the execution time of the algorithm needs to be added to the total workflow execution time. This problem is similar to the comparison of heuristics and meta-heuristics for task or best-effort scheduling (Yu *et al*., 2008). Optimization by meta-heuristics considers the whole workflow while heuristics takes into account partial information about the workflow.

The time and cost trade-off for scheduling parallel applications on utility grids was considered by Garg *et al*. (2010), in which the input was a set of applications with submission and execution times, including the number of CPUs required, as well as the resources, and the output was a mapping of the applications to the resources.

This paper investigates these relationships for scheduling in utility grids and dynamic changes of service availability and parameters during the workflow execution. The contributions of this paper are as follows:

1. Perform a detailed side-by-side performance-cost comparison of ILP, divide-and-conquer, GA, and GAIN algorithms in a dynamic environment with changeable service availability and potentially other QoS parameters. The optimization goal is to minimize the workflow execution time with an upper bound on the cost of selected services. The aim is to assess whether using global data and more refined but slower algorithms should be preferred over faster heuristics and considering only local data when scheduling. Comparison is especially important when workflow applications are run repeatedly using different data sets. The latter is called running parameter sweep (PS) workflows for which various execution models have been analyzed (Chirigati *et al*., 2012) as well as providing the characterization of various PS patterns.

2. Adopt ILP which is traditionally used for service selection and extend for either optimal or suboptimal solutions for scheduling in utility grids considering scheduling and rescheduling of workflow applications. A similar ILP approach was used by Genez *et al*. (2012) for scheduling workflows in a cloud environment. However, Genez *et al*. (2012) considered scheduling at discrete time steps while this work allows for scheduling services at time points which are real variables, which gives better flexibility and accuracy. Furthermore, a divide-and-conquer approach using the proposed ILP formulation for subproblems is proposed.

## 4 Problem statement

We consider a workflow model well known in the literature (Yu and Buyya, 2006b; Yuan *et al*., 2007), in which a directed graph $G(V, E)$ represents a workflow where nodes $V$ correspond to tasks while edges $E$ denote dependencies between tasks. A task can start only after its predecessors have finished and it is executed nonpreemptively. It is assumed that there is at least one starting node with initial data and one termination node which terminates computations. The starting nodes do not have predecessors. Each node should have a successor except the termination node. Input data is processed by each task and output data is sent to the following tasks. It is assumed that data sizes are known in advance. For each task there is a set of services capable of executing the task out of which only one is selected for execution of the particular task. Each service is installed on a computing node such as a server or a cluster. From this point of view, selection of a service results in selection of a computing node to run the service. However, there can be many services installed on a single node. A service also needs to be scheduled to run at a particular time. If services assigned to different workflow nodes are installed on the same computing node, we assume their execution must not overlap. The service selected for the task to which other tasks are connected will be executed only after all the predecessors have finished. Table 1 contains a summary of the notations used in the algorithm.

Taking the notations into account, the problem is now stated as follows:

Input data: $G(V, E)$, $t_i$, $S_i$, $c_{ij}$, $P_{ij}$, $N_{ij}$, $d_i=d_i^{\text{in}}$, $f_{ti}$, $d_i^{\text{out}}$, $t_{N_{ij}N_{kl}}^{\text{startup}}$, $\text{bandwidth}_{N_{ij}N_{kl}}$, $t_{ijkl}^{\text{tr}}$, distribution of $d_i^{\text{out}}$ to the nodes following $t_i$.

**Table 1 Summary of notations**

| Symbol | Description |
|---|---|
| $t_i$ | Task $i$ of the workflow application |
| $v_i \in V$ | The vertex of graph $G(V, E)$ that corresponds to task $t_i$ of the workflow application |
| $S_i = \{s_{i1}, ..., s_{i\|S_i\|}\}$ | A set of alternative services each able to execute task $t_i$, out of which only one must be selected to execute task $t_i$ |
| $c_{ij} \in \mathbb{R}$ | Cost of processing a unit of data by service $s_{ij}$ |
| $P_{ij}$ | Provider of service $s_{ij}$ |
| $N_{ij}$ | Node on which service $s_{ij}$ runs; each $s_{ij}$ is installed on a computing node $N_{ij}$ |
| $d_i = d_i^{\text{in}} \in \mathbb{R}$ | Size of data required and processed by task $t_i$ |
| $d_i^{\text{out}} \in \mathbb{R}$ | Size of data produced by task $t_i$ |
| $d_{ij}^{\text{in}} \in \mathbb{R}, d_{ij}^{\text{out}} \in \mathbb{R}$ | Sizes of the input and output data for service $s_{ij}$ |
| $d_{ijkl} \in \mathbb{R}$ | Size of data sent from service $s_{ij}$ to service $s_{kl}$ |
| $f_{t_i}$ | A parameter representing how the size of output data depends on the size of input data: $d_i^{\text{out}} = f_{t_i}(d_i^{\text{in}})$ |
| $t_{ij}^{\text{exec}}(d_{ij}^{\text{in}}) \in \mathbb{R}$ | Execution time of service $s_{ij}$ |
| $t_{ijkl}^{\text{comm}}$ | Communication time of data of size $d_{ijkl}$ sent from $s_{ij}$ to $s_{kl}$ |
| $t_{N_{ij}N_{kl}}^{\text{startup}}$ | Startup time between $N_{ij}$ and $N_{kl}$ |
| $\text{bandwidth}_{N_{ij}N_{kl}}$ | Bandwidth between $N_{ij}$ and $N_{kl}$ |
| $t_{ijkl}^{\text{tr}} \in \mathbb{R}$ | Additional time needed if output data from service $s_{ij}$ is sent as input data to service $s_{kl}$ if $P_{kl} \neq P_{ij}$ |
| $t_i^{\text{st}} \in \mathbb{R}$ | Time at which service $s_{ij}$ chosen to execute $t_i$ starts processing it |
| $t^{\text{workflow}} \in \mathbb{R}$ | Time when the last service finishes |
| $B \in \mathbb{R}$ | Available budget |

$1 \le i \le |V|$, $1 \le j \le |S_i|$, $1 \le k \le |V|$, $1 \le l \le |S_k|$

Variables (i.e., values that determine the solution): selection of exactly one service $s_{ij}$ per $t_i$ and orderings of services that execute on the same computing nodes, i.e., for each computing node on which services run, their order must be determined if not specified by the workflow graph.

Goal: minimization of the workflow execution time $t^{\text{workflow}}$ with a constraint on the total cost, i.e., $\sum d_{ij}^{\text{in}} c_{ij} < B$.

The following relationships hold true:

1. $d_i^{\text{out}} = f_{t_i}(d_i^{\text{in}})$. For instance, for sorting algorithms, $d_i^{\text{out}} = d_i^{\text{in}}$. For example, integration of a function on a given range returns just one result, image processing returns an image that can be of a size dependent on the size of the input image or small descriptors, multiplication of two square matrices returns one square matrix, and processing a business order returns a confirmation in the form of another document.

$$d_i = d_i^{\text{in}} = \sum_j d_{ij}^{\text{in}}, \tag{1}$$

$$d_i^{\text{out}} = \sum_j d_{ij}^{\text{out}}. \tag{2}$$

$d_i^{\text{out}}$ can be distributed to subsequent tasks in various ways; e.g., a copy can be sent to each node or data can be partitioned for parallel execution by the following tasks. The latter was used in all experiments of this work. In this case the following equations are valid:

$$d_i^{\text{out}} = \sum_{j,l,i,k:(v_i,v_k)\in E} d_{ijkl}, \tag{3}$$

$$d_k^{\text{in}} = \sum_{j,l,i,k:(v_i,v_k)\in E} d_{ijkl}. \tag{4}$$

2. The communication time is modeled as

$$t_{ijkl}^{\text{comm}}(d_{ijkl}) = t_{N_{ij}N_{kl}}^{\text{startup}} + \frac{d_{ijkl}}{\text{bandwidth}_{N_{ij}N_{kl}}}.$$

3. $\forall i,k : (v_i, v_k) \in E$,

$$t_k^{\text{st}} \ge t_i^{\text{st}} + \sum_j t_{ij}^{\text{exec}} + \sum_{j,l} t_{ijkl}^{\text{comm}} + \sum_{j,l} t_{ijkl}^{\text{tr}}, \tag{5}$$

$t_{ij}^{\text{exec}}$ will be larger than 0 only for one $j$ since one service is chosen for execution of $t_i$. Similarly, for the given $i$ and $k$, $t_{ijkl}^{\text{comm}}$ and $t_{ijkl}^{\text{tr}}$ will be larger than 0 only for one pair of $l$ and $k$ since only one service per node $i$ and one per node $k$ will be selected. However, there may be several tasks preceding task $t_k$. As mentioned before, each service is installed on a particular computing node.

$$\forall i : \neg\exists_q (v_i, v_q) \in E,$$

$$t^{\text{workflow}} \ge t_i^{\text{st}} + \sum_j t_{ij}^{\text{exec}} + \sum_{j,l} t_{ijkl}^{\text{comm}} + \sum_{j,l} t_{ijkl}^{\text{tr}}. \tag{6}$$

## 5 Proposed algorithms and methods

Within this work several algorithms are implemented (Table 2) using either global or local knowledge about the workflow. The algorithms are both simpler and faster or employ more time-consuming methods to find a better solution. The algorithms are deployed in a real middleware BeesyCluster that allows workflow editing and subsequent execution using any of the proposed algorithms. The algorithms are tested in BeesyCluster on several workflow applications of varying sizes as described in Section 7.

**Table 2 Summary of the implemented algorithms**

| Algorithm | Description and contribution |
|---|---|
| ILP/ ILPHEU | An ILP based approach. The traditional usage of ILP for service selection (Zeng *et al*., 2003; Aggarwal *et al*., 2004a; 2004b) was extended to prevent overlapping of services executing on the same resource as indicated in Section 5.1. The algorithm either returns an optimal solution within a predefined time limit (algorithm ILP) or the solver returns a reasonable suboptimal solution if it cannot find an optimal one (algorithm ILPHEU). Thus, it is a GLOBAL/OPT algorithm if possible but becomes a GLOBAL/HEU algorithm otherwise. |
| DaC | A known approach (Yu *et al*., 2005) which partitions the initial problem into subproblems which are solved independently and merges the results. In this case, as shown in Section 5.2, the graph is partitioned into sets of nodes which form smaller graphs and the budget available is divided into smaller budgets for the smaller graphs. Each tries to optimize the execution time using ILP (LOCAL/OPT) or ILPHEU (LOCAL/HEU). |
| GA | Application of a known algorithm for scheduling workflow applications by selection of services and scheduling their execution on the resources from which they were published as described in Section 5.3 (GLOBAL/HEU). |
| GAIN | The GAIN algorithm proposed by Sakellariou *et al*. (2007) for mapping tasks to resources applied for selection of services installed on resources (Section 5.4). Firstly, the algorithm selects the cheapest services that satisfy the cost constraint. Afterward, some services are replaced with possibly more expensive ones to still meet the constraint and minimize the execution time. |

### 5.1 ILP-based algorithm

The algorithm tries to solve the problem optimally using ILP. An ILP formulation is proposed. It consists of two parts:

1. Assignment of a particular service $s_{ij}$ to task $t_i$. This is done by forcing exactly one $d_{ij}^{\text{in}}$ to be equal to $d_i$.

2. Since services are installed on computing nodes, i.e., service $s_{ij}$ runs on $N_{ij}$, if there are services running on the same resource and their order is not determined by the workflow graph, their execution must not overlap. This is done by ensuring that the times when such services start are spread away from each other sufficiently.

#### 5.1.1 General statements

The equations and inequalities stated in Section 4 will be naturally considered in the ILP formulation along with some extensions (all variables are equal to or greater than 0 in this model):

1. Constraints (1)–(6) imposed in the problem formulation will be stated as constraints of a mixed integer linear programming (MILP) problem (Floudas and Lin, 2005).

2. $t_{ij}^{\text{exec}} = f_{ij}(d_{ij}^{\text{in}})$, e.g., $t_{ij}^{\text{exec}} = t_{ij}^{1} d_{ij}^{\text{in}}$.

3.

$$t_{ijkl}^{\text{comm}} = \begin{cases} 0, & \text{if } N_{ij} = N_{kl}, \\ t_{ijkl}^{\text{startup}} b_{ijkl}^{\text{startup}} + \dfrac{d_{ijkl}}{\text{bandwidth}_{ijkl}}, & \text{otherwise}, \end{cases}$$

which considers startup time. $b_{ijkl}^{\text{startup}}$ is a flag such that

$$b_{ijkl}^{\text{startup}} = \begin{cases} 0, & d_{ijkl} = 0, \\ 1, & d_{ijkl} > 0, \end{cases}$$

and can be modeled as

$$b_{ijkl}^{\text{startup}} \geq \frac{1}{U_{ijkl}} d_{ijkl},$$

$$b_{ijkl}^{\text{startup}} \leq 1, \quad b_{ijkl}^{\text{startup}} \in \mathbb{Z},$$

where $U_{ijkl}$ is set at a large enough value so that $d_{ijkl} / U_{ijkl} < 1$ if $d_{ijkl} > 0$. This can be done since the upper bound on $d_{ijkl}$ is the sum of initial data sizes for the workflow. Since the algorithm will, in particular,

minimize $t^{\text{workflow}}$, it will also minimize startup times, if possible. $U_{ijkl}$ forces to consider the startup time if data is sent from $s_{ij}$ to $s_{kl}$ (i.e., $d_{ijkl}>0$). If $d_{ijkl}=0$, then $b_{ijkl}^{\text{startup}}=0$; if $d_{ijkl}>0$, then $b_{ijkl}^{\text{startup}}=1$. Alternatively, it is possible to drop the $t_{ijkl}^{\text{startup}}$ component for a less accurate estimate on communication. As a consequence, this results in fewer constraints and variables and a shorter running time for an optimization algorithm. This is the case for the practical experiments shown in Section 7.

4. $t_{ijkl}^{\text{tr}}=\begin{cases}0, & \text{if } P_{ij}=P_{kl},\\ \sigma_{ijkl}d_{ijkl}, & \text{otherwise,}\end{cases}$ where $\sigma_{ijkl}>0$ is a given constant. $\sigma_{ijkl}$ allows for considering a delay corresponding to the size of data sent between $s_{ij}$ and $s_{kl}$ if providers of the two services are different, e.g., for translation of data formats. The value of $\sigma_{ijkl}>0$ provides a model for how large this delay is.

5.1.2 Assignment of services to workflow tasks

As stated above, the assignment forces that for each task $t_i$ for only one $j$, $d_{ij}=d_i$, which can be stated as (all variables in this ILP formulation are equal to or greater than 0)

$$\left|d_{ij}^{\text{in}}-\frac{d_i}{2}\right|\geq\frac{d_i}{2}. \tag{7}$$

It denotes whether $s_{ij}$ has been chosen or not. This stems from both $d_{ij}^{\text{in}}\geq 0$ required by linear programming and constraint (1). Constraint (7) can be written (http://lpsolve.sourceforge.net/5.5/, chapter 'Absolute values') as two inequalities for integer programming. Namely, an additional variable $b_{ij}$ is introduced for service $s_{ij}$, which is to take one of the following values:

$$b_{ij}=\begin{cases}0, & d_{ij}^{\text{in}}=0,\\ 1, & d_{ij}^{\text{in}}=d_i.\end{cases}$$

Then, a large constant $M_{ij}$ is chosen so that $M_{ij}\geq\left|d_{ij}^{\text{in}}-d_i/2\right|+d_i/2$ is satisfied, e.g., $M_{ij}=d_i$. Then constraint (7) can be written as two constraints:

$$d_{ij}^{\text{in}}+M_{ij}-M_{ij}b_{ij}-d_i\geq 0, \tag{8}$$

$$-d_{ij}^{\text{in}}+M_{ij}b_{ij}\geq 0, \tag{9}$$

$$b_{ij}\leq 1,\quad b_{ij}\in\mathbb{Z}. \tag{10}$$

Constraints (8) and (9) are equivalent to constraint (7) for ILP as referenced above.

5.1.3 Scheduling services on a node

Some services, especially those published by one provider, can execute on one computing node. The algorithm should consider possible orderings of the services while guaranteeing that the services selected from a given node will not overlap. This is not straightforward with an ILP formulation but must be assured for nodes that feature single processors. The solution incorporates flags $b_{ij}$ used previously to assure that only one service per task is chosen, e.g., for constraints (8) and (9). In this case, starting times $t_i^{\text{st}}$ are split as

$$t_i^{\text{st}}=\sum\nolimits_j t_{ij}^{\text{st}},$$

where

$$t_{ij}^{\text{st}}=\begin{cases}0, & d_{ij}=0,\\ t_i^{\text{st}}, & d_{ij}>0.\end{cases}$$

The execution of service $s_{ij}$, running on node $N_{ij}$, starts at time $t_{ij}^{\text{st}}$ and terminates at time $t_{ij}^{\text{st}}+t_{ij}^{\text{exec}}$, lasting $t_{ij}^{\text{exec}}$. This means that the halftime of the execution of service $s_{ij}$ on node $N_{ij}$ occurs at time $t_{ij}^{\text{half}}=(2t_{ij}^{\text{st}}+t_{ij}^{\text{exec}})/2$. Thus, for every pair of services ($s_{ij}$, $s_{ij}$): $N_{ij}=N_{kl}$, $i\neq k$, which could potentially overlap, the following constraint is added:

$$\left|t_{ij}^{\text{half}}-t_{kl}^{\text{half}}\right|\geq(t_{ij}^{\text{exec}}+t_{kl}^{\text{exec}})/2, \tag{11}$$

meaning that the execution times must be spread away by at least the sum of halves of their lengths. Thus, the model would need to specify constraint (11) for each pair of potentially overlapping services running on nodes which can be executed in any order; i.e., the order of their execution is not imposed by the edges of graph $G$. In the latter case, there is no possibility of overlapping. The following cases can be analyzed:

1. $s_{ij}$ and $s_{kl}$ are not chosen. In this case, $d_{ij}=d_{kl}=0$ and thus we assume $t_{ij}^{\text{exec}}=t_{kl}^{\text{exec}}=0$, and constraint (11) is true.

2. Only one, say $s_{kl}$, is not chosen. In this case, $d_{kl}=t_{kl}^{\mathrm{st}}=0$, then $t_{kl}^{\mathrm{half}}=0$ and constraint (11) becomes $t_{ij}^{\mathrm{half}}=t_{ij}^{\mathrm{st}}+t_{ij}^{\mathrm{exec}}/2\geq t_{ij}^{\mathrm{exec}}/2$, which is satisfied.

3. Both $s_{ij}$ and $s_{kl}$ are selected. In this case, constraint (11) defines the required space between the middles of the execution of the services.

As above, we can express constraint (11) as two inequalities for integer programming. An additional variable $b_{ijkl}$ is introduced for services $s_{ij}$ and $s_{kl}$, indicating one of the two possibilities (order) specified by the absolute value. Then, a large constant $M_{ijkl}$ is chosen so that $M_{ijkl}\geq\left|t_{ij,n}^{\mathrm{half}}-t_{kl,n}^{\mathrm{half}}\right|+(t_{ij,n}^{\mathrm{exec}}+t_{kl,n}^{\mathrm{exec}})/2$ is satisfied. Then constraint (11) will be written as two constraints:

$$t_{ij,n}^{\mathrm{half}}-t_{kl,n}^{\mathrm{half}}+M_{ijkl}b_{ijkl}\geq(t_{ij,n}^{\mathrm{exec}}+t_{kl,n}^{\mathrm{exec}})/2, \quad (12)$$

$$-t_{ij,n}^{\mathrm{half}}+t_{kl,n}^{\mathrm{half}}+M_{ijkl}-M_{ijkl}b_{ijkl}\geq(t_{ij,n}^{\mathrm{exec}}+t_{kl,n}^{\mathrm{exec}})/2, \quad (13)$$

$$b_{ijkl}\leq 1,\quad b_{ijkl}\in\mathbb{Z}.$$

Note that for each service $s_{ij}$, it is easy to generate all successor vertexes up to the terminating vertex: set VS($s_{ij}$) and all predecessors on paths from the source vertex VP($s_{ij}$). Then services to be considered for constraint (11) along with service $s_{ij}$ would be only services running on $V-\mathrm{VS}(s_{ij})-\mathrm{VP}(s_{ij})$ and running on the same node as $s_{ij}$. This helps reduce the number of constraints of type (11).

The model allows parallel communication between the same nodes for various pairs of tasks. There could be additional constraints similar to those for ordering tasks (constraint (11)).

5.1.4 Formulation of the goal

The objective can be formulated exactly as in the original problem formulation, i.e.,

$$\text{minimization of } t^{\mathrm{workflow}}, \quad (14)$$

with an additional constraint $\sum d_{ij}^{\mathrm{in}}c_{ij}<B$, where $B$ is the budget.

The actual implementation uses lp_solve (http://lpsolve.sourceforge.net/5.5/), -presolve, -timeout, and <timeout> as a MILP solver to find assignment of services to tasks according to the proposed model. The parameters instruct the solver to look for a solution within a <timeout> time limit (given in seconds) and pre-solve the problem before optimization is started (rows and columns).

If the algorithm is able to find the solution within the imposed time limit such as 10 or 20 s, then it can be either an optimal (we call this version ILP) or a suboptimal solution (we call it ILPHEU) returned by the solver. If the solver is not able to return any solution within the time frame, it creates a list of tasks which have not been executed yet and executes the following in a loop:

1. Takes the first task from the list and assigns the cheapest service to it.

2. Tries to solve the problem using ILP within the time limit—the number of integer variables has been decreased. If successful, the algorithm terminates; otherwise, the loop is repeated.

### 5.2 Divide-and-conquer algorithm

In this case, we try to reduce the complexity of the ILP-based approach by partitioning the graph into subdomains, solving each one individually and merging results into a global solution. Obviously, it does not necessarily lead to the optimal global solution but can reduce the execution time of the algorithm.

The algorithm works as follows: It defines a batch size, e.g., 4 or 15 tasks. Then, at the moment the algorithm is to be invoked, the tasks not yet executed are divided into subdomains, each containing at most the batch size number of tasks. Then, the algorithm uses the ILP-based algorithm for each subdomain independently. For each subproblem constraint, $\sum_{i\in\mathrm{subproblem}}d_{ij}^{\mathrm{in}}c_{ij}<(B/B_{\mathrm{C}})$ is used, where $B_{\mathrm{C}}=n_{\mathrm{t}}/b_{\mathrm{s}}$, with $n_{\mathrm{t}}$ denoting the number of tasks and $b_{\mathrm{s}}$ the batch size. In some cases, it may be impossible to meet this constraint for a subproblem. In this case, it is possible to set tighter cost bounds for initial batches so that a larger margin remains for the last ones. Alternatively, the algorithm could solve all subproblems it can solve and apply ILPHEU for the remaining problem with the yet unsolved subproblems. The algorithm is generally OPT/LOCAL if subproblems can be solved optimally or HEURISTIC/LOCAL if heuristic algorithms are returned for subproblems.

### 5.3 Genetic algorithm

In a GA, a population of chromosomes is created. Each chromosome represents a complete solution to a

problem. Best (in terms of the fitness function) chromosomes are selected for crossover and creation of chromosomes in the next generation. Mutation is used to leave a local minimum. Best chromosomes in next generations should represent better and better solutions.

Each chromosome consists of the representation shown in Fig. 2. This is analogous to the solution proposed by Czarnul (2010) for an extended problem with service selection and data partitioning.

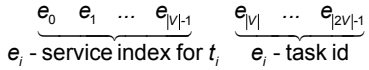


**Fig. 2 Representation of the genetic algorithm**

The first $|V|$ numbers contain indexes of selected services for each task, i.e., $0 \le e_i < |S_i|$, $0 \le i < |V|$. Initial values are chosen by a random selection of services for each task, i.e., $e_i$=random() mod $|S_i|$. The last $|V|$ numbers contain ordering of the $|V|$ tasks. Namely, assuming tasks $t_k$, $t_l$ are executed on one processor (i.e., services on one processor are chosen to execute them), $t_l$ is executed after $t_k$ if $e_i=k$, $e_j=l$: $i<j$, $i \ge |V|$. Initial values are chosen randomly out of the following tasks starting from the initial task. In the next step, its successors are added to the pool of the tasks from which, in the next step, one will be chosen randomly and the procedure is repeated. The fitness function used for each chromosome is equal to $C / t^{\text{workflow}}$ if the cost constraint is satisfied, otherwise 0. In this way, better chromosomes give greater fitness values. In the experiments $C$=10 000 and the population size of 10 chromosomes were used due to the reasonably long time on the evaluation of the schedule corresponding to the chromosome.

The GA runs one or more iterations of the following loop:

1. Assess the fitness function and store values for all chromosomes.

2. Copy the best chromosome to the pool for the next iteration.

3. Select chromosomes for crossover based on their fitness values and perform crossover.

4. Apply mutation.

Crossover for service selection works as follows: a pivot (task) is selected randomly and services for tasks smaller than the pivot are taken from the first chromosome and for larger than the pivot are taken from the second chromosome. The order is taken from one of the two chromosomes with a probability of 0.35 and a new one is generated randomly with a probability of 0.3 (Fig. 3).

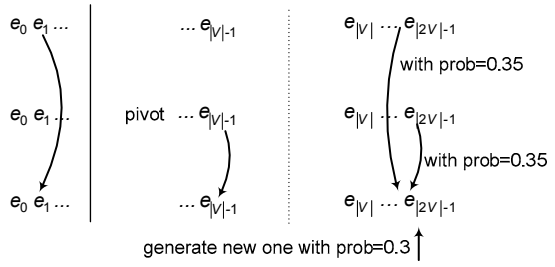


**Fig. 3 Crossover for service selection**

Mutation for service selection is applied as follows: a new service is selected for a task with a probability of 0.1 and a new order is generated with a probability of 0.02 (Fig. 4).

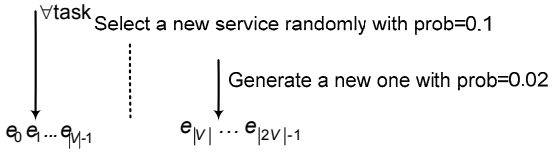


**Fig. 4 Mutation for service selection**

## 5.4 GAIN

This algorithm is analogous to the idea of the GAIN algorithm proposed by Sakellariou *et al*. (2007) and compared to GA (Yu *et al*., 2008). For budget constrained scheduling, the idea is firstly to select the cheapest services for each task to meet the cost constraint. Obviously, the execution time corresponding to this case is large. To minimize the execution time, the algorithm executes the following code shown in Algorithm 1 to select new services still meeting the cost constraint and optimizing the execution time. The algorithm tries to find the service with the largest value of tempGainWeight (lines 10–14) on condition that the total cost of services with the new one does not exceed the given budget. If this service is different from the one previously set for its task, it is substituted (lines 15–20). The algorithm proceeds until there is a new better service that can substitute the former for the task.

**Algorithm 1** GAIN algorithm

```
1   do {
2    prevWorkflowEval=EvaluateWorkflow(...);
3    bestGainWeight=0;
4    bestTask=−1;
5    bestServiceIndex=−1;
6    for each task t_i {
7     for each available service s_ij
8     such that the global cost
9     constraint is still met with it {
10     tempGainWeight=(double)(previousService
11     For t_i.getRunTime()
12     −s_ij.getRunTime())/(double)(1000
13     +s_ij.getRunCost()
14     −prevServiceFor t_i.getRunCost());
15     if ((bestGainWeight<tempGainWeight) &&
16      (previousServiceFor t_i.getRunTime()
17       >s_ij.getRunTime())) {
18       bestGainWeight=tempGainWeight;
19       bestTask=i;
20       bestServiceIndex=j;
21      }
22     }
23    }
24    if (bestTask!=−1) {
25     assign s_(bestTask)(bestServiceIndex) to t_bestTask;
26    }
27   } while (bestTask!=−1)
```

Note that this version uses a local evaluation based on the execution times and costs of the given service and the service previously assigned to the task corresponding to the GAIN algorithm proposed by Sakellariou *et al*. (2007). As such, it is an estimated measure of improvement. Another version was also tested in which tempGainWeight is evaluated as shown in line 3 in Algorithm 2 and corresponds to GAIN2 proposed by Sakellariou *et al*. (2007). While this version potentially results in a more accurate evaluation, it takes much more time to evaluate the whole workflow compared to the local evaluation of GAIN, especially for a large number of services, and finally yields worse results because of that.

**Algorithm 2** Workflow evaluation in GAIN2

```
1   newWorkflowEval=EvaluateWorkflow(with s_ij
2     instead of the previous service for t_i);
3   (...)
4   tempGainWeight=(double)
5     (prevWorkflowEval.getRunTime()
6     −newWorkflowEval.getRunTime())/(double)(1000
7     +newWorkflowEval.getRunCost()
8     −prevWorkflowEval.getRunCost());
```

## 6 Modeling, scheduling, and execution of workflow applications in a BeesyCluster testbed

The algorithms discussed in Section 5 were implemented, deployed, and tested in a workflow management environment developed previously (Czarnul, 2013a). The environment uses BeesyCluster as a middleware that allows users to publish services and define parameters such as cost and execution time. These can be used for workflow definition, optimization, and workflow execution.

BeesyCluster can be regarded as a JEE-based front-end and middleware that allows publishing and consuming services offered by various providers and consumers from locations managed by them. The system (Czarnul, 2013b) was deployed at the Academic Computer Center in Gdansk, Poland for using high performance clusters and at the Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology for using a cluster, servers, and laboratories. It is used for research and teaching high performance computing.

Users access BeesyCluster through WWW or web services while BeesyCluster accesses distributed resources via SSH (Fig. 5). Each BeesyCluster user sets up an account in BeesyCluster and associates with it; i.e., several user accounts on the distributed clusters or servers are registered in the middleware. The middleware allows you to perform any operations on those accounts as if they were performed locally, including copying, editing files, management of directories and archives in a version management system, compilation, and running text or graphical applications in a browser. In the case of clusters, BeesyCluster hides details of queuing sequential or parallel applications using a portable batch system (PBS), load sharing facility (LSF), etc.

Note that the user can publish an application from his/her user account as a service within seconds (Czarnul, 2006) and can grant access to selected BeesyCluster users for a given fee. Each user has a virtual wallet that can be used for buying non-free services. Such services can be assigned to workflow tasks and selected for workflow execution.

The workflow management environment contains several modules that allow (Czarnul, 2010; 2013a; 2013b):

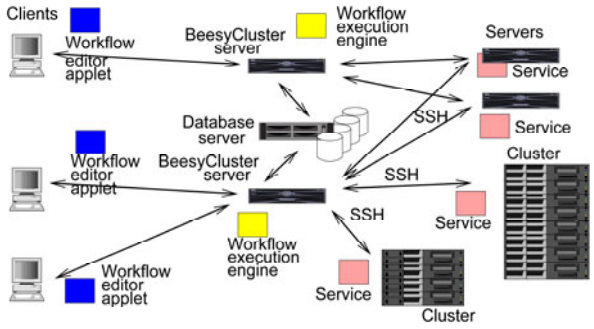


**Fig. 5 BeesyCluster architecture (extended from Czarnul (2013a))**

1. Definition of a workflow in an editor implemented as a Java applet that allows modeling workflow applications as shown in Figs. 6 and 12. The editor uses the workflow model described in Section 4 and allows assignment of BeesyCluster services to workflow tasks. The user can specify input data to the workflow as a set of files from any location available to them through the BeesyCluster layer.

2. Selection and scheduling services using the aforementioned algorithms to optimize the goal.

3. Actual workflow execution in the BeesyCluster environment.

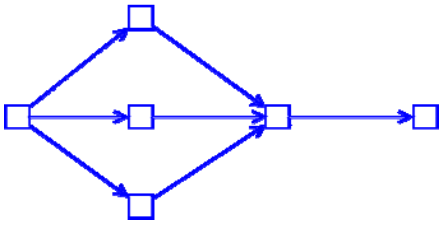

**Fig. 6 Workflow application with 6 nodes, 30 services**

The workflow management environment in BeesyCluster was described in detail by Czarnul (2013a). However, ILP was analyzed by Czarnul (2013a) for a small workflow with only 6 nodes. This paper contains a comparison of several algorithms for several workflows of considerably larger sizes. A similar model was considered by Czarnul (2010) but additionally it determines data flows among services. Namely, given the input data the algorithm is also able to determine values of $d_i$ but does so using just one algorithm. Various cost bounds were considered by Czarnul (2010) rather than deeper analysis of service failures as in this work.

Existing workflow management systems for grid environments were characterized by Yu and Buyya (2005). Yu and Buyya (2005) provided a comparison of several systems, including: Gridbus, Kepler (Ludäscher *et al*., 2006), Pegasus (Deelman *et al*., 2004), Triana (Majithia *et al*., 2004), directed acyclic graph manager (DAGMan), ICENI, GridFlow, GrADS, Askalon, UNICORE, Taverna, and GridAnt. These grid-oriented systems use mainly middlewares such as Globus Toolkit, Grid Application Toolkit, or other resource management systems for running jobs.

BeesyCluster acts as a middleware for the workflow management system described in this work. Compared to the aforementioned middlewares, BeesyCluster differs as it accesses services published by users through user accounts via SSH. Globus Toolkit uses security certificates based on asymmetric cryptography (public and private keys). BeesyCluster stores security credentials in its own database (in encrypted fields) and requires that BeesyCluster users add these credentials to the BeesyCluster database. Thus, the management layer lies entirely within the BeesyCluster server. Regarding the startup times for spawning tasks, BeesyCluster maintains a pool of open secure sessions with the registered servers or clusters, which allows fetching one of the already open and available sessions. A special mechanism for monitoring and respawning sessions works in the background.

Consequently, contrary to many other systems, adding new clusters or registering new user accounts can be accomplished in a matter of seconds as it requires only registration of the respective IPs and accepting keys for secure communication.

## 7 Experiments

The test methodology assumes that all algorithms are compared against each other using the same workflows.

Services can be offered by independent providers from their own locations (servers/clusters) on the terms (cost) imposed by them. We consider workflow

applications that allow processing data in parallel by parallel workflow paths with minimization of the workflow execution time and a bound on the cost. Workflow nodes model tasks that need to be accomplished and the algorithms are to select services capable of executing the tasks so that the goals considering execution time and costs are optimized. The on-the-fly selection is needed as there can be problems with accessing services (due to servers/clusters down times, network issues that are independent from the provider) or runtime changes in service parameters.

### 7.1 Methodology for modeling dynamic changes and invoking dynamic scheduling

For a given optimization algorithm A, BeesyCluster runs the algorithm before the workflow execution starts, assuming that all services assigned to the nodes are available. This corresponds to the usual static scheduling. Best services are chosen to optimize the goal as defined in Section 4. Then the execution of the workflow starts according to the workflow graph. Before running a service for the particular task, BeesyCluster checks if the service previously selected for the task is still available, or if conditions such as prices of services for the task have changed since the initial optimization, or if new services capable of executing the task have appeared. The probability that the service is not available, failed, new services for the task have appeared or the other conditions have changed is $p$. The range of $p$ tested varies from 0 to 0.6 from an environment with no changes during the workflow execution to a highly changeable one. If $p$=0, the system will require only the initial static scheduling. If $p$>0, BeesyCluster will apply dynamic scheduling by rerunning algorithm A considering that services for the already executed tasks have been chosen. These steps are depicted in Fig. 1. The system is designed in such a way that before a task is executed, a method is invoked that can check whether any of the aforementioned events have occurred. In the tests, the method considers service unavailability to trigger rescheduling. Whether it is service unavailability or change in service parameters is not important for the comparison of the performance of various algorithms, as this is only a triggering mechanism and is the same for all the algorithms tested. Obviously, various adjustment strategies for price or other QoS metrics might be used by providers, but this does not affect the comparison of the algorithms. Such strategies could be implemented as either changing the service parameters before the task is executed or in the background according to the model.

The particular value of $p$ may depend on particular types of services and type of market from which the workflow uses services. Note that high values of $p$ may result not only from unreliable services but also from highly changeable service parameters. For example, many Internet shops adjust their prices on a daily basis and this would qualify for changed service parameters. If the service is available and the other conditions have not changed, the service is executed by BeesyCluster. For each configuration and service availability, results are averaged from multiple runs for the given workflow. At least 20–30 runs are used for an average with the observation that for $p$=0 five runs turned out to be sufficient to provide reliable results. This is further discussed in Section 7.5.

The execution time includes both the running time of algorithm A (may be rerun several times in one workflow run if several services are not available or failed) as well as the execution time of actual services. Results allow us to investigate the performance of the proposed approaches, in particular if and in which cases global knowledge is needed and which returns better results than approaches with only local data. There is a trade-off between additional quality gained by using global data and a more refined algorithm and the execution time of such an algorithm which needs to be added to the total workflow execution time.

### 7.2 Testbed environment

All tests of the proposed algorithms were performed in the real BeesyCluster system (https://beesycluster.eti.pg.gda.pl/ek/AS_LogIn). As presented in Section 7.4, depending on particular configurations, for each task there are several services with different time/cost parameters which are deployed on real servers registered in BeesyCluster. The workflow is constructed in the BeesyCluster editor by assigning real services to particular tasks.

BeesyCluster uses services installed on nodes of a university cluster and laboratories at the Faculty of ETI, Gdansk University of Technology. The nodes that we currently use have the following configurations: 4 GB of RAM, two dual core Intel Xeon CPU @ 2.80 GHz with HT for a total of 8 logical cores running CentOS Linux version 2.6.32-220.13.1.el6.x86_64; 8 GB of RAM, Intel Core i7-2600K CPU @ 3.40 GHz for a total of 8 logical cores running SUSE Linux version 3.1.10-1.16-desktop; 12 GB of RAM, Intel Xeon CPU W3540 @ 2.93 GHz for a total of 8 logical cores running Debian 4.3.5-4 Linux version 2.6.32-5-amd64.

Note that the execution times $t_{ij}^{\text{exec}}$ of various services $s_{ij}$ are set to different values using various $t_{ij}^{1}$, e.g., $t_{ij}^{\text{exec}} = f_{ij}(d_{ij}^{\text{in}}) = t_{ij}^{1} d_{ij}^{\text{in}}$. Function $f_{ij}(\cdot)$ considers both the efficiency of the service code and the performance of node $N_{ij}$. Thus, the real environment services have different execution times and costs as outlined in Section 7.4, even if the services are installed on nodes with the same specifications. This is natural because services with shorter execution times per data unit $t_{ij}^{1}$ will usually be more expensive (requiring more effort to be faster), and vice versa. In the end, these are the relative differences in service execution times that make the scheduling problem difficult.

### 7.3 Testbed workflow applications

Tests were performed on several workflow structures that are typical of many scientific (Coutinho *et al*., 2010) or business (Czarnul, 2013a) workflows with parallelization of certain stages and synchronization, useful for a variety of real applications as suggested in Section 2. Following characterization of scientific workflows described by Juve *et al*. (2013), the workflow structures tested fall into the category of the epigenomics workflow, e.g., as shown in Fig. 6, or sequences of the epigenomics workflows executed in parallel as shown in Figs. 9, 12, 18, and 21. Additionally, these workflow structures correspond to workflow applications for processing of multimedia content such as application of successive filters on digital images described by Czarnul *et al*. (2012). In the business context, the structures such as the one shown in Fig. 6 may correspond to a production company subcontracting services or buying components from others and integrating these with their own know-how (Czarnul, 2013a).

Such workflow applications are aimed at parallel processing of data and assignment of such services so that the workflow execution time is minimized while keeping the cost of the selected services below the budget which is a very practical criterion. In case of scientific workflows, such as the aforementioned epigenomics workflow (Juve *et al*., 2013) and multimedia workflow (Czarnul *et al*., 2012), these are computation-intensive workflow applications. In the case of business workflow applications with several providers offering functionally equivalent services at various costs (Czarnul, 2013a), the execution time of services is also crucial for the total workflow execution time and workflow edges are used to control and synchronize particular tasks.

For each data file flowing through the workflow, a proper service is invoked. Data is processed in parallel by parallel workflow paths. To assess the relative performance of scheduling algorithms, for each task there are several services (between 3 and 10 for particular test cases as outlined below) that differ in execution time and monetary cost. Time and cost parameters for the services are assumed similar to the experiments performed by Czarnul (2013a) and Yu *et al*. (2008) and are shown in Tables 3 and 4. The services used in experiments are run for a particular time period for each input file at the given financial cost as indicated in Tables 3 and 4. To use faster services, higher financial costs are required, which is a real world dependency. In a dynamic environment where some of these services may become unavailable at runtime, we assess the algorithm performance considering the need for rescheduling as described in the methodology in Section 7.1. The same balanced

**Table 3 Service execution time $t_{ij}^{1}$ per data unit and cost of the workflow as shown in Figs. 6, 15, 18, and 21**

| Name | Cost (mEUR) | Time (s) |
|---|---|---|
| pd1 | 12 | 8 |
| pd3 | 10 | 9 |
| pd5 | 8 | 10 |
| pd7 | 7 | 11 |
| pd9 | 6 | 12 |

**Table 4 Service execution time $t_{ij}^1$ per data unit and cost of the workflow as shown in Figs. 9 and 12**

| Name | Cost (mEUR) | Time (s) |
|---|---|---|
| pd11 | 12 | 8 |
| pd12 | 12 | 8 |
| pd13 | 10 | 9 |
| pd14 | 10 | 9 |
| pd15 | 8 | 10 |
| pd16 | 8 | 10 |
| pd17 | 7 | 11 |
| pd18 | 7 | 11 |
| pd19 | 6 | 12 |
| pd20 | 6 | 12 |

structure allows us to evaluate relative performance of algorithms for various workflow sizes.

As a consequence, results obtained in this paper, especially suitability and comparison of various scheduling algorithms in a dynamically changing service availability, can also be used for these various workflow applications, in particular, the epigenomics workflow (Juve *et al*., 2013), multimedia workflow (Czarnul *et al*., 2012), and business workflow applications (Czarnul, 2013a).

The input files are split as uniformly as possible among parallel paths of the workflow. Thus, in the experiments $d_i \in \mathbb{Z}$, $d_i \geq 0$ corresponds to the number of data files processed by a service. The input files in the experiments are files that indicate input data and parameters for processing. Using workflows of a similar structure but varying sizes allows us to draw conclusions about the relative performance of the algorithms that might be affected by workflow sizes and not by varying workflow structures. Due to service failures and imposed budgets, selection of services for the workflow paths becomes a challenge.

Furthermore, the workflow sizes tested vary from 6 nodes and 5 services per node up to over 100 nodes and several hundred services per node and are typical of many scientific and business applications. Workflows of similar sizes were analyzed by Chin *et al*. (2010) for adaptive service scheduling for workflow applications to minimize the makespan of the workflow application. Services assigned to successive tasks are installed on various distinguished cluster nodes.

The sizes of text ILP models generated for the larger experiments considered in this work are as follows: 106 KB (40 nodes and 200 services), 180 KB (102 nodes and 418 services).

### 7.4 Results

Firstly, the workflow application shown in Fig. 6 is run with all the optimization algorithms. For each node there are 5 services available with parameters as shown in Table 3, and 18 input files are passed to the workflow. In the particular task, a service is invoked for each of the input files.

A bound equal to a half of the sum of the cheapest possible workflow and the cost of the fastest possible workflow is used, i.e., 648 in this case. Fig. 7 presents workflow execution times. The ILP algorithm shows the best results and is followed by divide-and-conquer, GAIN, and GA. The ILP with a timeout of 10 s returns optimal results much smaller than the time limit as shown in Fig. 8.

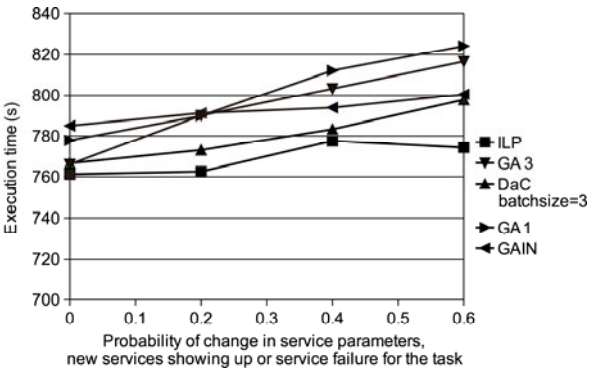


**Fig. 7 Execution time $t^{\text{workflow}}$ of workflow with 6 nodes, 30 services while $\sum d_{ij}^{\text{in}} c_{ij} < B$, $B$=648**

For the workflow with 10 nodes and 100 services (10 services per task) shown in Fig. 9 with parameters shown in Table 4, results are presented in Fig. 10. Corresponding execution times of the algorithms and the whole workflow applications are presented in Fig. 11. The cost bound of 972 is used, and 18 files are passed as input to the workflow. ILPHEU with a 10 s timeout returns the best results followed closely by GAIN and divide-and-conquer as well as the genetic versions with similar performance.

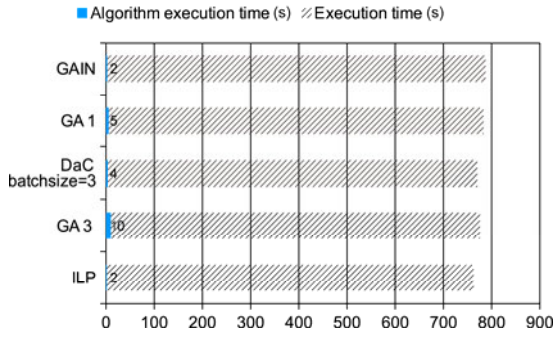


**Fig. 8 Algorithm execution time of workflow with 6 nodes, 30 services, $p$=0**

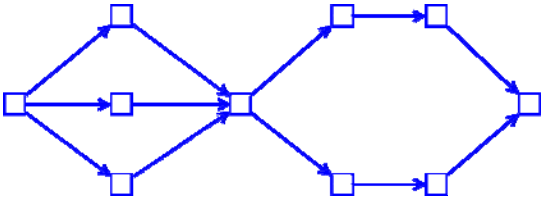

**Fig. 9 Workflow with 10 nodes, 100 services, and service parameters as given in Table 4**

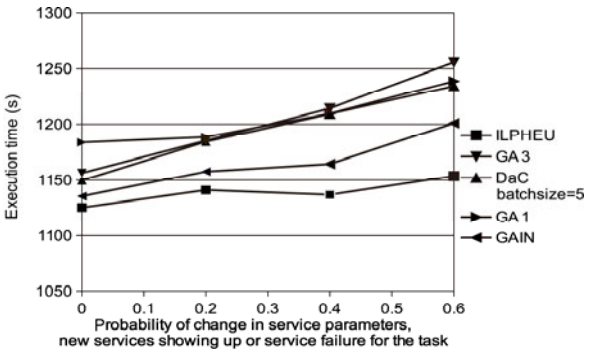


**Fig. 10 Execution time $t^{\text{workflow}}$ of workflow with 10 nodes, 100 services while $\sum d_{ij}^{\text{in}} c_{ij} < B$, $B$=972**

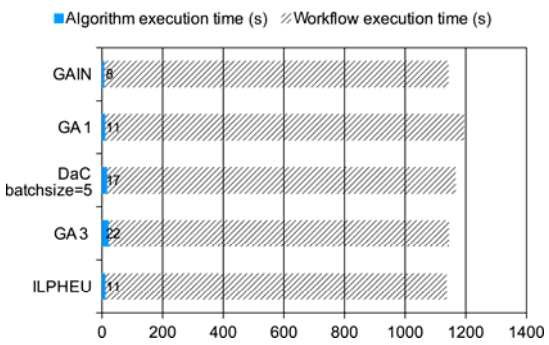


**Fig. 11 Algorithm execution time of workflow with 10 nodes, 100 services**

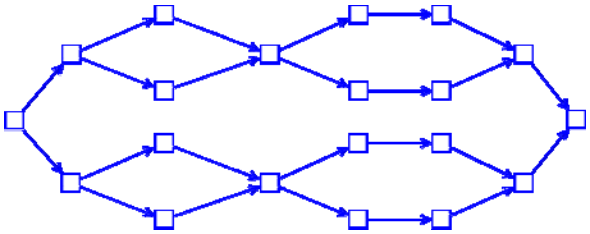

**Fig. 12 Workflow with 20 nodes, 200 services, and service parameters as given in Table 4**

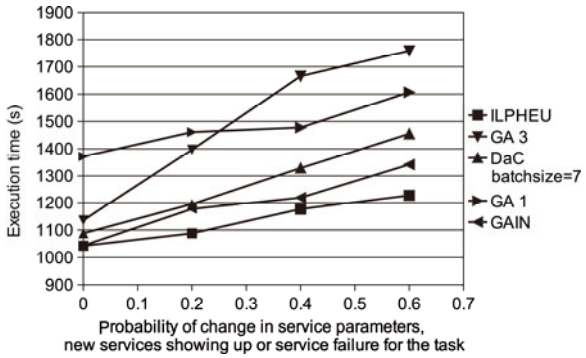


**Fig. 13 Execution time $t^{\text{workflow}}$ of workflow with 20 nodes, 200 services while $\sum d_{ij}^{\text{in}} c_{ij} < B$, $B$=1272**

For the workflow with 20 nodes shown in Fig. 12 with parameters shown in Table 4, we obtain the results presented in Figs. 13 and 14, in which 16 input files are passed to the workflow. ILPHEU with a 20 s timeout algorithm returns a suboptimal solution. The cost bound of 1272 is used. As shown in Fig. 13, the ILPHEU algorithm, even though it returns suboptimal solutions, is much better than the divide-and-conquer and GA solutions. GAIN is only slightly worse than ILP and also significantly better than DaC and GA.

Additional tests are performed for a less balanced workflow application of similar size with 21 tasks and 5 services per task. This workflow application shown in Fig. 15 was used previously for testing data streaming in workflow execution by Czarnul (2013b). Service parameters are shown in Table 3, in which 16 input files are passed to the workflow. The cost bound of 1172 is used. In this case, it turns out that for scheduling, results shown in Fig. 16 are similar to those of the balanced workflow applications of similar sizes; i.e., ILPHEU leads, followed by GAIN, divide-and-conquer, and GA. Corresponding execution times of algorithms for this case are presented in Fig. 17.

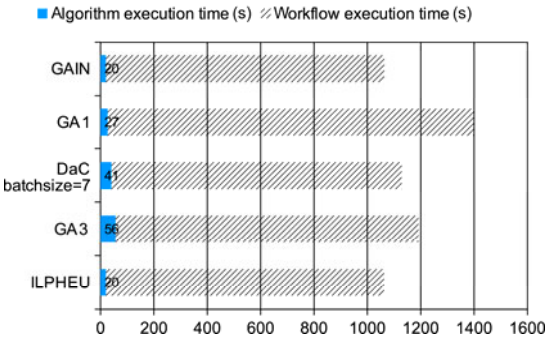


**Fig. 14 Algorithm execution time of workflow with 20 nodes, 200 services, $p$=0**

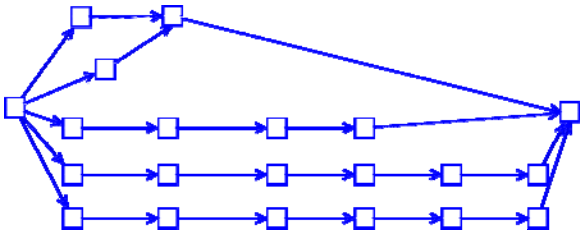

**Fig. 15 Workflow with 21 nodes, 105 services**
Service parameters are as given in Table 3; the workflow was previously used for testing data streaming in workflow execution by Czarnul (2013b)

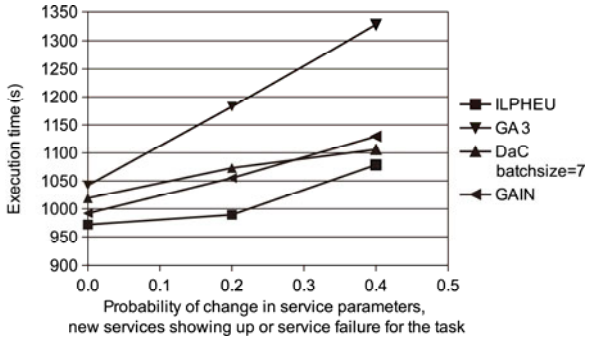


**Fig. 16 Execution time $t^{\text{workflow}}$ of workflow with 21 nodes, 105 services while $\sum d_{ij}^{\text{in}} c_{ij} < B$, $B$=1172**

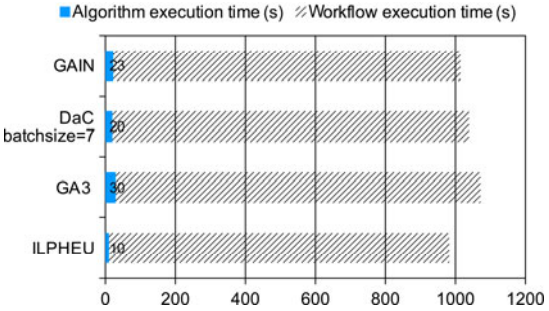


**Fig. 17 Algorithm execution time of workflow with 21 nodes, 105 services, $p$=0**

Increasing the size of the problem, a workflow application with 40 nodes, each with 5 services assigned to it, is considered (Fig. 18). Figs. 19 and 20 present corresponding results. The cost bound $B$=2048 is used, and 16 input files are passed to the workflow. Service parameters are as shown in Table 3. It can be seen that relative performances of the algorithms are similar to those for the workflow with 20 nodes. However, the differences between GAIN and the worse performing algorithms are smaller. This is due to the increasing execution time of GAIN.

Finally, a workflow application with 102 nodes, each with either 3 or 5 services assigned to it, for a total of 418 services, is considered (Fig. 21). Figs. 22 and 23 present the results for this large scenario, in which 16 input files are passed to the workflow. Service parameters are as shown in Table 3 for nodes with 5 services assigned. Every second task in the graph has 3 services with parameters for services pd1, pd5, and pd9. The cost bound $B$=2526 is used. In this case, the relative performance of GAIN is worse than that of DaC but still better than that of the GA approach. This is due to the fact that for a large workflow with a large number of services and a large cost bound compared to the sum of the cheapest services, GAIN requires many substitutions of services to finally approach the cost bound. This impacts the workflow execution time.

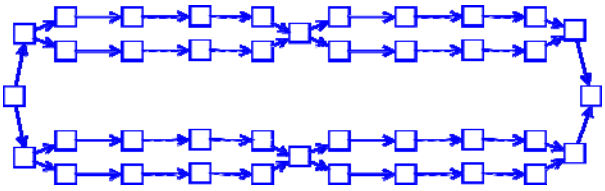

**Fig. 18 Workflow with 40 nodes, 200 services, and service parameters as given in Table 3**

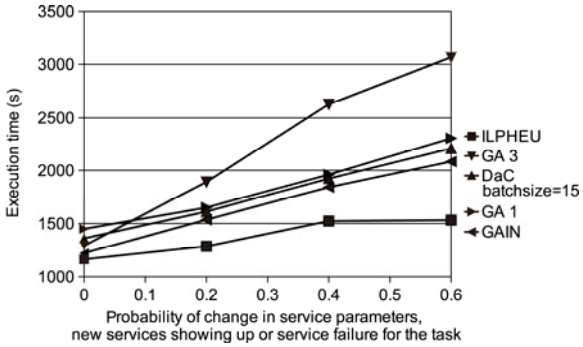


**Fig. 19 Execution time $t^{\text{workflow}}$ of workflow with 40 nodes, 200 services while $\sum d_{ij}^{\text{in}} c_{ij} < B$, $B$=2048**

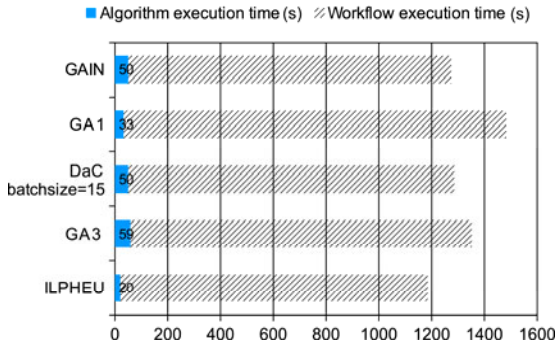


**Fig. 20 Algorithm execution time of workflow with 40 nodes, 200 services, $p$=0**

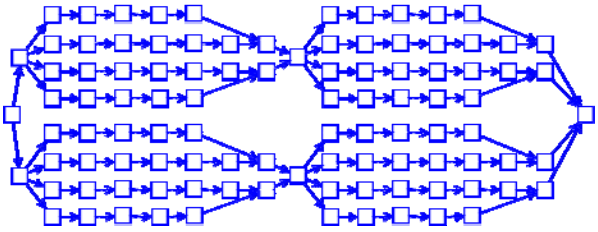

**Fig. 21 Workflow with 102 nodes, 418 services**

Service parameters are as given in Table 3 for nodes with 5 services assigned. Every second task in the graph has 3 services with parameters as for services pd1, pd5, and pd9

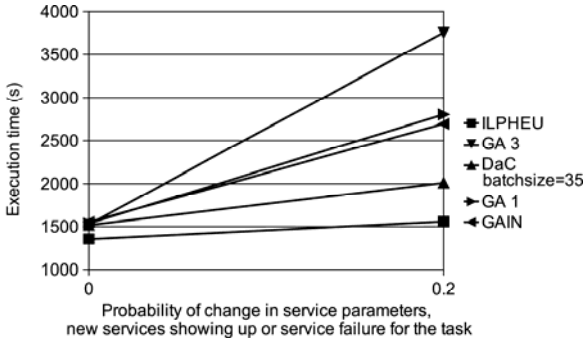


**Fig. 22 Execution time $t^{workflow}$ of workflow with 102 nodes, 418 services while $\sum d_{ij}^{in} c_{ij} < B$, $B$=2526**

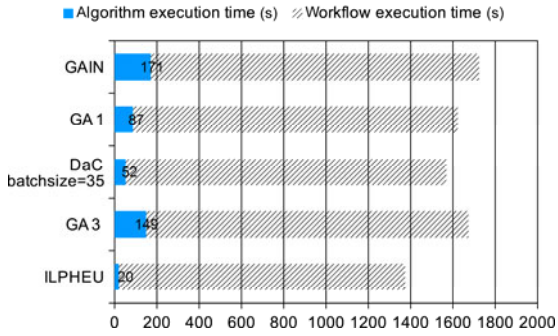


**Fig. 23 Algorithm execution time of workflow with 102 nodes, 418 services, $p$=0**

To investigate the relative performance of the solutions, Fig. 24 is prepared which shows results for the workflow application described above with 20 nodes and a probability of service failure or change in service parameters equal to 0.6. The relationship between the workflow execution times and ability to approach the cost bound can be concluded.

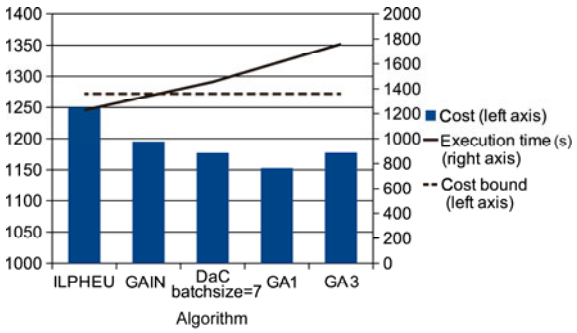


**Fig. 24 Execution times and costs for algorithms with 20 nodes, 200 services**

Given similar algorithm execution times with the exception of GA3, the algorithms able to approach the cost bound better are able to use faster services and minimize the workflow execution time better. It can be seen that GA3 is able to approach the cost bound better than GA1 due to a larger number of iterations. However, the total execution time of the algorithm does not allow us to take advantage of this when considering the workflow execution time.

### 7.5 Discussion

To better understand the algorithms, Table 5 presents conclusions on the execution times of the algorithms tested with notes related to workflow sizes and the factors on which the times are dependent. Additionally, Fig. 25 presents the standard deviation of the workflow execution times for various probabilities of change in service parameters, new services showing up, or failures of previously selected services. As can be seen in Fig. 25, the standard deviation of the workflow execution times for all the algorithms except GA for $p$=0 is a few seconds, which proves that the presented averaged results are repeatable numbers. It is interesting to see that GA shows the largest deviation, which is expected due to the randomness of this algorithm when generating new populations. The graph allows us to assess what deviation can be expected for the particular configuration.

**Table 5 Conclusions on the execution times of the algorithms**

| Algorithm | Execution time |
|---|---|
| ILP/ ILPHEU | For small problems (only the configuration with 6 nodes in the experiments), returns optimal results; for all other configurations, basically returns so far the best solutions within the smallest imposed time limits (20 s for larger configurations) |
| GAIN | Short for small workflows (6 nodes in the experiments), average for mid-sized workflows (20 nodes), and long for large workflows with a large number of nodes, a larger number of services, and a cost bound considerably above the minimum cost solution; in the latter case many substitutions of services are required to arrive at the cost bound and return a solution; the execution time depends highly on all these parameters |
| DaC | Generally longer than ILP/ILPHEU although smaller than ILP/ILPHEU times $n_t/b_s$ |
| GA | Generally the longest compared to other algorithms; requires recalculations of workflow execution times for many chromosomes; highly dependent on the number of chromosomes in the population, number of iterations, and also the workflow size—the number of nodes, not that much on the number of services per node or the cost bound—as opposed to GAIN |

Furthermore, the tendency, i.e., the growth of the standard deviation versus the probability of service failure, change in parameters, or occurrence of new services, is shown. Note that there are very slight differences in workflow execution times even for no changes during workflow execution. This is due to small variations in several runs for each probability for which execution times are averaged.

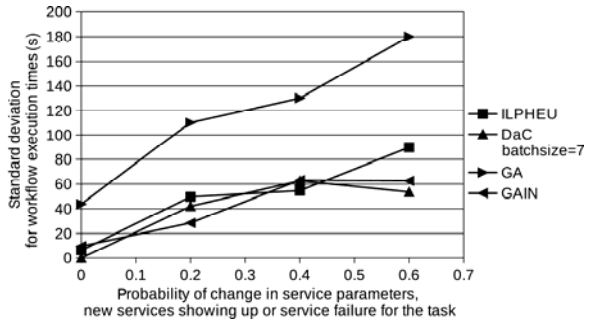


**Fig. 25 Standard deviation of the workflow execution times with 20 nodes, 200 services**

The execution times of the algorithms tested in this study are expected to grow with an increasing number of services per task. Namely, for ILP/ILPHEU the number of equations, i.e., Eqs. (8) and (9), will grow with the number of services, similarly for the divide-and-conquer approach where the ILP model is used for subdomains. GA may take more time to converge to a good solution. For GAIN, the service substitution phase will also depend on the number of services. Thus, it is recommended to limit the number of services to a few representatives per task, e.g., 5 services. This can be achieved as follows: For the given task, services with the smallest and the largest $tc_{ij}=t_{ij}/c_{ij}$ can be selected for $tc_{ij}^{min}$ and $tc_{ij}^{max}$, respectively. Then, the services selected for scheduling should be services for which $tc_{ij}$ is closest to the following values: $tc_{ij}^{min}$, $tc_{ij}^{min}+a$, $tc_{ij}^{min}+2a$, $tc_{ij}^{min}+3a$, $tc_{ij}^{min}+4a$, where $a=(tc_{ij}^{max}-tc_{ij}^{min})/4$. In this way, the selected services represent the wide spectrum of the execution time or cost of the actual services for the task and allow us to use the results obtained in this study even if there are more available services.

### 7.6 Conclusions

To summarize the above simulations, we can conclude the following, considering static scheduling for $p$=0 and dynamic scheduling with rerunning the algorithm if necessary for $p$>0:

1. For small problems, the algorithms from best to worst are: ILP, DaC, GAIN, GA. For the smallest workflow with 6 nodes, ILP returns the best results as it is able to find an optimal solution quickly; no other algorithm can match its results.

2. For mid-sized problems up to 40 workflow nodes, the algorithms from best to worst are: ILPHEU, GAIN, DaC, GA. ILPHEU returns the best results followed closely by GAIN, then DaC and GA. It is interesting to see that the global knowledge gives ILPHEU an edge over the GAIN algorithm in spite of similar execution times for 20 nodes.

3. For the large workflow with over 100 nodes and several hundred services, the algorithms from best to worst are: ILPHEU, DaC, GAIN, GA. As mentioned above, this is due to the large number of substitutions GAIN needs to perform. Advantages and disadvantages of the algorithms are presented in Table 6.

**Table 6 Advantages and disadvantages of the algorithms**

| Algorithm | Advantage | Disadvantage |
|---|---|---|
| ILP/ILPHEU | Able to solve optimally and quickly small problems (ILP) of around 6 nodes; returns better solutions compared to other algorithms (ILPHEU) for the sizes and structures of workflows tested up to 100 nodes and several hundred services | May be unable to return a solution if the time limit is too small or the problem is sufficiently large |
| GAIN | Returns good solutions in reasonable time, always returns a solution (if possible at all) since the algorithm starts from the cheapest services, with the exception of ILPHEU better than the others for mid-sized workflows with 10–40 nodes | Execution time heavily depends on the number of services and difference between the sum of the cheapest services and the imposed cost bound; slower for larger workflows |
| DaC | A reasonably good solution if subproblems are solved successfully with the partitioned subcosts, especially for larger workflows for which ILPHEU might fail | Solutions to subproblems may be infeasible when cost is divided among subproblems and cheaper services may fail |
| GA | No need for any configuration, parameter tuning if starting with the cheapest services; possible to define any constraints and optimization goals; constraints and optimization goal can be non-linear | Reasonably slow |

4. DaC returns results worse than GAIN except for larger workflows where it is better. It is generally slightly better than GA. Sometimes, the algorithm is unable to solve a subproblem for the subcost assigned to it. In this case, tightening the cost bound for initial batches can be applied or all unsolved subproblems can be solved globally using ILPHEU, which increases the execution time.

5. GA is reasonably slow although for $p$=0, the performance of GA is very close to those of ILP/ILPHEU. This means that the quality of the solution is good. For $p$>0 the total execution time from rerunning the algorithm is just too large compared to the others. For GA, the author's tests are performed with varying numbers of iterations, GA1 (smaller) and GA3 (greater), to see if a larger number of iterations could improve the results. It can be seen that for a small workflow with 6 nodes their performances are similar. For a workflow with 20 nodes and $p$=0, GA3 is significantly better, meaning that additional iterations do result in better quality. However, for $p$ higher than approximate 0.1–0.25 depending on the configuration, GA1 is better. In this case, GA1 is good enough to cope with service failures, unavailability, or changing parameters, but its execution time is shorter than that of GA3. This leads to the conclusion that it is better to engage a better algorithm in advance and react to the changing parameters and service unavailability with a faster algorithm.

These findings can be incorporated into Beesy-Cluster for use with both static scheduling before the workflow execution starts and dynamic scheduling for invoking a scheduling algorithm when a previously selected service is not available, new services have appeared, or parameters of services have changed considerably. If ILP/ILPHEU is able to return an optimal or suboptimal solution within an acceptable time frame such as 10 s for workflows with up to 6–10 nodes (additional tests show that 10 s is long enough to return suboptimal solutions for 10 nodes) or 20 s for larger workflows, then it is used. It can always be used for small workflows up to around 6 nodes where it returns optimal results. Otherwise, GAIN is used for mid-sized problems and DaC could be used for larger ones when ILPHEU fails due to the complexity/size of the workflow.

Still, the user can choose to use any of the implemented algorithms and other algorithms can be preferred. For instance, to specify a non-linear goal, GA can be used. Similarly, if one does not want to be forced into any kind of configuration, GA can be preferred.

## 8 Summary and future work

This paper presents a model for selection and scheduling of services for execution of workflow applications. Services capable of executing particular

tasks of the workflow are offered by various providers on various terms such as cost and execution time. The goal is to select and schedule services to minimize workflow execution time with a bound on the cost of selected services. The paper adopts integer linear programming to solve the problem and compares the results to genetic algorithms, fast local heuristic, GAIN, and divide-and-conquer. The comparison is done in a setting in which services may become unavailable, new services may appear, or conditions of existing services may change. In these cases, rescheduling must be performed, which prolongs the workflow execution time. The paper assesses the best algorithms for workflows of particular sizes suitable for scientific and business applications, namely ILP/ILPHEU followed closely by either GAIN or DaC depending on the size and GA. All the tests are run in a real workflow management environment implemented on top of the BeesyCluster middleware.

Future work will focus on performing thorough tests using workflow applications of various sizes for other structures.

## References


Abrishami, S., Naghibzadeh, M., Epema, D., 2010. Cost-driven scheduling of grid workflows using partial critical paths. 11th IEEE/ACM Int. Conf. on Grid Computing, p.81-88. [doi:10.1109/GRID.2010.5697955]

Aggarwal, R., Verma, K., Miller, J., *et al.*, 2004a. Constraint driven web service composition in METEOR-S. Proc. IEEE Int. Conf. on Services Computing, p.23-30.

Aggarwal, R., Verma, K., Miller, J., *et al.*, 2004b. Dynamic Web Service Composition in METEOR-S. Technical Report, LSDIS Lab, Univeristy of Georgia, Georgia, USA.

Bittencourt, L., Madeira, E., 2011. HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. *J. Internet Serv. Appl.*, **2**(3):207-227. [doi:10.1007/s13174-011-0032-0]

Blazewicz, J., Ecker, K., Schmidt, G., *et al.*, 1993. Scheduling in Computer and Manufacturing Systems. Springer Publishing Company. [doi:10.1007/978-3-662-00074-8]

Blythe, J., Jain, S., Deelman, E., *et al.*, 2005. Task scheduling strategies for workflow-based applications in grids. IEEE Int. Symp. on Cluster Computing and the Grid, p.759-767.

Braun, T., Siegel, H., Beck, N., *et al.*, 1999. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. Proc. Heterogeneous Computing Workshop, p.15-29.

Canfora, G., Penta, M., 2004. A lightweight approach for QoS-aware service composition. Proc. 2nd Int. Conf. on Service Oriented Computing, p.1-10.

Canfora, G., Penta, M., Esposito, R., *et al.*, 2005a. An approach for QoS-aware service composition based on genetic algorithms. Proc. Conf. on Genetic and Evolutionary Computation, p.1069-1075.

Canfora, G., Penta, M., Esposito, R., *et al.*, 2005b. QoS-aware replanning of composite web services. Proc. IEEE Int. Conf. on Web Services, **1**:121-129. [doi:10.1109/ICWS.2005.96]

Cardoso, J., Sheth, A., Miller, J., 2002. Workflow Quality of Service. Technical Report, LSDIS Lab, Computer Science, Univiersity of Georgia, Georgia, USA.

Chin, S., Suh, T., Yu, H., 2010. Adaptive service scheduling for workflow applications in service-oriented grid. *J. Supercomput.*, **52**(3):253-283. [doi:10.1007/s11227-009-0290-9]

Chirigati, F., Silva, V., Ogasawara, E., *et al.*, 2012. Evaluating parameter sweep workflows in high performance computing. Proc. 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, p.1-10. [doi:10.1145/2443416.2443418]

Coutinho, F., Ogasawara, E., de Oliveira, D., *et al.*, 2010. Data parallelism in bioinformatics workflows using Hydra. Proc. 19th ACM Int. Symp. on High Performance Distributed Computing, p.507-515. [doi:10.1145/1851476.1851550]

Czarnul, P., 2006. Integration of compute-intensive tasks into scientific workflows in BeesyCluster. Proc. 6th Int. Conf. on Computational Science, p.944-947. [doi:10.1007/11758532_127]

Czarnul, P., 2010. Modelling, optimization and execution of workflow applications with data distribution, service selection and budget constraints in BeesyCluster. Proc. Int. Multiconf. on Computer Science and Information Technology, p.629-636.

Czarnul, P., 2013a. Modeling, run-time optimization and execution of distributed workflow applications in the JEE-based BeesyCluster environment. *J Supercomput.*, **63**(1):46-71. [doi:10.1007/s11227-010-0499-7]

Czarnul, P., 2013b. A model, design, and implementation of an efficient multithreaded workflow execution engine with data streaming, caching, and storage constraints. *J Supercomput.*, **63**(3):919-945. [doi:10.1007/s11227-012-0837-z]

Czarnul, P., Dziubich, T., Krawczyk, H., 2012. Evaluation of multimedia applications in a cluster oriented environment. *Metrol. Meas. Syst.*, **19**(2):177-190. [doi:10.2478/v10178-012-0016-9]

Deelman, E., Blythe, J., Gil, Y., *et al.*, 2004. Pegasus: mapping scientific workflows onto the grid. Grid Computing, p.11-20. [doi:10.1007/978-3-540-28642-4_2]

Deelman, E., Singha, G., Sua, M., *et al.*, 2005. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, **13**(3):219-237.

Floudas, C., Lin, X., 2005. Mixed integer linear programming in process scheduling: modeling, algorithms, and applications. *Ann. Oper. Res.*, **139**(1):131-162. [doi:10.1007/s10479-005-3446-x]

Gao, A., Yang, D., Tang, S., *et al.*, 2005. Web service composition using integer programming-based models. IEEE Int. Conf. on e-Business Engineering, p.603-606. [doi:10.1109/ICEBE.2005.127]

Garg, S., Buyya, R., Siegel, H., 2010. Time and cost trade-off management for scheduling parallel applications on utility grids. *Fut. Gener. Comput. Syst.*, **26**(8):1344-1355. [doi:10.1016/j.future.2009.07.003]

Genez, T., Bittencourt, L., Madeira, E., 2012. Workflow scheduling for SaaS/PaaS cloud providers considering two SLA levels. IEEE Network Operations and Management Symp., p.906-912.

Geppert, A., Kradolfer, M., Tombros, D., 1998. Market-based workflow management. *In*: Trends in Distributed Systems for Electronic Commerce. Springer Berlin Heidelberg, p.179-191. [doi:10.1007/BFb0053410]

Juve, G., Chervenak, A., Deelman, E., *et al.*, 2013. Characterizing and profiling scientific workflows. *Fut. Gener. Comput. Syst.*, **29**(3):682-692. [doi:10.1016/j.future.2012.08.015]

Kyriazis, D., Tserpes, K., Menychtas, A., *et al.*, 2008. An innovative workflow mapping mechanism for grids in the frame of quality of service. *Fut. Gener. Comput. Syst.*, **24**(6):498-511. [doi:10.1016/j.future.2007.07.009]

Lin, C., Lu, S., 2011. Scheduling scientific workflows elastically for cloud computing. IEEE Int. Conf. on Cloud Computing, p.746-747. [doi:10.1109/CLOUD.2011.110]

Ludäscher, B., Altintas, I., Berkley, C., *et al.*, 2006. Scientific workflow management and the Kepler system. *Concurr. Comput. Pract. Exper.*, **18**(10):1039-1065. [doi:10.1002/cpe.994]

Majithia, S., Shields, M., Taylor, I., *et al.*, 2004. Triana: a graphical web service composition and execution toolkit. IEEE Int. Conf. on Web Services, p.514-521. [doi:10.1109/ICWS.2004.1314777]

Mika, M., Waligora, G., Weglarz, J., 2011. Modelling and solving grid resource allocation problem with network resources for workflow applications. *J. Schedul.*, **14**(3):291-306. [doi:10.1007/s10951-009-0158-0]

Patel, C., Supekar, K., Lee, Y., 2003. A QoS oriented framework for adaptive management of web service based workflows. Proc. 14th Int. Database and Expert Systems Applications Conf., p.826-835.

Rahman, M., Hassan, R., Ranjan, R., *et al.*, 2013. Adaptive workflow scheduling for dynamic grid and cloud computing environment. *Concurr. Comput. Pract. Exper.*, **25**(13):1816-1842. [doi:10.1002/cpe.3003]

Sakellariou, R., Zhao, H., Tsiakkouri, E., *et al.*, 2007. Scheduling workflows with budget constraints. *In*: Integrated Research in Grid Computing. Springer, p.189-202. [doi:10.1007/978-0-387-47658-2_14]

Stricker, C., Riboni, S., Kradolfer, M., *et al.*, 2000. Market-based workflow management for supply chains of services. Proc. 33rd Hawaii Int. Conf. on System Sciences, p.1-10. [doi:10.1109/HICSS.2000.926843]

Topcuoglu, H., Hariri, S., Wu, M., 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parall. Distr. Syst.*, **13**(3):260-274. [doi:10.1109/71.993206]

Varalakshmi, P., Ramaswamy, A., Balasubramanian, A., *et al.*, 2011. An optimal workflow based scheduling and resource allocation in cloud. *In*: Advances in Computing and Communications. Springer Berlin Heidelberg, p.411-420. [doi:10.1007/978-3-642-22709-7_41]

Wieczorek, M., Prodan, R., Fahringer, T., 2005. Scheduling of scientific workflows in the ASKALON grid environment. *ACM SIGMOD Rec.*, **34**(3):56-62. [doi:10.1145/1084805.1084816]

Wieczorek, M., Prodan, R., Fahringer, T., 2006. Comparison of workflow scheduling strategies on the Grid. Int. Conf. on Parallel Processing and Applied Mathematics, p.792-800. [doi:10.1007/11752578_95]

Wieczorek, M., Hoheisel, A., Prodan, R., 2009. Towards a general model of the multi-criteria workflow scheduling on the grid. *Fut. Gener. Comput. Syst.*, **25**(3):237-256. [doi:10.1016/j.future.2008.09.002]

Yao, Y., Liu, J., Ma, L., 2010. Efficient cost optimization for workflow scheduling on grids. Int. Conf. on Management and Service Science, p.1-4. [doi:10.1109/ICMSS.2010.5577645]

Yassa, S., Chelouah, R., Kadima, H., *et al.*, 2013. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *Sci. World J.*, **2013**:350934. [doi:10.1155/2013/350934]

Young, L., McGough, S., Newhouse, S., *et al.*, 2003. Scheduling architecture and algorithms within the ICENI grid middleware. UK e-Science All Hands Meeting, p.5-12.

Yu, J., Buyya, R., 2005. A taxonomy of workflow management systems for grid computing. *J. Grid Comput.*, **3**(3-4):171-200. [doi:10.1007/s10723-005-9010-8]

Yu, J., Buyya, R., 2006a. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. Workshop on Workflows in Support of Large-Scale Science, p.1-10.

Yu, J., Buyya, R., 2006b. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Sci. Program.*, **14**(3-4):217-230.

Yu, J., Buyya, R., Tham, C., 2005. Cost-based scheduling of scientific workflow applications on utility grids. Proc. 1st IEEE Int. Conf. on e-Science and Grid Computing, p.1-8. [doi:10.1109/E-SCIENCE.2005.26]

Yu, J., Buyya, R., Ramamohanarao, K., 2008. Workflow scheduling algorithms for grid computing. *In*: Metaheuristics for Scheduling in Distributed Computing Environments. Springer Berlin Heidelberg, p.173-214. [doi:10.1007/978-3-540-69277-5_7]

Yuan, Y., Li, X., Sun, C., 2007. Cost-effective heuristics for workflow scheduling in grid computing economy. Proc. 6th Int. Conf. on Grid and Cooperative Computing, p.322-329. [doi:10.1109/GCC.2007.57]

Zeng, L., Benatallah, B., Dumas, M., *et al.*, 2003. Quality driven web services composition. Proc. 12th Int. Conf. on World Wide Web, p.411-421.

Zeng, L., Benatallah, B., Ngu, A.H.H., *et al.*, 2004. QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, **30**(5):311-327. [doi:10.1109/TSE.2004.11]