J Supercomput (2013) 63:46–71
DOI 10.1007/s11227-010-0499-7

Modeling, run-time optimization and execution of distributed workflow applications in the JEE-based BeesyCluster environment

Pawel Czarnul

Published online: 13 November 2010

Abstract The paper presents a complete solution for modeling scientific and business workflow applications, static and just-in-time QoS selection of services and workflow execution in a real environment. The workflow application is modeled as an acyclic directed graph where nodes denote tasks and edges denote dependencies between the tasks. The BeesyCluster middleware is used to allow providers to publish services from sequential or parallel applications, from their servers or clusters. Optimization algorithms are proposed to select a capable service for each task so that a global criterion is optimized such as a product of workflow execution time and cost, a linear combination of those or minimization of the time with a cost constraint. The paper presents implementation details of the multithreaded workflow execution engine implemented in JEE. Several tests were performed for three different optimization goals for two business and scientific workflow applications. Finally, the overhead of the solution is presented.

Keywords Workflow execution · Just-in-time service selection · Workflow management environment · Workflow applications · Scientific and business workflows

1 Introduction

There is a need for integration of various software components for a variety of applications, in various fields. In scientific computing, highly dedicated codes written in languages such as C++/C/Fortran etc., either sequential or parallel, could be used together to achieve a complex task. Grid computing [28, 31, 33] allows coordinated

P. Czarnul (✉)
Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Narutowicza 11/12, 80-233 Gdansk, Poland
e-mail: pczarnul@eti.pg.gda.pl

resource sharing and integration of computational resources and building scientific workflows. In a business service oriented environment, providers expose their services using standards such as WSDL, UDDI, which allows others to invoke the services using SOAP. Integration into workflows with QoS service selection is usually performed [4, 5]. Services are also integrated with each other in context-aware applications in ubiquitous computing [15, 17].

Integration requires interoperability between services, optimization of QoS goals while possibly meeting other QoS constraints, as well as security. The paper presents a complete solution addressing all these issues with an ability to cope with runtime service failures and unavailability.

Section 2.1 presents current approaches and models for integration of services while Sect. 2.2 describes workflow management systems for scientific and business environments. Section 3.1 defines the workflow model used by the proposed workflow management environment in BeesyCluster which is presented in Sect. 3.2. Section 3.3 presents its editor, Sect. 3.4 the multithreaded workflow execution engine in JEE using JMS, Sect. 3.5 optimization algorithms implemented by the author, Sect. 3.6 workflow monitoring. Section 4 presents results for three different optimization goals for two applications that can be regarded as extendable templates useful for business and scientific purposes. Section 4.1 shows the overhead of the execution engine and finally Sect. 5 summarizes the work.

2 Related work

2.1 Approaches and models

The most common approach [4, 28, 31] proposes to model a workflow or a complex task as an acyclic directed graph $G(V, E)$ where nodes denotes simple tasks out of which the workflow is composed. Edges denote dependencies between particular tasks and are used to model a sequence of tasks, parallel execution of tasks or synchronization.

Traditional approaches associate each task with code that is mapped to one of several resources to be executed [3, 22]. In service oriented approaches it is assumed that each task has a set of alternative services associated with it, which are capable of executing the given task i.e. to perform the function defined for the task and generate needed output out of the given input data. Such services may be offered by their providers on various terms such as specified execution time and cost. The workflow is called abstract if there are more than one service assigned for at least one node. In this case, one service needs to be chosen for each node to perform the given task [4, 28, 33, 34]. Services must be chosen in such a way so that a certain criterion is minimized or maximized, possibly also meeting other constraints. As an example, the optimization goal can be minimization of a linear combination of the workflow execution time and the total cost of selected services or minimization of the workflow execution time with an upper bound on the total cost of selected services [33, 34]. The workflow is called concrete if there is one service associated with each task. Such a model is suitable for both scientific [31, 33] and business applications [4, 5]. Scheduling in utility grids [31, 33] or workflow scheduling in grids [28] consider selection of

one service for each task and scheduling service execution considering service execution times and costs. Business applications [4, 5] usually consider execution time, cost, availability [4, 20, 34, 35], accessibility [20], fidelity [7] or conformance [20], security [20], reputation [34].

Yu and Buyya [29] points out that scheduling can be global/local and centralized/distributed. An optimization algorithm may use either global knowledge about the workflow including all nodes and services or only local knowledge about a part of the workflow or even one node to select services. The former may be required to meet global constraints such as a constraint on the total cost of services selected for all tasks. Also, one centralized manager may make decisions or the latter may be taken in a distributed way.

Finally, service selection/scheduling may be performed *statically* before the execution of the workflow starts. However, in highly dynamic environments when new services appear and existing disappear and when conditions such as execution times, costs and others change [29], *just-in-time* decisions are necessary. In this case, a service is chosen for a particular task just before the task is to be executed. Şensoy and Yolum [23] presents an approach on how to select the best service selection mechanism to satisfy the client's requirements by observing results from various approaches and learning in a dynamic environment.

There also exist AI-based approaches [21] such as rule-based planning that distinguish rules allowing to determine if services may be connected. Output of one service must be compatible with input of another. The client can formulate a desired effect or goal that must be achieved and an algorithm, based on the knowledge about functions and effects, input and output of various services, is able to define intermediate goals and select services to achieve the desired goal.

Apart from the aforementioned workflow model, Petri nets and finite automatas are used to model workflow applications in ubiquitous computing [2]. In this case, a service can be invoked in a certain context which is assumed to be any information describing the situation of the given object. uWDL [15] specifies nodes, links along with the processing flow and context information.

2.2 Workflow management systems for distributed applications

Yu and Buyya [29] provides detailed characterization of existing workflow management systems for grid environments and compares Gridbus, Kepler [18], Pegasus [13], Triana [19], P-GRADE [16], Directed Acyclic Graph Manager (DAGMan), ICENI, GridFlow, GrADS, Askalon, UNICORE, Taverna, GridAnt. These systems use middlewares or service technologies such as Globus Toolkit (in Kepler, Askalon, Gridbus, Pegasus, ICENI, Triana), Unicore (in Unicore) or Web Services (in Triana, Taverna, Askalon, Kepler).

In service oriented computing, the basic standards and technologies used for Web Services include WSDL, SOAP and UDDI. The latter are complemented with standards such as RDF, OWL, OWL-S allowing definition of ontologies representing the domain of services. Namely, concepts that are used in service descriptions and queries for services are linked semantically in the ontology. This allows intelligent semantic search for services matching a textual user query in terms of IOPE (input,

output, precondition, effects) and using ontologies [24]. The IOPE of a service along with QoS parameters such as the execution time, cost, reliability, accessibility [4, 34] can be described in OWL-S and mapped to UDDI [24]. Meteor-S [1] allows this type of search along with selection of best services, composition and workflow execution. Business workflows are often defined in the BPEL (Business Process Execution Language for Web Services) format for which there exists several execution engines such as Silver [14] for mobile devices and desktop systems, bexee [25], The ActiveBPEL Engine [26].

For ubiquitous computing, FollowMe [17] is a platform that allows definition of workflows using CPDL (Compact Process Definition Language) including events triggering processes/activities. It allows running a workflow application on nodes managed by FollowMe.

In this paper, the author presents a model and an integrated environment for definition, run-time optimization and execution of scientific and business workflow applications when the availability of services is limited and changing in time. Before the workflow is run, services are chosen statically for workflow tasks based on the information available. Services are reselected just-in-time if some selected services become unavailable, new ones appear or the conditions have changed. Practical workflow applications for scientific and business uses are modeled and run in this environment developed by the author [9] in BeesyCluster.[1]

The contribution of this work is the workflow management environment that uses BeesyCluster, implemented using Java Enterprise Edition, as a middleware for publishing and running services. The motivation is to make use of many distinct features of the latter for service integration. BeesyCluster [9, 12] allows users to access their user accounts on distributed clusters or servers via WWW/Web Services and publish sequential or parallel applications, both scientific and business, as services [12] in a matter of seconds. The provider specifies the conditions including the price of a service and other BeesyCluster users who can invoke such services via WWW/Web Services or can later integrate them into own workflows. BeesyCluster allows very quick registration of new resources such as clusters, servers and workstations in the system as well as registration of user accounts to be used by BeesyCluster. This is different from many other workflow management systems which use middlewares or service technologies such as Globus Toolkit (in Kepler, Askalon, Gridbus, Pegasus, ICENI, Triana), Unicore (in Unicore) or Web Services (in Triana, Taverna, Askalon, Kepler) [29]. BeesyCluster only requires adding corresponding entries in the database storing IPs of accessed locations and saving corresponding security certificates. This is because ssh is used for establishing communication with the resources. The proposed environment allows integration of services, especially bridging business and scientific worlds by integration of two types of services. Secondly, the environment allows plugging in various optimization algorithms for both static and dynamic selection of services as well as partitioning of data for parallel computations [10] as well as intelligent searching for services for particular workflow tasks [11]. Integration into workflows, security, file transfer, payment for services are all provided by BeesyCluster.

[1] https://lab527.eti.pg.gda.pl:10030/ek/AS_LogIn.

3 A JEE-based workflow management environment

3.1 Model with just-in-time service selection

A workflow application is a complex task represented by an acyclic directed graph $G(V, E)$ where each vertex corresponds to a simple task and directed edges denote dependencies between the simple tasks. We assume that the user defines input files and the files that will be passed to particular initial tasks of the workflow before the workflow is executed. The model distinguishes the following:

1. t_i—denotes the ith task of the workflow.
2. d_i—denotes the size of input data passed to t_i, d_i is given.
3. s_{ij}—denotes the jth service available to execute task t_i. Only one service needs to be chosen to execute this task. Furthermore, each service has QoS parameters associated with it, in particular:
   - c_{ij}—the cost of performing the service.
   - t_{ij}—the time for performing task t_i by service s_{ij}.
   - P_{ij}—the provider of service s_{ij}.
   - N_{ij}—the node on which service s_{ij} runs while sp_n denotes the speed of node n.
   - d_{ij}—the size of data processed by service s_{ij}. Equal to d_i if the service is selected or 0 otherwise. d_{ijkl} denotes the size of data passed from s_{ij} to s_{kl}.
   - Possibly other QoS parameters such as the reliability of the service, conformance with established standards etc.

   It is assumed that each service is installed on a node e.g. a single computer, a cluster etc. The value of a particular parameter of such a node results from a reduction of values of the resources the service uses. For instance, the speed of a cluster on which a service is installed denotes the total speed considering all cluster nodes used by the service.
4. t_{ijkl}^{comm}—communication time of data of size d_{ijkl} sent from service s_{ij} to s_{kl}. This depends on the nodes N_{ij} and N_{kl} the services run on and is modeled as a sum of startup time and size of data divided by bandwidth [27]:

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

5. $t_i^{st} : i \in |V|$ is the time at which service s_{ij} chosen to execute t_i starts. Then $\forall_{i,k:(v_i,v_k)\in E}$

$$t_k^{st} \geq t_i^{st} + \sum_j t_{ij} + \sum_{j,l} t_{ijkl}^{comm} \tag{1}$$

   t_{ij} is larger than 0 only for one j as only one service is selected. Similarly, for the given i and k t_{ijkl}^{comm} is larger than 0 only for one pair of j and l since only one service per node i and one per node k are selected. On the other hand, there may be several tasks preceding task t_k.

We consider three alternative optimization criteria where $t^{workflow}$ ($\forall_{i,j}\ t^{workflow} \geq t_i^{st} + \sum_j t_{ij} + \sum_{j,l} t_{ijkl}^{comm}$) is the time when the last service finishes. The goal is to select such s_{ij} for every t_i so that one of the following is minimized:

1. $v_{\text{MIN\_TC\_PRODUCT}} = t^{workflow}(\sum d_{ij}c_{ij})$ (problem MIN_TC_PRODUCT)—product of workflow execution time and the sum of costs of selected services or
2. $v_{\text{MIN\_T\_C\_BOUND}} = t^{workflow}$—minimization of the workflow execution time with an additional constraint on the total cost of selected services i.e. $\sum d_{ij}c_{ij} < B$ where B is the budget (problem MIN_T_C_BOUND) or
3. $v_{\text{MIN\_TC\_SUM}} = \alpha t^{workflow} + \sum d_{ij}c_{ij}$—minimization of a linear combination of workflow execution time and the total cost of selected services (problem MIN_TC), $\alpha > 0$.

Assuming that services for each task are found and selected before the execution of the workflow starts, it may turn out that the particular service is not available at the time the given task is to be executed. Also, the service may fail. In these cases, another service for the task needs to be chosen just-in-time i.e. dynamically.

As an example, the workflow shown in Fig. 1 represents assembly of a product using components acquired in parallel followed by integration and subsequent parallel actions. This example can be regarded as a general template suitable for various business or scientific applications such as:

1. Task t_3 represents producing a jam out of various fruits (t_1 and t_2 correspond to purchasing fruits) or assembly of toys out of simple components or assembly of computer systems using individual components (t_1 and t_2 represent purchase of components), tasks t_4, t_5, t_6 represent subsequent distribution of final products to various markets.
2. t_1 and t_2 represent collection of data, t_3 integration and t_4, t_5, t_6 represent publishing the data in information portals.
3. t_1 and t_2 represent parallel processing of data on various HPC clusters, t_3 represents data integration and t_4, t_5, t_6 correspond to subsequent parallel processing.

For the assembly/distribution example, it is assumed that 13 units of component A (task t_1) and seven units of component B (task t_2) are bought. There are several

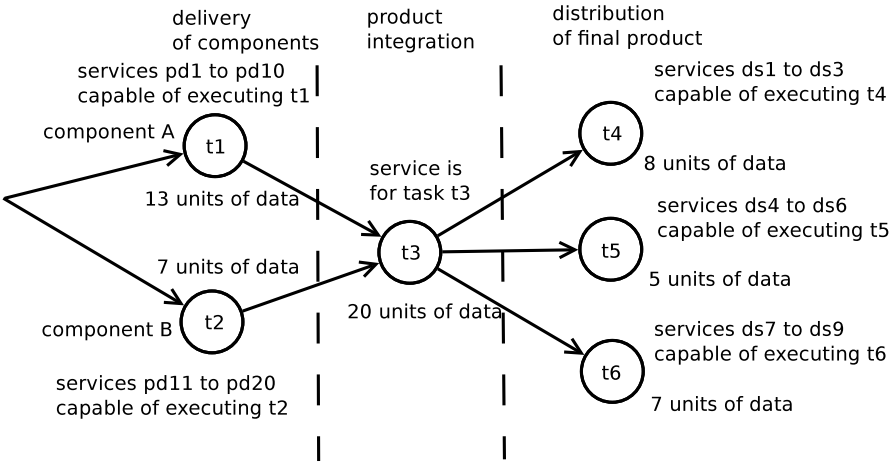


Fig. 1 Example of an assembly/distribution workflow

Table 1 Service data for assembly/distribution workflow

Service	Workflow node	Time per unit [s]	Cost per unit [mEUR]
pd1/pd11	1/2	8	12
pd2/pd12	1/2	8	12
pd3/pd13	1/2	9	10
pd4/pd14	1/2	9	10
pd5/pd15	1/2	10	8
pd6/pd16	1/2	10	8
pd7/pd17	1/2	11	7
pd8/pd18	1/2	11	7
pd9/pd19	1/2	12	6
pd10/pd20	1/2	12	6
is	3	8	32
ds1	3	3	12
ds2	3	5	6
ds3	3	7	4
ds4	4	3	12
ds5	4	5	6
ds6	4	7	4
ds7	5	3	12
ds8	5	5	6
ds9	5	7	4

providers offering each component on various terms such as delivery time t_{ij} and cost c_{ij} (services $pd1$ to $pd10$ for task t_1 and $pd11$ to $pd20$ for task t_2). As usual, lower delivery time results in a higher cost (Table 1).

Task t_3 corresponds to the integration of components into a final product. There is only one service is with the cost equal to the own costs and markups of the producer of the product i.e. the entity simulating the workflow.

Finally, tasks t_4, t_5 and t_6 correspond to distribution of the final product to three various markets. For each, a distributor needs to be chosen (out of $ds1$ to $ds3$ for task $t4$, $ds4$ to $ds6$ for task $t5$, $ds7$ to $ds9$ for task $t6$) for sending eight, five and seven units, respectively.

3.2 BeesyCluster's architecture and services

The author has implemented an environment for modeling and execution of workflow applications with just-in-time service selection based on the proposed model. The environment is based on the BeesyCluster middleware. The latter is a JEE-based front-end and middleware that allows publishing and consuming services offered by various providers and consumers from locations managed by the former. BeesyCluster [9, 12], designed and co-developed by the author, was deployed at Academic Computer Center in Gdansk, Poland on large IA64 processor clusters and on a cluster

and servers at Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology.

A general architecture of the system is presented in Fig. 2. The user sets up an account and registers possibly several system accounts on distributed clusters or servers. The system accesses these accounts via SSH which makes it very easy to add new servers or clusters to the system, either managed by individuals, companies or universities. BeesyCluster allows the registered user to access these accounts via WWW (Fig. 3) and Web Services [12]. It allows management of files such as copying, editing, management of directories and archives and running Unix applications interactively or using a graphical interface through a browser as if they were run locally. Computational parallel applications available on accounts on high performance clusters are additionally supported. BeesyCluster hides details of queuing systems

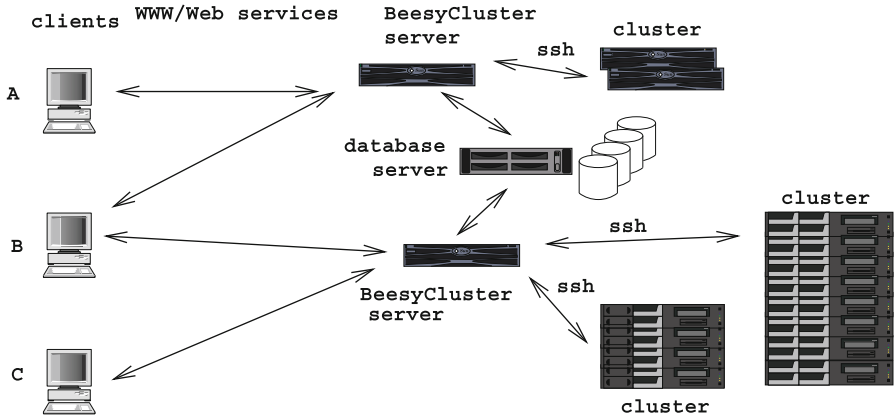


Fig. 2 BeesyCluster architecture

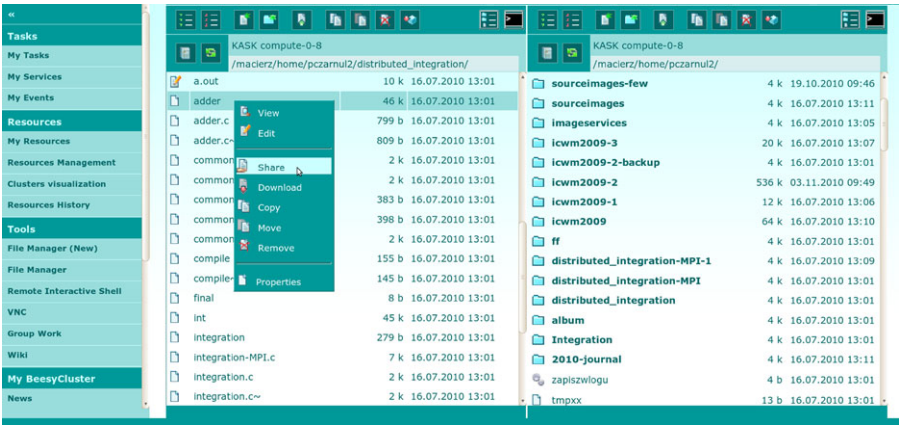


Fig. 3 Publishing a service in BeesyCluster

such as PBS, LSF, LoadLeveler by choosing a queue automatically based on the requested number of processors. It also allows monitoring statuses and browsing results of tasks.

Furthermore, each user who has access to any application installed on his/her account, either sequential or parallel, can publish it as a service in the BeesyCluster middleware (Fig. 3) and grant access to other BeesyCluster users on specified terms. The provider can define the cost of the service, periods of time when the service is available and others. Each user has a virtual wallet which can be used to buy services published by others. When a client runs a service offered by a provider, BeesyCluster provides a sandbox to run the service securely on the provider's account. Since, in general, users can publish either business services (such as ordering of goods, printing photos, translation etc.) as well as computational ones (such as parallel processing of data like FFT, matrix multiplication etc.), it makes BeesyCluster a platform allowing integration of businesses and science.

Furthermore, the workflow management environment developed by the author within BeesyCluster extends previous works [9] and allows the following:

- Incorporation of such services into workflows as described by the model.
- Management of queuing systems. If the cluster on which the parallel application represented by a service executes uses a queuing system (e.g. PBS, LSF) it will be used and results fetched transparently to the user.
- Support for a complete workflow creation and execution cycle i.e.:
  - workflow editing using a GUI.
  - workflow optimization (service selection and data distribution).
  - workflow execution in a distributed environment based on the schedule and distribution.

3.3 Workflow modeling applet

The author has developed a workflow editor in BeesyCluster as a Java applet shown in Fig. 4 implementing the model proposed in Sect. 3.1. The applet communicates with BeesyCluster's server using secure sockets. It consists of a panel on which the user can draw the workflow graph. The user is presented with a list of services available, either own or made available by other BeesyCluster users. One of BeesyCluster modules allows automatic registration of applications installed from Linux packages such as rpm or deb as BeesyCluster services [11]. In this case, categories and descriptions embedded in the packages are used to describe the service automatically. An intelligent search engine allows searching for services capable of performing the given function. Keywords from the query are searched for in service description, taking into account WordNet synonyms. This allows automatic and fast creation of a large service database out of existing applications and using an efficient search mechanism. The user, knowing the requirements of a particular task i.e. what function it should perform, what data it has to take and produce, can use the search mechanism to obtain services whose descriptions contain concepts best matching the concepts in the task requirements [11]. Out of the services presented, the user can narrow the list and assign services capable to perform the given task to the node representing the latter. For each service, apart from the defined cost and execution time, the author

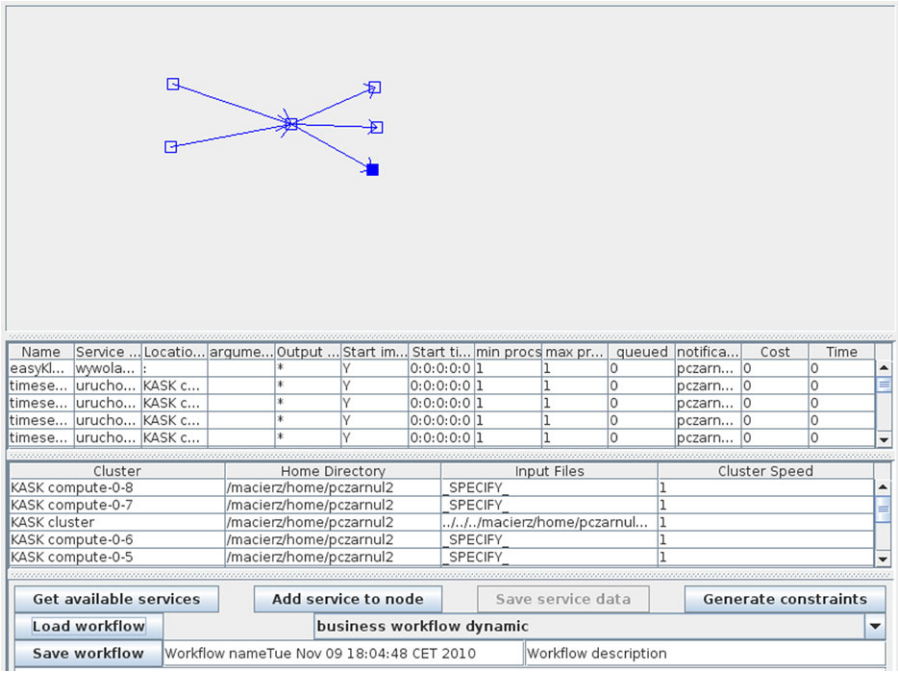


Fig. 4 Assembly/distribution workflow in BeesyCluster's editor

of a workflow can define the number of processors to be used, email for notification about status and errors, extensions of output files which will be transferred to following tasks or whether the service should be executed for each input file individually or once. In the latter case, the service will read all input files from its dedicated directory. After editing of the workflow has been finalized, the user must indicate paths to the input data located on any servers or clusters available to him/her. The system then generates a description of the model which is sent by the applet to BeesyCluster's server and saved in a database. The implementation assumes that data are stored in chunks in separate files and that data sizes from the model, i.e. d_i, correspond to the number of files processed by the given task and d_{ijkl} corresponds to the number of files copied between tasks t_i and t_k.

The user only specifies a workflow application and can invoke execution of the workflow. If new services appear, conditions such as costs have changed, or any of the services becomes unavailable or fails at runtime, the execution engine will automatically reselect a new service so that the specified criterion is optimized. This is very useful for both:

Scientific workflows—sometimes a cluster or a cluster node becomes unavailable due to maintenance, network or hardware failure. It is also possible that due to programming errors, the application published as a computational service fails. The execution engine will repeat computations using an alternative service, again considering optimization criteria. In BeesyCluster, each service may be a sequential or a parallel application to be run using the queuing system on the cluster it has been

installed on. The BeesyCluster middleware hides details of queuing the application. Furthermore, many scientific workflows consider the problem of finding resources to run the workflow tasks rather than choosing services to execute the tasks [3, 22]. The proposed model addresses this issue. If there are multiple services with same executables installed on various clusters or servers of various speeds, and accessible at various prices, choosing a service is equivalent to finding the best resource to run the task on.

Business workflow—in a dynamically changing market with intense competition, new businesses as well as offers constantly appear and change. In such an environment, it is very likely that the set of services available for a task will differ from the one available before the workflow execution starts. As above, this requires just-in-time service selection considering available offers. Businesses can request registration of their own servers in BeesyCluster from which, upon acceptance, will be able to publish services allowing ordering goods they produce such as TV sets, fruits etc. or services they offer like consulting, translation etc. Registration requires only adding the server's name and IP address to the database and can be done in seconds. Contrary to computational services on dedicated clusters, no queuing system will be used in this case and the service is invoked immediately.

3.4 Multithreaded workflow execution engine

The workflow is stored in several tables in the BeesyCluster database and processed using Enterprise Java Beans (EJBs) in the JEE-based BeesyCluster system (Fig. 5). One object of `SIWorkflowNodeBean` represents a particular service used in the workflow and contains in particular:

`workflowId`—identifier of the workflow whose node has this service associated with.
`nodeId`—the node the service is assigned to.
`serviceIndex`—an index of the service in the set of services assigned to a particular workflow node.
`runTime`—execution time of the service for one data unit.
`runCost`—cost of running the service for one data unit.
`clusterId`—identifier of the cluster/server where the service is installed on.
`filePath`—the application/command published as this service.
`queued`—indicates whether the application is queued or run immediately.
`outputFiles`—defines (possibly using wild cards) files that will be copied to following services.
`min/maxProcs`—number of processors to be used by the application.
`startDay/Hour`—when to run the application, immediately if not specified.

BeesyCluster offers a management panel implemented by `ServiceIntegrationServlet` where all available workflows are listed and can be run from (Fig 6). When the particular workflow is selected for the run, the servlet does the following (Fig. 7):

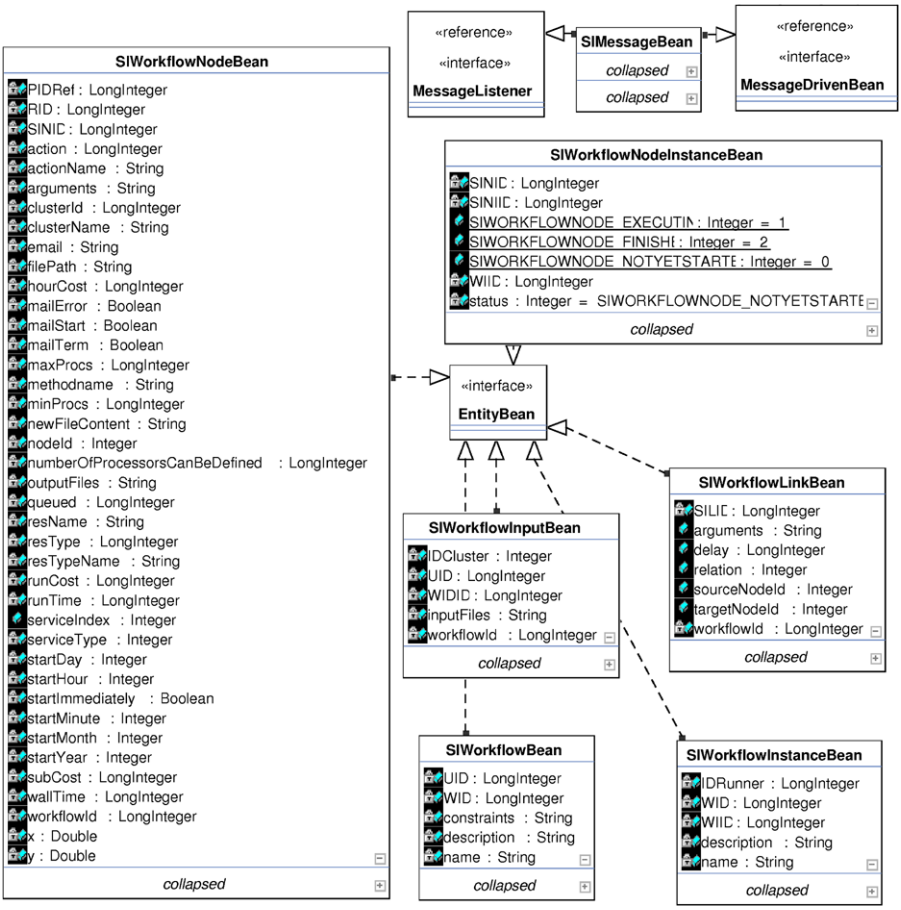


Fig. 5 Class diagram of core workflow classes

1. Creates an `SIWorkflowInstance` object which contains data for the particular instance of the workflow along with a name, description and the user running it.
2. Creates and uses a `WorkflowMappingSolver` to find an initial assignment of services to workflow nodes. The solver uses a file with workflow constraints. As described in Sect. 3.5, BeesyCluster supports several algorithms for service selection for the criteria mentioned in Sect. 3.1, either optimal or heuristic, using global or local knowledge about the workflow. For instance, an algorithm with a global knowledge may be run before the workflow starts and a fast heuristic algorithm may be run for just-in-time reselection when new services appear or existing services are not available.
3. Creates `SIWorkflowNode` objects for all workflow nodes out of the data stored in the database. In particular, `ServiceIntegrationServlet` creates dedicated directories on the nodes where the available services are located. This step is performed in parallel by a pool of threads.

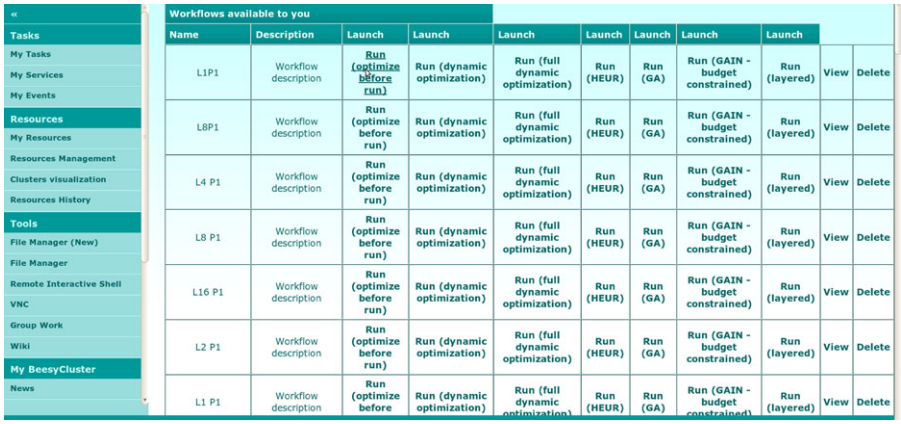


Fig. 6 Workflow panel in BeesyCluster

4. Finds nodes corresponding to the initial tasks as well as workflow input data from `SIWorkflowInput`.
5. For each initial task, it checks whether the previously selected service is available; if not, a new alternative service is found using one of available algorithms.
6. Several `DataCopier` threads are spawned for fast and parallel copying of input files to the locations of selected services.
7. After the copying threads synchronize using join, an JMS message is sent for each following task. Each message starts execution of the `onMessage()` method of `SIMessageBean` JMS bean which is responsible for execution of each service.

`SIMessageBean` bean executes the service as follows (Fig. 8):

1. Executes the given service using the input files already in the dedicated directory on the server/cluster the service is installed on. The service may be invoked for each individual input file or once in which case the service may read and process all the input files at once. This depends on the flag set for the node in the editor. If the `queued` value is non-zero then BeesyCluster will choose a queue on the cluster based on the number of processors requested for the service and submit a request to the queuing system e.g. PBS or LSF. Otherwise, the application associated with the service is run immediately. Communication with the cluster or server is done using the `jsch` library.
2. Each `SIMessageBean` thread increments a counter of already executed tasks preceding any of its successors by calling a synchronized method on the shared `WorkflowMappingSolver` object. This way each thread is able to determine if it has just executed the last predecessor of the given successor task [9]. If this is the case, BeesyCluster checks availability of the service associated with the following task. If not available, one of service selection algorithms is run to reselect the service.
3. The thread reads output files from the service using `outputFiles` of `SIWorkflowBean` and copies to locations associated with following services. To minimize the copying time, if there are many small files, these are packed into an

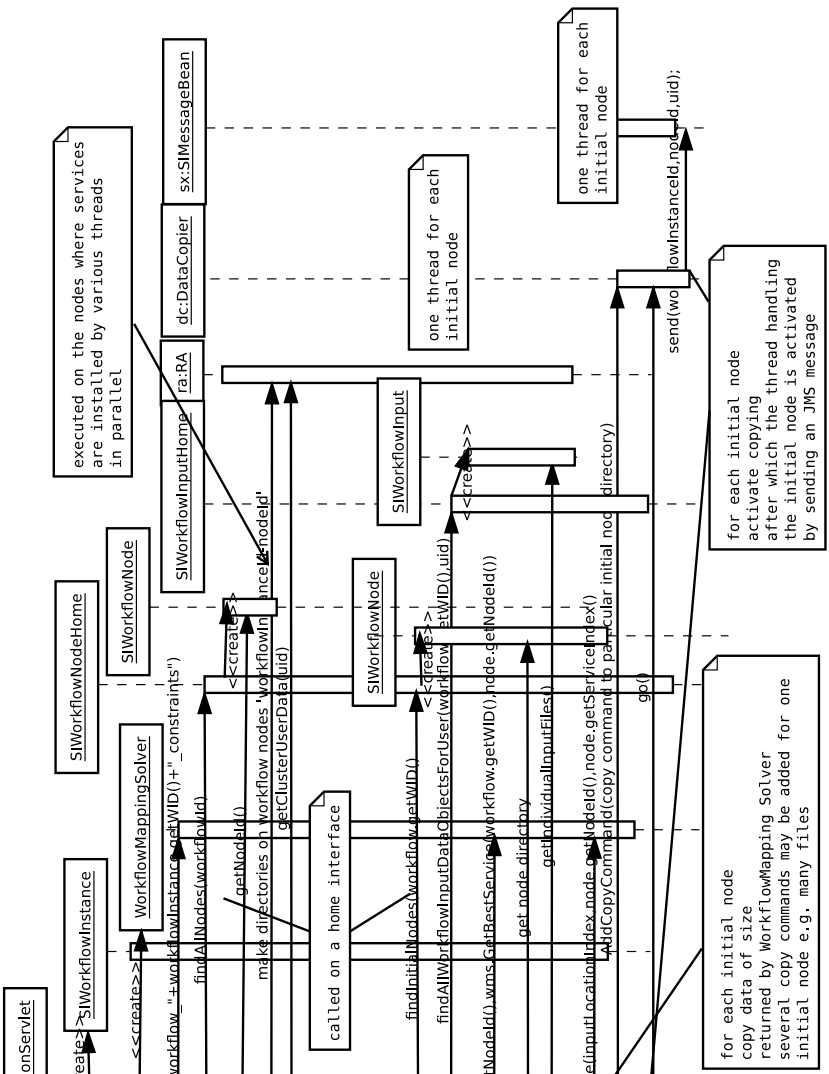


Fig. 7 Interaction diagram: startup of the workflow

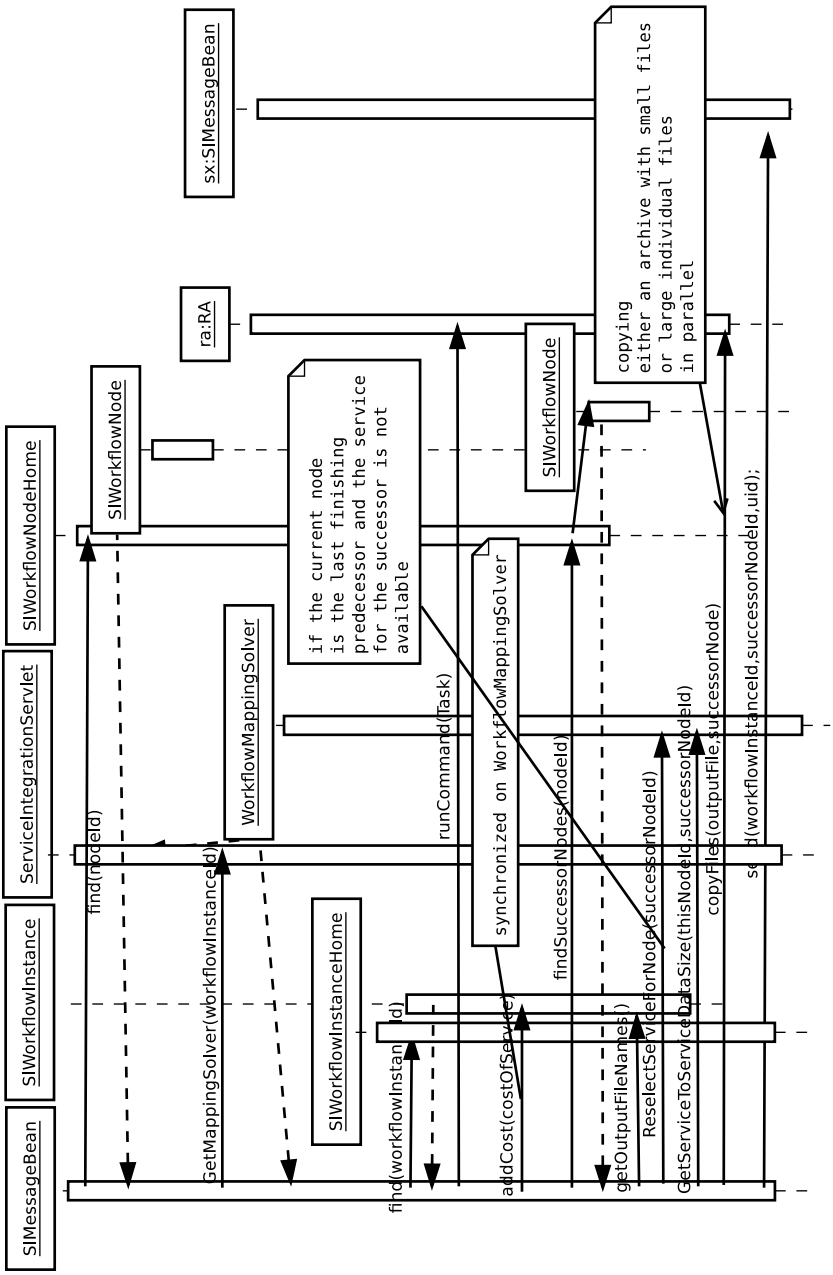


Fig. 8 Interaction diagram: execution of a service for workflow node

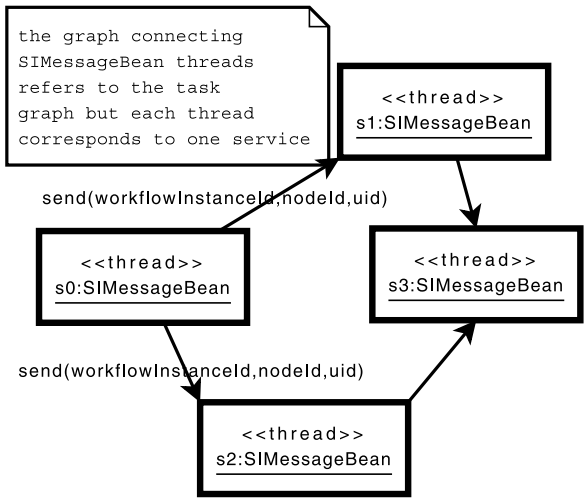


Fig. 9 Communication diagram: communication between workflow nodes

archive that is copied and unpacked on the destination node. Otherwise, large files are copied individually as packing would introduce additional overhead in this case. If the service fails later, a new alternative must be found and data copied again.

4. For each following nodes: if the thread has just executed the last predecessor task, a new JMS message is sent along with parameters for invoking the given successor (Fig. 9) This allows the join construct apart from the sequence and parallel execution of tasks by parallel threads and services.

The workflow execution engine uses the BeesyCluster middleware to check for dynamic changes of conditions:

- Services becoming unavailable. When a service for the given node is to be executed, the execution engine calls the service through BeesyCluster which uses `jsch` to run it via SSH on a remote node. The service may become unavailable in the following situations:
  - No connection possible with the node. BeesyCluster returns that the server cannot be reached or connection has been refused. There is a separate thread that checks for availability of nodes registered in BeesyCluster.
  - The provider has blocked access to the service. The execution module selects a new service on a different node and launches it.
  - The service was invoked but the connection has been interrupted. Since the service is supposed to write output files to its designated directory on the server, the execution module can either poll for appearance of specific output files (can be defined for each service in the workflow editor) or select a new service on a different node.
  - The service was invoked but has not terminated in the expected time frame (which is the service execution time with a certain delay added). The execution module selects a new service on a different node and launches it.

– Services changing parameters. As in other service oriented environments, a provider can change parameters of their services at any time, for instance increase the cost of a service. A separate thread checks if any of the parameters have changed since the workflow was defined or if any new services have appeared that might be better suited for the given task.

Currently, the user specifies their requirements when the workflow is defined.

3.5 Optimization module with Pluggable service selection algorithms

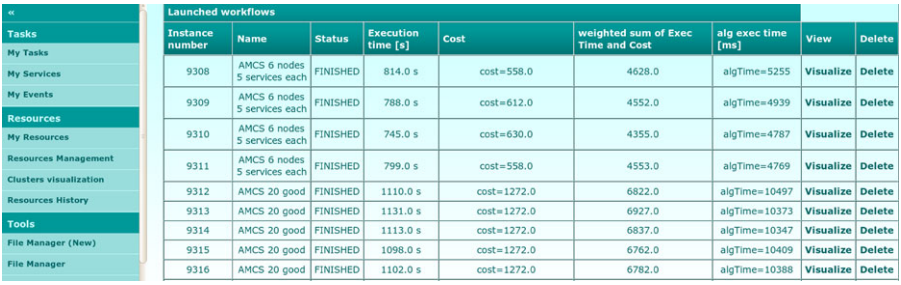
The design of the workflow management module allows easy addition of a new optimization algorithm that is invoked in `WorkflowMappingSolver`. Currently, BeesyCluster uses the following implemented by the author:

1. A fast heuristic algorithm FASTHEU which selects a service based only on the services available for the given task. It works as follows. For goal $v_{\text{MIN\_TC\_PRODUCT}}$, for execution of a particular task t_i, out of all services available for this task it selects the service with the lowest value of $t_{ij}c_{ij}$. For goal $v_{\text{MIN\_TC\_SUM}}$, for task t_i the service with the lowest $\alpha t_{ij} + c_{ij}$ is selected. Naturally, the algorithm is not suited for solving $v_{\text{MIN\_T\_C\_BOUND}}$ as it does not have a global knowledge to always meet the cost constraint.
2. A Mixed Integer Linear Programming (MILP) method MILP for goals $v_{\text{MIN\_TC\_SUM}}$ and $v_{\text{MIN\_T\_C\_BOUND}}$ where integer variables denote which service is selected for a particular task and the optimization goal and constraints contain only linear combination of these variables. The solution is optimal but may take time for large graphs.
3. A genetic algorithm where each chromosome represents assignments of one service to each task [10]. This algorithm is able to solve any of the assumed goals.
4. Layered algorithm which divides the graph into layers and solves each layer using the MILP algorithm. For large graphs, it is able to solve subdomains optimally and reasonably quickly. In some cases, may not meet the cost requirements for $v_{\text{MIN\_T\_C\_BOUND}}$.

It is possible to invoke a more complex and possibly more time-consuming algorithm before the workflow execution starts and then use a faster but heuristic algorithm for just-in-time selection after services for the particular task have changed. However, for goal $v_{\text{MIN\_T\_C\_BOUND}}$ an algorithm with global knowledge is needed to meet the cost constraint.

3.6 Monitoring workflow execution

BeesyCluster offers a panel (Fig. 10) for management of workflow instances currently running or recently completed for the given user. For each workflow it shows the status, total cost of selected services, workflow execution time and time spent by the optimization algorithm. Each workflow and workflow instance can be visualized and deleted from the database.



Instance number	Name	Status	Execution time [s]	Cost	weighted sum of Exec Time and Cost	alg exec time [ms]	View	Delete
9308	AMCS 6 nodes 5 services each	FINISHED	814.0 s	cost=558.0	4628.0	algTime=5255	Visualize	Delete
9309	AMCS 6 nodes 5 services each	FINISHED	788.0 s	cost=612.0	4552.0	algTime=4939	Visualize	Delete
9310	AMCS 6 nodes 5 services each	FINISHED	745.0 s	cost=630.0	4355.0	algTime=4787	Visualize	Delete
9311	AMCS 6 nodes 5 services each	FINISHED	799.0 s	cost=558.0	4553.0	algTime=4769	Visualize	Delete
9312	AMCS 20 good	FINISHED	1110.0 s	cost=1272.0	6822.0	algTime=10497	Visualize	Delete
9313	AMCS 20 good	FINISHED	1131.0 s	cost=1272.0	6927.0	algTime=10373	Visualize	Delete
9314	AMCS 20 good	FINISHED	1113.0 s	cost=1272.0	6837.0	algTime=10347	Visualize	Delete
9315	AMCS 20 good	FINISHED	1098.0 s	cost=1272.0	6762.0	algTime=10409	Visualize	Delete
9316	AMCS 20 good	FINISHED	1102.0 s	cost=1272.0	6782.0	algTime=10388	Visualize	Delete

Fig. 10 Workflow instance panel in BeesyCluster

4 Simulations

The author has created two workflow applications in BeesyCluster which were subsequently executed in a real environment for the three aforementioned optimization goals. The applications show the suitability of the approach for business and scientific uses in a dynamic environment. The applications can be regarded as practical templates which allow substitution of the services used with particular services required for the practical application, as explained for the assembly/distribution workflow in Sect. 3.1.

For each run, static optimization was performed first using the given algorithm considering that all known services are available. The author has then considered various probabilities which indicate availability of the services—for each task 20%, 40%, 60%, 80% or 100% of known services are available with an assumption that at least one service for the task is present. If the service chosen before for the task is not available, then the given algorithm reselects a new service considering the optimization criterion. In case of an algorithm that uses global knowledge the services already used for the executed tasks are also considered.

The simulations used the BeesyCluster instance available at https://lab527.eti.pg.gda.pl:10030/ek/AS_LogIn through a user account who has access to several servers and a cluster located Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology and at Academic Computer Center of GUT with IA-64 and IA-32 clusters (http://www.task.gda.pl/kdm/index.html) including the 288-processor `holk`. The environment is used for research and teaching of high performance computing.

The following two tests were performed:

Business workflow: The assembly/distribution workflow presented in Sect. 3.1 was tested in the aforementioned BeesyCluster environment. For each task, services with parameters as shown previously in Table 1 were published on various nodes in the university network. This situation resembles well a service oriented environment where independent providers publish services performing certain tasks with possibly various execution times and prices. BeesyCluster allows providers to manage such services derived from applications as well as parameters and adjust them at any time. Such services are used in the workflow composed by a client. If a previously chosen service becomes unavailable or the terms of the services that could be used for a task have changed, just-in-time reselection is invoked. For each distinct

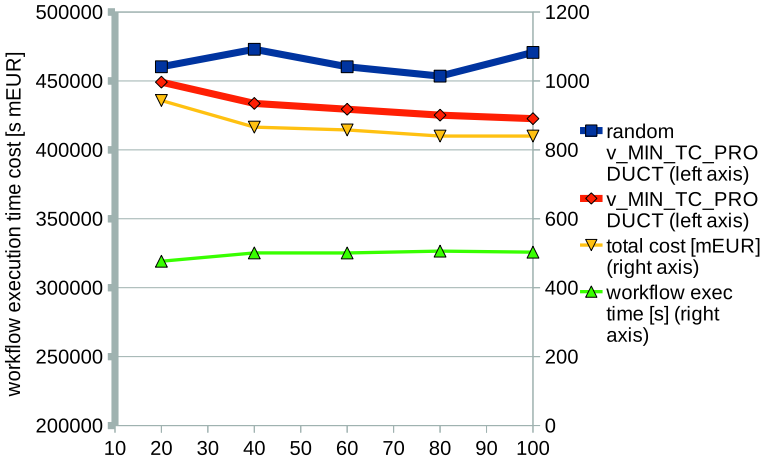


Fig. 11 Average $t^{workflow}(\sum d_{ij}c_{ij})$ vs service availability for assembly/distribution workflow, various costs for services

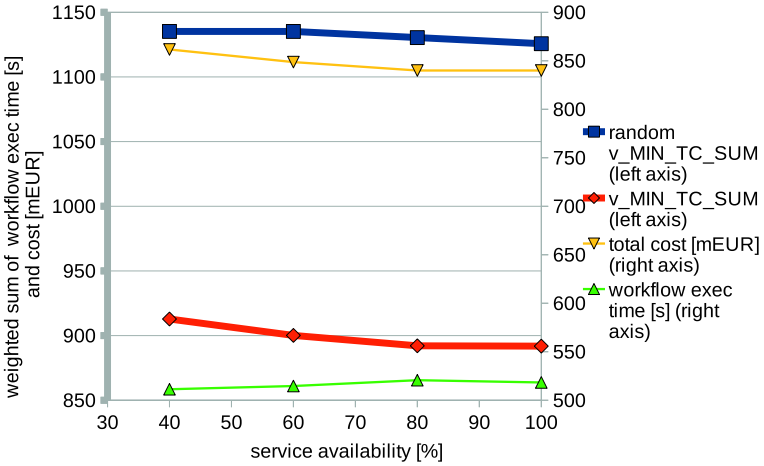


Fig. 12 Average $\alpha t^{workflow} + \sum d_{ij}c_{ij}$ with $\alpha = 0.1$ vs service availability for assembly/distribution workflow, various costs for services

optimization criterion and each probability of service availability, several tests were performed and averaged. The following criteria were optimized:

$v_{\text{MIN\_TC\_PRODUCT}}$—for the minimization of a product of the workflow execution time and the total cost of selected services, the FASTHEU algorithm described in Sect. 3.5 was used. If a service needs to be reselected because the previously chosen one is unavailable or conditions for other services have changed, the service with the lowest $t_{ij}c_{ij}$ value out of those available at the moment is selected. Figure 11 shows that the average time-cost is lower for better service availability i.e. closer to 100%. It can be seen that the algorithm selects services with higher

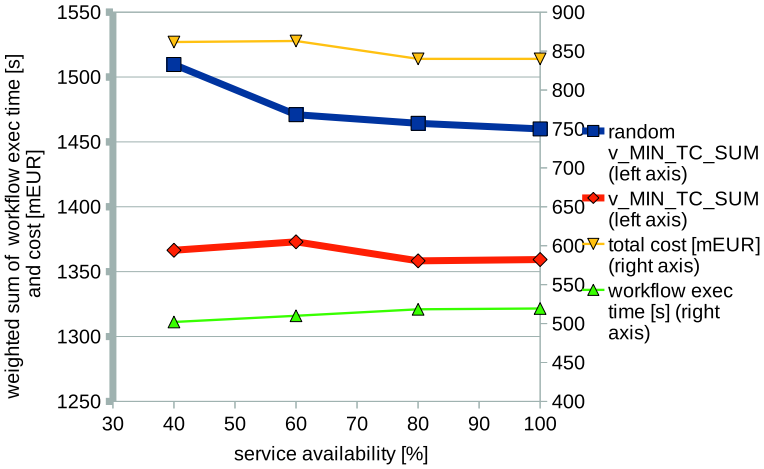


Fig. 13 Average $\alpha t^{workflow} + \sum d_{ij} c_{ij}$ with $\alpha = 1$ vs service availability for assembly/distribution workflow, various costs for services

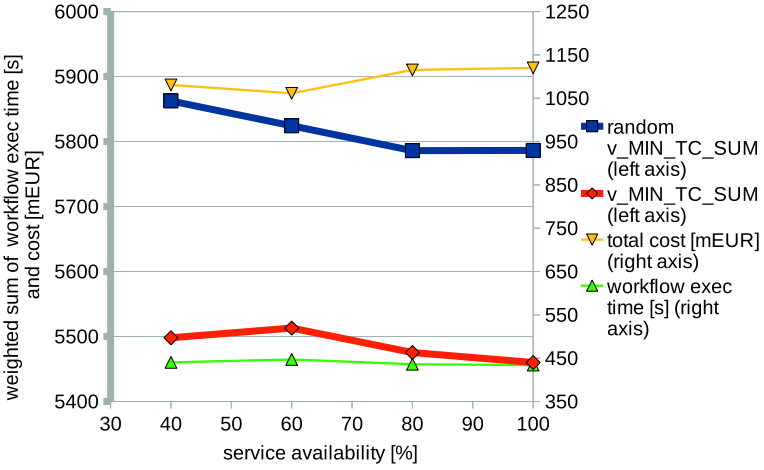


Fig. 14 Average $\alpha t^{workflow} + \sum d_{ij} c_{ij}$ with $\alpha = 10$ vs service availability for assembly/distribution workflow, various costs for services

execution times but lower costs. $t_{ij}c_{ij}$ is lower for such services than for those with lower execution times and higher costs as shown in Table 1. This algorithm may yield suboptimal results which are, however, considerably better than random selection as shown in Fig. 11.

$\upsilon_{\text{MIN\_TC\_SUM}}$—as for $\upsilon_{\text{MIN\_TC\_PRODUCT}}$, the FASTHEU algorithm was used which, if needed, selects the service with the lowest $\alpha t_{ij} + c_{ij}$. Figures 12, 13 and 14 present results obtained for $\alpha = 0.1$, $\alpha = 1$, $\alpha = 10$, respectively. In all cases, the results are considerably better than the random approach. For $\alpha = 0.1$ and $\alpha = 1$, the better the availability of the services, the lower the average total cost

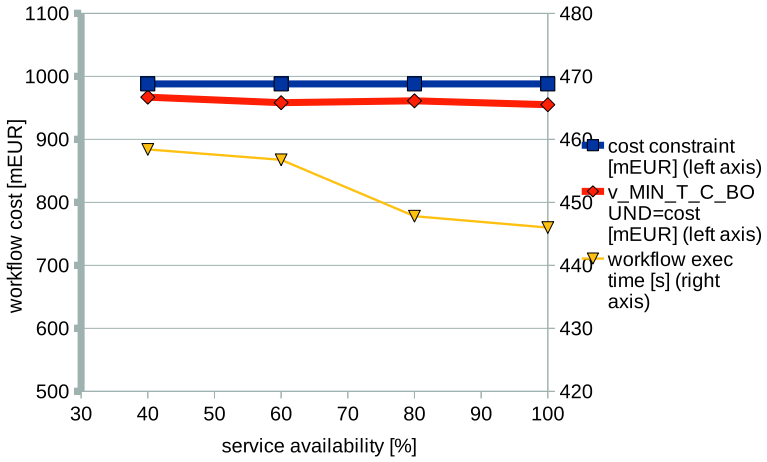


Fig. 15 Average $t^{workflow}$; $\sum d_{ij}c_{ij} < B$, $B = \frac{costofcheapestworkflow+costoffastestworkflow}{2}$ vs service availability for assembly/distribution workflow, various costs for services

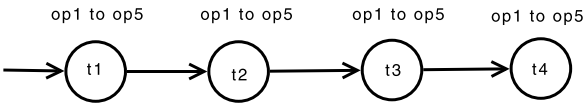


Fig. 16 Scientific workflow for four successive simulations

of selected services. For $\alpha = 10$, however, due to the large weight of the execution time, services with higher costs and lower times are preferred. This is consistent with lowest values of $\alpha t_{ij} + c_{ij}$ for services for particular values of α resulting from Table 1.

$\upsilon_{\text{MIN\_T\_C\_BOUND}}$—in this case, it is not possible to use the aforementioned FASTHEU algorithm as in general it may not be able to meet the cost constraint. Figure 15 presents results obtained using the MILP algorithm. It can be seen that the better availability of services, the lower the workflow execution time. In all cases, the algorithm is able to meet the cost constraint.

Scientific workflow: optimization of execution time. In this example, a sequence of four tasks forms a workflow (Fig. 16). This is a frequent case in scientific computing where several tasks (e.g. simulations) need to be performed one by one where output data from one task are input for another. For each task, there are five alternative *opx* services. The execution time of service `opx` is $10(4.8 + x)$ seconds. 8.2 MB of data in 10 files is passed between tasks through the BeesyCluster server. The particular execution times and the data passed correspond to processing of data and conversion from PS to PDF using `ps2pdf14`. This or a similar ratio of computation/communication can also be considered as representative for some mathematical operations on input data etc. Various execution times result from various processing speeds of computing nodes. If, as in the previous cases, just-in-time reselection is required, the service with the lowest t_{ij} is chosen as costs are equal in this case where a dedicated environment is assumed. If the same application is published as

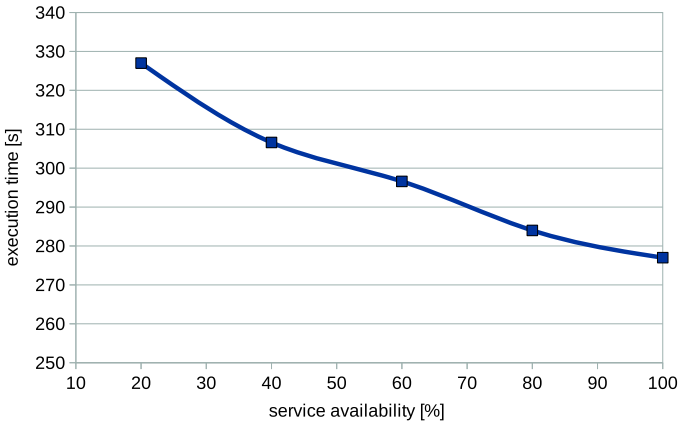


Fig. 17 Average execution time vs service availability for sequential scientific workflow, same costs for services

several services on various nodes with various speeds, such a workflow gives a reliable solution for minimizing the workflow execution time irrespective of node (and service) failures. Figure 17 presents an average execution time for given service availabilities. The better the availability, the lower the execution time, since there is higher probability that quicker services (due to faster nodes) are available.

It should be noted that for the given workflow application, various optimization algorithms may exhibit various performance. The total execution time comprises the execution time of a scheduling algorithm, execution of services selected for particular tasks, communication time and possibly rescheduling in case previously chosen services become unavailable or conditions have changed. There is a trade-off between the quality an algorithm returns that shortens the execution time of the workflow and the algorithm execution time that adds up to the total execution time. FASTHEU is fast but not optimal and turns out to be good for large workflow sizes, the genetic algorithm slow but able to find a solution for all considered criteria. MILP is optimal and best for small size workflows but unable to return optimal results for large ones. This is related to what information is used for scheduling and rescheduling. Yu et al. [32] considers meta-heuristics that optimize considering the whole workflow and heuristics taking into account only partial information about the workflow. Yu et al. [32] considers both deadline and budget constrained scheduling with bounds on the execution time and cost, respectively. It also discusses LOSS and GAIN algorithms for budget constrained scheduling which adjust the schedule resulted from optimization of only time or only cost to meet the cost constraint and optimize the execution time. Yu et al. [32] compares LOSS and GAIN to GA for budget constrained scheduling, and compares GA, deadline distribution among partitions [33] and back-tracking algorithms for deadline constrained scheduling problem. Chin et al. [8] proposes an adaptive scheduling algorithm with rescheduling ALSS (Adaptive List Scheduling for Service) for optimization of workflow makespan and states it is better than AHEFT, SLACK, max-min, min-min and myopic algorithms. Yu and Buyya [30] shows application of the genetic algorithm for workflow application

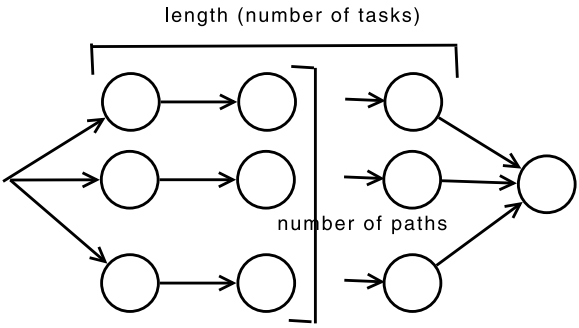


Fig. 18 Workflow for performance/overhead testing

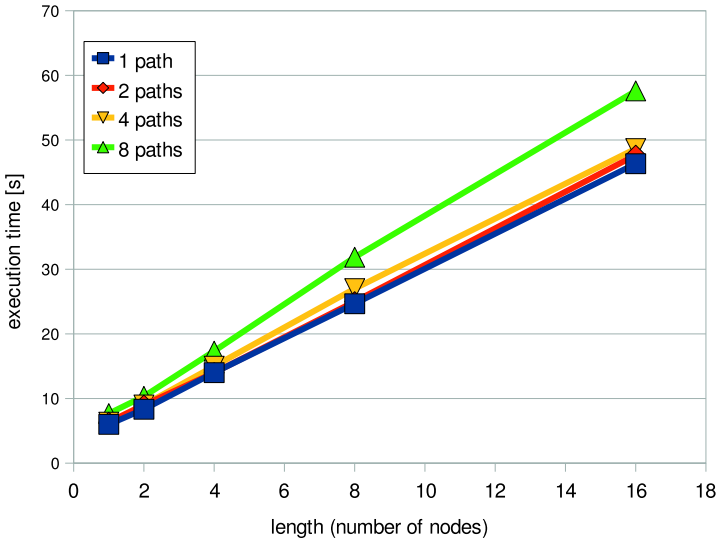


Fig. 19 Execution time vs length and number of parallel paths

scheduling. Reference [6] concludes that GAs are preferred for a large number of concrete services per abstract service, otherwise integer programming is better.

4.1 Overhead of the environment

To measure the overhead of the solution, the author has measured execution times for workflow applications of various sizes. The workflow is constructed from a certain number of parallel paths, out of which every one contains a given number of sequenced tasks (Fig. 18). For each task there is one "date" service that just prints the current date and time and returns immediately. The number of input files (each file is of length 0) is equal to the number of parallel paths so each service takes one input file as input and passes it on to the output.

The total execution time of the workflow is presented in Fig. 19 for various lengths and various numbers of parallel paths. Section 3.4 discussed how multithreading is

used in the startup phase of workflow execution. This results in very similar execution times for various numbers of paths such as 1–4. Apparently, the initialization phase takes approximately 5 seconds. Since the whole overhead time includes initialization like preparation of dedicated directories for services, authorization of the user to run each service, remote invocation of each service via SSH, the overhead is reasonable. It should be noted that in the latest version, BeesyCluster uses pooling of SSH sessions with various clusters which means that subsequent remote commands via SSH use the same session.

5 Conclusions and future work

The author presented a complete environment for modeling scientific and business workflow applications modeled as acyclic directed graphs. Nodes of the graph model simple tasks for which independent services capable of executing given tasks and offered by various providers in the BeesyCluster middleware can be mapped. One of several optimization algorithms can be run before workflow execution starts and also when previously chosen services become unavailable or conditions on which services are offered have changed.

Two practical applications have been tested in a real environment for various service availabilities proving the solution can adapt to available services. The applications can be considered as templates that can be reused and extended for several other scientific or business applications.

It was shown that the overhead of the execution engine is low compared to execution times that can be expected in practical workflows.

Future work will focus on support of handling extremely large data sets processed by workflow services and testing more workflow applications.

Acknowledgement Work sponsored by research grant N N516 383534 “Strategies for management of information services in distributed environments”.

References

1. Aggarwal R, Verma K, Miller J, Milnor W (2004) Constraint driven web service composition in meteors. In: Proceedings of IEEE international conference on services computing (SCC’04), pp 23–30
2. Ben Mokhtar S, Fournier D, Georgantas N, Issarny V (2005) Context-aware service composition in pervasive computing environments. In: Rapid integration of software engineering techniques, second international workshop: RISE, pp 129–144. Heraklion, Crete Greece. http://hal.archives-ouvertes.fr/inria-00415111/en/
3. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K (2005) Task scheduling strategies for workflow-based applications in grids. In: CCGrid 2005, IEEE international symposium on cluster computing and the grid, vol 2, pp 759–767
4. Canfora G, Penta MD, Esposito R, Villani M (2004) A lightweight approach for QoS-aware service composition. ICSOC 2004 forum, IBM Tech Report Draft

5. Canfora G, Penta MD, Esposito R, Villani M (2005) Qos-aware replanning of composite web services. In: Proc of IEEE international conference on web services, vol 1, pp 121–129. Res Centre on Software Technol, Sannio Univ, Italy
6. Canfora G, Penta MD, Esposito R, Villani ML (2005) An approach for qos-aware service composition based on genetic algorithms. In: GECCO'05: proceedings of the 2005 conference on genetic and evolutionary computation. ACM, New York, pp 1069–1075. doi:10.1145/1068009.1068189
7. Cardoso J, Sheth A, Miller J (2002) Workflow quality of service. Tech Rep, LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA 30602, USA
8. Chin SH, Suh T, Yu HC (2010) Adaptive service scheduling for workflow applications in service-oriented grid. J Supercomput 52(3):253–283. doi:10.1007/s11227-009-0290-9
9. Czarnul P (2006) Integration of compute-intensive tasks into scientific workflows in BeesyCluster. In: Computational science—ICCS 2006. LNCS, vol 3993. Springer, Berlin, pp 944–947
10. Czarnul P (2010) Modelling, optimization and execution of workflow applications with data distribution, service selection and budget constraints in BeesyCluster. In: Proceedings of 6th workshop on large scale computations on grids and 1st workshop on scalable computing in distributed systems, international multiconference on computer science and information technology, IEEE catalog number CFP0964E, Wisla, Poland, pp 629–636
11. Czarnul P, Kurylowicz J (2010) Automatic conversion of legacy applications into services in BeesyCluster. In: Proceedings of 2nd international IEEE conference on information technology ICIT'2010, Gdansk, Poland
12. Czarnul P, Bajor M, Fraczak M, Banaszczyk A, Fiszer M, Ramczykowska K (2005) Remote task submission and publishing in BeesyCluster: Security and efficiency of web service interface. In: Proceedings of PPAM, Poznan, Poland. LNCS, vol 3911. Springer, Berlin
13. Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Patil S, Su MH, Vahi K, Livny M (2004) Pegasus: mapping scientific workflows onto the grid. In: Across grids conference, Nicosia, Cyprus. http://pegasus.isi.edu
14. Hackmann G, Haitjema M, Gill C, Catalin Roman G, (2006) Sliver: A bpel workflow process execution engine for mobile devices. In: Proceedings of 4th international conference on service oriented computing (ICSOC). Springer, Berlin, pp 503–508
15. Han J, Kim E, Choi J (2004) Workflow language based on web services for autonomic services in ubiquitous computing. In: Proceedings of international conference on artificial reality and telexistence, ICAT, Coex, Korea
16. Laboratory of Parallel and Distributed Systems, MTA SZTAKI, Hungary: Parallel grid runtime and application development environment, User's manual, ver. 8.4.2. http://www.lpds.sztaki.hu/~smith/pgrade-manual/manual.html
17. Li J, Bu Y, Chen S, Tao X, Lu J (2006) FollowMe: on research of pluggable infrastructure for context-awareness. In: 20th International conference on advanced information networking and applications, AINA 2006, vol 1. doi:10.1109/AINA.2006.182
18. Ludascher B, Altintas I, Berkley C, Higgins D, Jaeger-Frank E, Jones M, Lee E, Tao J, Zhao Y (2005) Scientific workflow management and the Kepler system. Concurrency and computation: practice & experience, special issue on scientific workflows. http://www.sdsc.edu/%7Eludaesch/Paper/kepler-swf.pdf
19. Majithia S, Shields MS, Taylor IJ, Wang I (2004) Triana: a graphical web service composition and execution toolkit. In: IEEE international conference on web services (ICWS'04). IEEE Computer Society, Los Alamitos, pp 512–524. http://www.trianacode.org/
20. Patel C, Supekar K, Lee Y (2003) A QoS oriented framework for adaptive management of web service based workflows. In: Proceedings of the 14th international database and expert systems applications conference (DEXA 2003), Prague, Czech Republic, 2003. LNCS. Springer, Berlin, pp 826–835
21. Rao J, Su X (2005) A survey of automated web service composition methods. In: LNCS, vol 3387/2005. Springer, Berlin, pp 43–54. http://www.springerlink.com/content/4m6w37g0jffk9bv4
22. Sakellariou R, Zhao H, Tsiakkouri E, Dikaiakos M (2007) Scheduling workflows with budget constraints. In: Gorlatch S, Danelutto M (eds) Integrated research in GRID computing, CoreGRID. Springer, Berlin, pp 189–202. http://www.cs.man.ac.uk/rizos/papers/coregrid2005a.pdf
23. Şensoy M, Yolum P (2007) On choosing an efficient service selection mechanism in dynamic environments. In: Proceedings of the 9th international workshop on agent-mediated electronic commerce (AMEC IX), vol 13, pp 105–118
24. Srinivasan N, Paolucci M, Sycara K (2004) Adding owl-s to uddi, implementation and throughput. In: First international workshop on semantic web services and web process composition (SWSWPC 2004), San Diego, USA

25. System bexee: Bpel execution engine (2004) http://bexee.sourceforge.net/index.html, Berne University of Applied Sciences
26. The ActiveBPEL Engine (2009) http://www.activevos.com/community-open-source.php, Active endpoints
27. Wilkinson B, Allen M (1999) Parallel programming: techniques and applications using networked workstations and parallel computers. Prentice Hall, New York
28. Yingchun, Li X, Sun C (2007) Cost-effective heuristics for workflow scheduling in grid computing economy. In: GCC'07: proceedings of the sixth international conference on grid and cooperative computing. IEEE Computer Society, Los Alamitos, pp 322–329. doi:10.1109/GCC.2007.57
29. 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
30. Yu J, Buyya R (2006) A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In: Workshop on workflows in support of large-scale science, proceedings of the 15th IEEE international symposium on high performance distributed computing (HPDC), Paris, France
31. Yu J, Buyya R (2006) Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci Program J 14(3–4):217–230
32. Yu J, Buyya R, Ramamohanarao K (2008) Workflow scheduling algorithms for grid computing. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.144.7107
33. Yu J, Buyya R, Tham CK (2005) Cost-based scheduling of workflow applications on utility grids. In: Proceedings of the 1st IEEE international conference on e-science and grid computing (e-Science), Melbourne, Australia. IEEE Comput Soc, Los Alamitos
34. Zeng L, Benatallah B, Dumas M, Kalagnanam J, Sheng Q (2003) Quality driven web services composition. In: Proceedings of WWW, Budapest, Hungary
35. Zeng L, Benatallah B, Ngu AH, Dumas M, Kalagnanam J, Chang H (2004) Qos-aware middleware for web services composition. IEEE Trans Softw Eng 30(5):311–327. doi:10.1109/TSE.2004.11