Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 629–636
ISBN 978-83-60810-27-9
ISSN 1896-7094

Modelling, Optimization and Execution of Workflow Applications with Data Distribution, Service Selection and Budget Constraints in BeesyCluster

Pawel Czarnul
Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland
Email: pczarnul@eti.pg.gda.pl, http://fox.eti.pg.gda.pl/~pczarnul

***Abstract*—The paper proposes a model which allows integration of services published by independent providers into scientific or business workflows. Optimization algorithms are proposed for both distribution of input data for parallel processing and service selection within the workflow. Furthermore, the author has implemented a workflow editor and execution engine on a platform called BeesyCluster which allows easy and fast publishing and integration of scientific and business services. Several tests have been implemented and run in BeesyCluster using services for a practical digital photography workflow with and without budget constraints. Two alternative goals are considered: minimization of the execution time with a budget constraint or a linear combination of cost and time.**

I. Introduction

SERVICE integration has become the main focus in distributed environments. Firstly, Service Oriented Architecture (SOA) promoting loosely coupled services interacting with each other, secondly Open Grid Service Architecture in the context of grid computing laid fundamental concepts for modern distributed applications. In both cases, workflow management systems using Web Services and grid middlewares allow composition of individual services into complex workflows. In the classic approaches, tasks are either mapped to available resources or available services are assigned to tasks so that QoS metrics for particular tasks or for the whole workflow are optimized. This can be minimization of the execution time with a budget constraint on the costs of resources or services used. Furthermore, many of the practical workflows are data-intensive and require data partitioning for parallel processing. From this point of view the author proposes a model with data partitioning for parallel execution and service selection, both important for practical workflows.

From the infrastructure point of view, it is desirable for a service provider to publish scientific or business services easily and instantly, define prices and other QoS parameters as well as be able to manage the service. For the client, it is important to be able to integrate such services from various providers. This requires selection of best services considering imposed QoS constraints and the goal followed by execution of selected services. The author has implemented these in BeesyCluster which satisfies all of these requirements for scientific and business environments.

II. Related Work and Motivations

Firstly, *task scheduling* considers mapping of workflow tasks to available resources so that workflow execution time is minimized [1], [2]. In the classic problem, task execution times are known in advance, neither data flows nor other metrics such as costs are usually considered. Some works take into account execution costs of particular resources [3]. Paper [4] considers resource requirements for tasks of the workflow.

Secondly, in *scheduling in utility grids* [5] or *workflow scheduling in grids* [6] for each task t_i we distinguish a set of services S_i out of which only one service is to be chosen to execute t_i. Other attributes such as service costs are considered [7]. As considered by [8] and [5] the goal is to find the best assignment of $t_i \rightarrow (s_k, t_{ik}^{st})$ where s_k is a service able to execute task t_i and t_{ik}^{st} is the starting time of 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 selected services below a predefined minimum. In the context of typical business interactions, the *QoS service selection/workflow composition* problem is stated. Compared to the scheduling problem, many more quality attributes are considered without dependencies between or overlapping of services executing different tasks. The goal is to select proper services so that a function of QoS metrics such as: execution time, cost, availability [9], [10], [11], accessibility [10], fidelity [12] or conformance [10], security [10], reputation [9] is minimized possibly also with additional conditions imposed on some of them [11], [13]. For instance, the cost of using the services must not exceed the given budget.

There exist several workflow management systems for grid computing. Paper [2] describes and compares Gridbus, Kepler [14], Pegasus [15], Triana [16], P-GRADE [17], Directed Acyclic Graph Manager (DAGMan), ICENI, GridFlow, GrADS, Askalon, UNICORE, Taverna, GridAnt. These grid-oriented systems mainly use middlewares such as Globus Toolkit, Grid Application Toolkit or other resource management systems etc. for running jobs. More business oriented environments focusing on execution of BPEL or semantic service discovery and compositon such as Meteor-S [18] usually do not offer support for HPC environments although [19] presents how to use BPEL for grid environments.

Such workflow management systems support data parallelism and various operations on the data [20], [21]. In Gridbus, input parameters can be defined as parameter values, files or data streams [20]. In Process Networks in Kepler the workflow is driven by data availability and can be used for parallel processing on distributed systems [22]. Since the service may receive data as it runs and possibly from various predecessors, data processing can be defined in many ways on the input data. In the one-to-one composition pattern successive portions of input data from two input sources are processed pairwise [21]. Paper [21] demonstrates that MOTEUR can handle one-to-one and all-to-all patterns as well as workflow, data and service parallelism and is compared to Taverna, Kepler and Triana which lack some of these features. [23] optimizes workflow makespan also considering the possibility of dynamic deployment of services on various resources to save large data communication costs.

From the point of view of the model, the contribution of this work is a combined model (Section III) with both data distribution for parallel computing by parallel tasks and selection of services to optimize various QoS goals and meet QoS constraints. The consumer constructs a workflow i.e. a complex scenario composed out of simple tasks. It is assumed that the workflow graph can contain both sections with parallel tasks for among which input data is distributed for parallel processing. In this case each task has a single service assigned to it. Also, the graph may contain tasks with alternative services out of which one needs to be chosen to perform the given task. As an example, the workflow shown in Figure 1 allows parallel processing of several input digital RAW images by several filters. These can be executed by services created from free applications such as `dcraw`, `RAWtherapee` or possibly offered by companies who offer own paid image processing software. A comparison of RAW converters is available at `http://www.photozone.de/conclusion-on-going`. It finally produces a Web album out of images previously processed in parallel. The problem is to find both: data distribution – partition input data for parallel processing by parallel paths to speed up workflow execution and service selection – a service to execute each task – for each task t_i there may be one or more services (s_{ij}) capable of executing the task at price per unit of data c_{ij} (one out of two web album generation services $s_{19\,0}$, $s_{19\,1}$ can be chosen for task t_{19}). The goal is to minimize one of the following: $v_{MIN\_T\_C\_BOUND}$ – minimization of the workflow execution time with a constraint that the total cost of selected services does not exceed the given budget or, v_{MIN_TC} – minimization of a linear combination of workflow execution time and the total cost of services. Services may be offered by independent providers for potentially various prices and each service is installed on a particular cluster or server of a predefined speed. Incompatibility between formats used by various providers is considered as additional cost in the model. If there is a budget constraint set then more tasks will be sent along cheaper paths and more expensive paths will not be used. What is important, the model proposed by the author and presented in Section III considers alternative approaches as to how data is passed between tasks. As discussed in Section III-A, either the next task starts when all data has been processed by preceding tasks or processing data in streams for overlapping communication and computations is proposed. In [24] the author has presented how to choose services at runtime for highly changeable environments rather then before workflow execution considered in this work.

From the infrastructure point of view, the contribution is the author's implementation of the model in BeesyCluster (Section VI) taking advantage of features of the latter. Secondly, the paper demonstrates usefulness of the solution for a real world workflow application (Section VII). BeesyCluster is the middleware for the workflow system described in Section VI in this work. BeesyCluster differs from other middlewares as it accesses services published by users through user accounts via SSH. This means that contrary to many other systems adding new clusters or registering new user accounts is a matter of seconds as requires just corresponding entries in the database regarding access. Similarly, each user can become a provider by publishing their parallel or sequential applications installed on various clusters as services in a matter of seconds in the file manager in BeesyCluster. This allows publication of scientific services from HPC clusters as BeesyCluster hides low level details like queueing systems. Also, businesses can publish services from their own servers attached to BeesyCluster. This also brings pricing for not only business but also scientific services and allows composition of both types.

III. Model of the Workflow Scheduling with Data Distribution

Consequently, the author proposes a model with data distribution in which a directed graph $G(V, E)$ represents a workflow where nodes V correspond to tasks while edges E denote task dependencies. There is at least one starting node with initial data and one termination node which terminates computations. The model allows: a sequence – for connected tasks, a service connected to the successor will be executed only after the service selected for the predecessor has completed (t_1 and t_2 in Figure 1), fork – services associated with tasks following the forked task can potentially be executed in parallel (if run on separate processors – e.g. t_1, t_4, ..., t_{16} in Figure 1), join – the service selected for the task to which other tasks are connected will be executed only after each of the predecessors has finished (in Figure 1 t_{19} will execute only after t_3, t_6, ..., t_{18} have finished).

The model distinguishes the following: a set of services $S_i = \{s_{i0}, s_{i1}, ..., s_{i(|S_i|-1)}\}$ out of which only one must be selected to execute task t_i, c_{ij} – cost of processing a unit of data by service s_{ij}, P_{ij} – the provider of service s_{ij}, N_{ij} – the node service s_{ij} runs on while sp_n the speed of node n, d_{ij}^{in} and d_{ij}^{out} are the size of the input data and the size of the data produced by service s_{ij} bound with $d_{ij}^{out} = f_{t_i}(d_{ij}^{in})$. d_i denotes the size of data processed by task t_i. d_{ijkl} denotes the size of data to be sent from service s_{ij} to service s_{kl}. f_{t_i} denotes how the size of output data for task t_i depends on the size of input data. Then the required

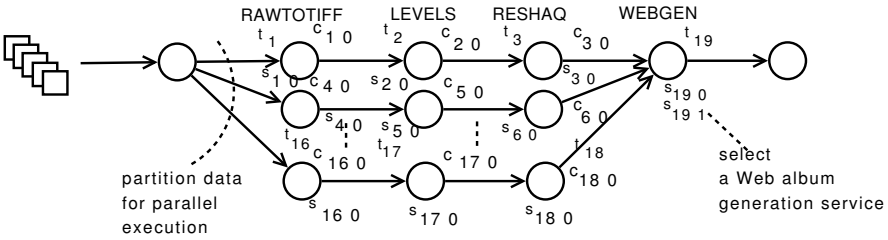

Figure 1: A Parallel Digital Photo Workflow with Budget Constraints

constraints on these variables would include: $d_i = \sum_j d_{ij}^{in}$, $d_{kl}^{in} = \sum_{i,j:(v_i,v_k)\in E} d_{ijkl}$, $d_{ij}^{out} = f_{t_i}(d_{ij}^{in})$. $d_{ij}^{in} > 0$ for only one selected service s_{ij} for task t_i. Output data from a task can be sent and/or partitioned into input files of successors. In particular, the following alternatives are possible: All Fork (AF) task i.e. t_{ij}: $\forall_{k:t_k\in Succ(t_i)} d_{ij}^{out} = \sum_l d_{ijkl}$ or Distribute Fork (DF) task i.e. t_{ij}: $d_{ij}^{out} = \sum_{l,k:(v_i,v_k)\in E} d_{ijkl}$. Then assuming $t_{ijkl}^{comm}(d_{ijkl}) = t_{N_{ij}N_{kl}}^{startup} + \frac{d_{ijkl}}{bandwidth_{N_{ij}N_{kl}}}$ – communication time of data of size d_{ijkl} sent from service s_{ij} and s_{kl}, $t_{ij}^{exec}(d_{ij}^{in})$ – the execution time of service s_{ij}, t_{ijkl}^{tr} – additional time for data conversion between output/input formats if connected services are offered by various providers, $t_i^{st} : i \in |V|$ – the time at which service s_{ij} chosen to execute t_i starts processing it, we have $\forall_{i,k:(v_i,v_k)\in E}\ t_k^{st} \geq t_i^{st} + \sum_j t_{ij}^{exec} + \sum_{j,l} t_{ijkl}^{comm} + \sum_{j,l} t_{ijkl}^{tr}$. t_{ij}^{exec} will be larger than 0 only for one j. Similarly, for the given i and k t_{ijkl}^{comm} and t_{ijkl}^{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. Additionally, $t^{workflow} = t_{termination}\ \ not\exists_q(v_{termination}, v_q) \in E$

We consider two alternative minimization goals where $t^{workflow}$ is the time when the last service finishes:

1) min $v_{MIN\_T\_C\_BOUND} = t^{workflow}$ while adding a constraint on the total cost i.e. $\sum d_{ij}^{in} c_{ij} < B$ where B is the budget (problem MIN_T_C_BOUND).
2) min $v_{MIN\_TC} = \alpha t^{workflow} + \sum d_{ij}^{in} c_{ij}$ (problem MIN_TC), $\alpha > 0$.

A. Data Processing

For data processing and communication between tasks, three possibilities are considered:

1) synchronized: as assumed above, before execution of the given task starts, all data processed by previous tasks must be sent and ready,
2) overlapping: instead of $t_i \rightarrow t_j$ we can introduce parallel paths $t_i \rightarrow t_{j'}$, $t_i \rightarrow t_{j''}$, ..., $t_i \rightarrow t_{j'^n}$ such that every $t_{j'^l}$ is in fact the same task as t_j and has the same services assigned to it as t_j and a constraint that the same service is chosen for every one of them. Instead of passing whole data d_i to t_j, $\frac{d_i}{n}$ can be passed to every following task independently. This means that services for $t_{j'^l}$ will be executed faster. Secondly, if copying to every $t_{j'^l}$ is delayed compared to $t_{j'^{l-1}}$ this results in overlapping communication and computations.
3) streaming: after receiving a unit of data, each task processes it and passes to a following task(s) and does so for every unit.

IV. Algorithms

The author has proposed and implemented three different algorithms to solve the problem:

1) genetic algorithm (GA) – a chromosome encodes both the assignment of services to tasks and data distribution among tasks. The algorithm does not impose linearity on the constraints but it comes at the cost of slower convergence compared to MGALP described next.
2) mixed genetic algorithm and linear programming (MGALP) – where the genetic algorithm is used to determine the assignment of services to tasks and fast linear programming [25] is used to determine optimal data distribution for the given schedule; a chromosome encodes just the assignment of services to tasks,
3) mixed integer linear programming (MILP) – the author adopts MILP traditionally used for solving the service selection problem and extends it for scheduling and solving data flows. The algorithm solves the problem optimally but at a high computational cost for large problem sizes. Similarly to [9], [18], [26], integer variables denote which service is selected for the particular task while real or integer variables sizes of data sent between tasks.

GA and MGALP are much faster for larger graphs at the cost of lower quality results. Due to space limitations, only GA is presented in more detail. For large workflows, the algorithm may take more steps to arrive at an acceptable data distribution or even such that make constraint $\sum d_{ij}^{inp} c_{ij} < B$ satisfied for goal MIN_T_C_BOUND. To make the starting solution better, for each of the initial chromosomes, linear programming was used to help find better data flows and following iterations used only crossover and mutation operators.

Each chromosome consists of the representation for the schedule shown in Figure 2.

1) the first $|V|$ numbers contain indexes of selected services for each task i.e. $0 \leq e_i < |S_i|\ 0 \leq i < |V|$. Initial values are chosen by a random selection of services for each task i.e.: $e_i = random()\ mod\ S_i$;
2) 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

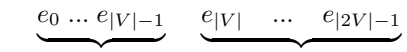

Figure 2: GA's Representation

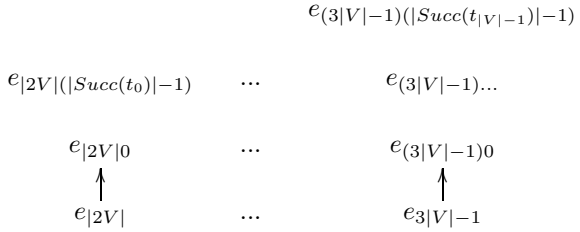

Figure 3: GA's Representation for Flow Distribution

execute them), t_l is executed after t_k iff $e_i = k\ e_j = l\ :$ $i < j\ i \geq |V|$. Initial values are chosen randomly out of 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.

Additionally each chromosome contains a distribution of d_{ij}^{out} into data of the following tasks. For DF tasks data from t_i is simply copied to each task in $Succ(t_i)$. For AF tasks the structure shown in Figure 3 is used. For task t_i, since data is divided into all following tasks, there is an array of $|Succ(t_i)|$ $e_{(2|V|+i)j}$ elements for this task such that $\sum_{j=0}^{|Succ(t_i)|-1} e_{(2|V|+i)j} = 1$ and d_{ij}^{out} $e_{(2|V|+i)y} = \sum_{t_k:t_k \text{ is } y\text{-th successor of } t_i \text{ out of tasks in } Succ(t_i)} d_{ijkl}$ which denotes how output data for task t_i is split into input data of following tasks.

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 from the second chromosome. The order is taken from one of the two chromosomes with probabilities 0.35 and a new one is generated randomly with probability 0.3 (Figure 4). The algorithm for data distribution is shown in Figure 5. Function `SetRndDistr` assigns the flow to only one successor with probability 0.5 or divides it into more following tasks with smaller probabilities. This ensures that the algorithm can both direct the flow to a small and a large number of following tasks.

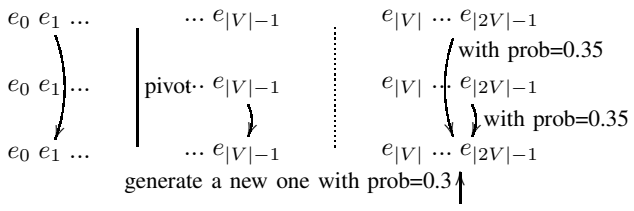

Figure 4: Crossover for Service Selection

Mutation for service selection is applied as follows – a new service is selected for a task with the probability of 0.1 and a new order is generated with the probability of 0.02. Function `SetRndDistr` is invoked for a new distribution with probability 0.05.

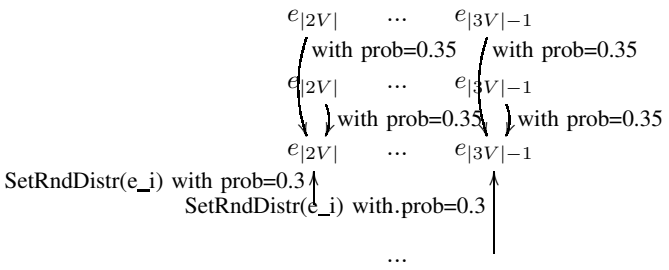

Figure 5: Crossover for Data Distribution

This version can be adjusted to use only integer data sizes by rounding d_{ijkl} values which result from $e_{|2V|}$ to $e_{3|V|-1}$ distributions to integer numbers such that constraints III are satisfied and successive evaluation of the chromosome using integers.

V. Optimal vs Heuristic Algorithms

The author has run the three implemented algorithms for workflow graphs with varying numbers of nodes (tasks) and services assigned to each task. The main goal is to assess workflow sizes for which the MILP algorithm is able to return an optimal solution within a reasonable time limit of 20 seconds. For the given number of nodes, out of all possible edges between the nodes, 10% are chosen randomly. Then all nodes without predecessors are connected to the initial node and all without successors connected to the terminating node. For this particular test service costs are random from the $[0, 40]$ range, service execution coefficients $e_{ij} : f_{eij}(d) = e_{ij}d$ are random from the $[0, 4]$ range, the time coefficient $\alpha = 100$.

MILP is able to solve only small problems with a small number of tasks and services optimally. Since e.g. solving for MIN_T_C_BOUND is NP-hard, heuristic algorithms need to be used for large workflows. Since it is not possible to compare to an optimal algorithm, for MGALP and GA, we measure the improvement the algorithm is able to make over an initial value (for goal MIN_T_C_BOUND or MIN_TC) for the first chromosome of the population $v_{initial\ i}$. The improvement is computed as follows:

$$impr[\%] = \frac{100}{sc} \sum_{i=1}^{sc} \frac{v_{initial\ i} - v_{final\ i}}{v_{initial\ i}} \qquad (1)$$

where sc denotes the number of simulations. It must be noted that both for GA and MGALP the initial chromosomes are set using random schedules and data distribution computed using linear programming. This allows GA to have a reasonable start for tuning using only crossover and mutation. $v_{initial\ i}$ is always obtained by taking the value from the initial chromosome of the population. For MIN_T_C_BOUND it may not meet the cost bound though. If this is the case, $v_{initial\ i}$ is obtained using linear programming minimizing the cost and taking the workflow time obtained for the minimum cost (if the minimum

cost is lower than the bound set). In the experiments B is set to $bestcost + \frac{worstcost-bestcost}{4}$ where $bestcost$ is the minimum possible workflow cost without considering the workflow time and $worstcost$ is the cost corresponding to minimization of only workflow execution time.

Figures 6 and 7 present which algorithm is best for a workflow with a particular number of tasks and services marked on the two axes respectively. Each algorithm is given 20 seconds and results are compared. Each measured number is an average from 50 runs for a particular input data set. MILP is best for small graphs as it generates optimal results. For larger graphs, either MGALP or GA are preferred. Colors denote the ratio of improvement over the initial solution of GA divided by the improvement for MGALP. The range of input data for which MILP completes in 20 seconds (marked as OPT) and thus is best is limited (white color). Light areas correspond to workflow sizes for which GA gives better results than MGALP.

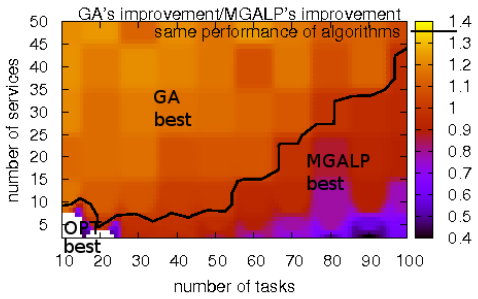

Figure 6: Comparison of OPT, MGALP, GA within 20 seconds vs Task Count and Average Service Count MIN_TC

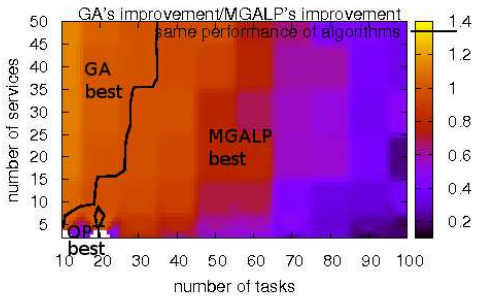

Figure 7: Comparison of OPT, MGALP, GA within 20 seconds vs Task Count and Average Service Count MIN_T_C_BOUND

VI. Modelling, Scheduling and Execution of Workflow Applications in BeesyCluster

The author has created an environment for modelling, scheduling and execution of workflow applications implementing the proposed model and algorithm. It is embedded in BeesyCluster. The latter is a JEE-based front-end and middleware which allows publishing and consuming services offered by various providers and consumers from locations managed by them. BeesyCluster, designed and co-developed by the author, was deployed at Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology for integration of clusters and laboratories. It is used for research and teaching high performance computing.

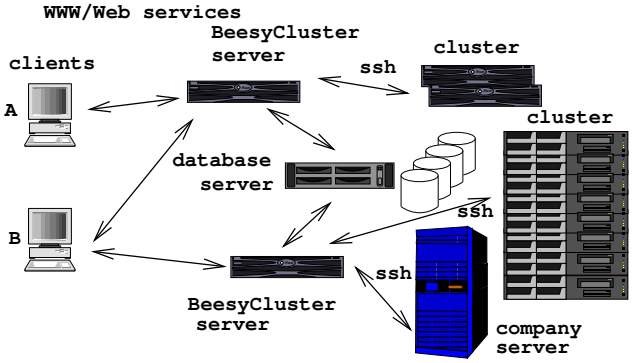

Figure 8: BeesyCluster Architecture

BeesyCluster allows the registered user to access accounts on various clusters or servers via WWW/Web Services (Figure 8). [27] presents details and performance of the Web Service interface. The system accesses these accounts via SSH. It allows to manage resources on these accounts allowing copying, editing files, management of directories and archives in a versioning system, compilation, queueing and even running applications interactively in a graphical mode via the browser.

Advantages of this complete environment with workflow support are as follows:

- the user can publish a sequential or a parallel (e.g. MPI) application from his/her user account as a service This brings business features to the scientific environment,
- companies can publish commercial applications as services from their servers,
- such services can be incorporated into workflows as described by the model,
- if the service is deployed on a cluster with a queueing system (e.g. PBS, LSF), the latter will be used 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 the real distributed environment.

The author has implemented a workflow editor in BeesyCluster shown in Figure 9 implementing the model proposed in Section III. It allows to define a workflow graph including tasks and dependencies, assign services to tasks, define input data which will be partitioned for parallel execution as well as one of the optimization goals. Data sizes including the size

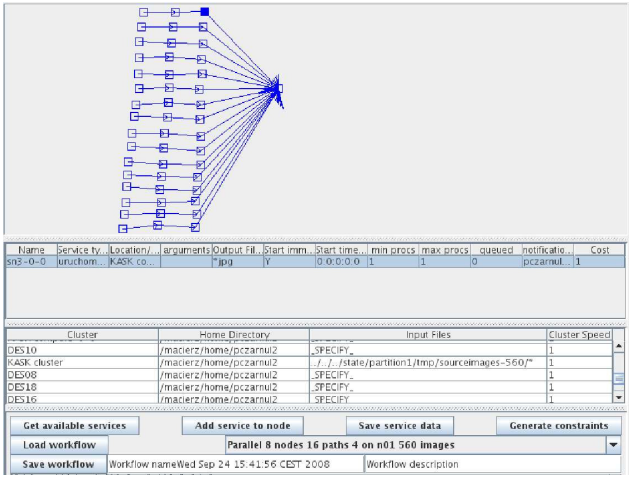

Figure 9: Workflow Editor in BeesyCluster

of input data and data flowing between services correspond to the numbers of files flowing between the services. Data distribution policies mentioned in Section III-A are possible. To run a workflow, BeesyCluster launches one of the algorithms implemented by the author to determine services for tasks and data distribution and executes the selected services handling data transfers between clusters and servers on which the services are installed. [24] and [28] present implementation details of the workflow execution engine which is also used in this work. However, [24] focuses on just-in-time service selection while [28] on incorporation of a checkpoint/restart mechanism for execution of scientific workflows.

VII. Compute and Data Intensive Digital Photography Workflow

The workflow shown in Figure 1 was modelled and tested in BeesyCluster. It shows a scenario in which there are three filters applied on input RAW images to generate a final web album to be published in the photographer's portfolio. Optimization problems include: partitioning a set of input images among services installed on nodes of various speeds, a trade-off between the budget for the execution and the execution time (some services are more expensive than others).

1) Testbed Environment and Services: The environment consists of 16 nodes installed at the Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology. The speeds of all the nodes used are identical for this application. One BeesyCluster server located at `https://lab527.eti.pg.gda.pl:10030/ek/AS_LogIn` was used.

The following services were used to execute the tasks (Figure 10): RAWTOTIFF – conversion from a RAW format to TIFF: each service is performed by script `dcraw -T $1` using the well known `dcraw` (`http://cybercom.net/~dcoffin/dcraw/`) converter, LEVELS – level adjustments so that whites, blacks and middle tones are mapped correctly: each service is performed by script

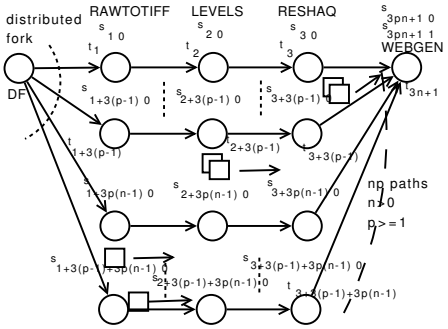

Figure 10: A Parallel Digital Photo Workflow with Several Paths per Node

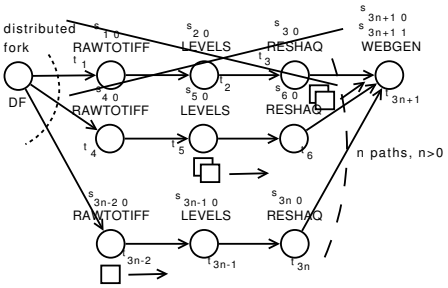

Figure 11: A Parallel Digital Photo Workflow with Budget Constraints

`convert $1 -normalize $1` using the `ImageMagick` (`http://www.imagemagick.org/script/index.php`) tool, RESHAQ – resizing, sharpening and saving as a 97% JPEG image for further generation of a web album: each service is performed by script `convert $1 -resize 600x400 -sharpen 1x1.2 -quality 97% $1.jpg` using the `ImageMagick` tool, WEBGEN – generation of a Web album: two services are installed on only one node (are alternative) and performed by applications `album` (`http://marginalhacks.com/Hacks/album/`) and `jigl` (`http://xome.net/projects/jigl/`) respectively.

2) Workflow Configurations and Results: The following configurations were tested (optimization goals are noted):

MIN_TC: parallel (Figure 10) – each filter is cloned into up up to $16p$ ($p \geq 1$) functionally equivalent t_{i+3k} $k \in \{0, 1, ..., 16p-1\}$ $i \in \{1, 2, 3\}$ tasks (Figure 10 also shows nodes the services are installed on). If $p = 1$ then each node waits for its batch of data to start processing. If $p > 1$ there are p services for each filter on a given node and it is possible to start processing faster as a smaller batch of input data can reach the service faster. Execution of services on one node is serialized. Each batch is processed independently allowing computations on the given node to start faster at the cost of running and managing more threads responsible for more paths. Since the cost of each service was set to the same value in this configuration the algorithm effectively minimizes the total execution time. Figures 12 and 13 present execution times and speed-ups obtained for 40, 160 and 560 input images

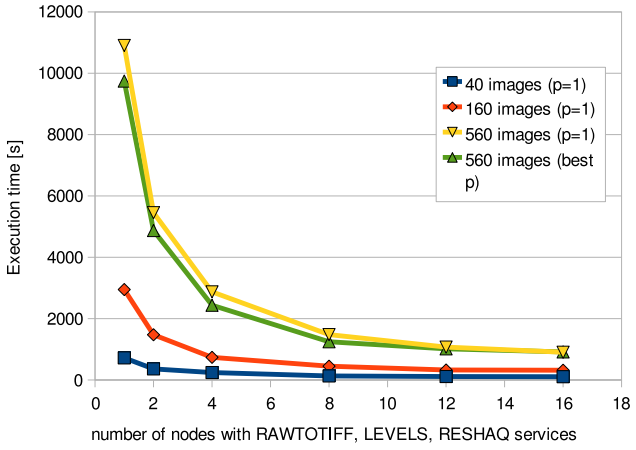

Figure 12: Digital Photography Workflow: Execution Times [s]

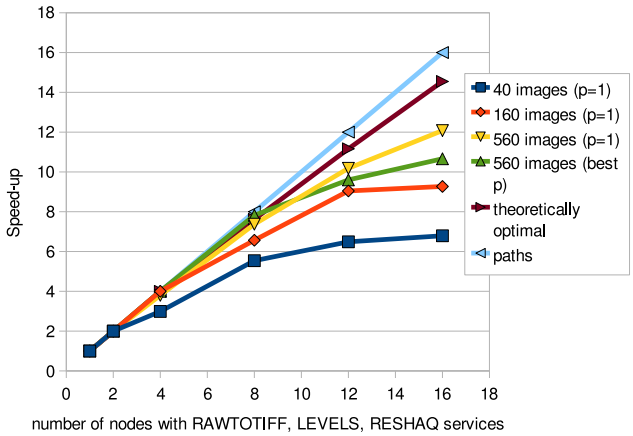

Figure 13: Digital Photography Workflow: Speed-up

(500MB, 2GB, 7GB of input data) respectively and from 1 to 16 nodes thus 1 to 16 paths.

As expected better speed-ups are obtained for a larger number of input images. For 560 images there is a comparison between $p = 1$ and best p for the given configuration. In the latter case, the time is lower as services can start processing earlier. BeesyCluster first sends a smaller batch to each service on a different node, then proceeds with following batches effectively overlapping communication and computations. It can be noted that the gain is much smaller for a larger number of nodes because the communication/transfer times between the tasks are much smaller for these configurations. Secondly, the node with the initial data may be a bottleneck for a large number of following tasks.

MIN_T_C_BOUND: parallel with budget constraints on the selection of services (Figure 11) – in this example the goal is to obtain the shortest possible execution time assuming the total cost of selected services does not exceed the given threshold. The literature proposes [29], [30] several pricing methods for services. Following the simplest strategy of charging per processor second and assuming the computational time

Table I: Services and Cost per Processor Second for Testbed Clusters

			cost per second	
Cluster	services	node count	day	night
1	$s_{1\,0}$ to $s_{24\,0}$	8	20	10
2	$s_{25\,0}$ to $s_{36\,0}$	4	10	20
3	$s_{37\,0}$ to $s_{48\,0}$	4	15	15

can be divided between clusters we obtain the same cost irrespective on the number of processors used [31]. However, if costs per processor can vary, we are faced with a non-trivial cost-performance trade-off. Similarly to [30] the author has distinguished services installed on three logical clusters with 8, 4 and 4 nodes with prices for a processor second as shown in Table I. The prices vary depending on the time of day. Two configurations were tested: clusters 1 and 2 (total of 12 nodes), clusters 1, 2 and 3 (total of 16 nodes) both during the day and the night. For each configuration the total allowed budget (for the parallel paths) is varied from the minimum cost allowing full parallelization to through 0.9 to 0.8 of this cost. Figures 14 and 15 show that if the budget is limited then the more expensive paths are not used which results in higher execution times. For day simulations, services from cluster 1 will be ommitted first, for night simulations services from cluster 2.

VIII. Summary and Future Work

The paper formulated a problem on how input data should be distributed among parallel tasks and how services should be selected for other workflow tasks so that a QoS goal is optimized while possibly other QoS constraints are met. Both the model, algorithms and an execution engine were implemented in BeesyCluster allowing to consume distributed services from various providers. As an example, a data and compute intensive digital photo workflow was executed in this environment. It has been demonstrated that the solution achieves good speed-ups and is able to select services in such a way that the execution time is minimized and the total cost of selected services does not exceed the budget. Clearly, the environment can be used for other workflow applications which can be easily constructed from distributed services deployed in BeesyCluster.

Further work will include testing more complex workflows in more distributed environments with services installed on distant clusters.

Acknowledgment

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

References

[1] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, “Task scheduling strategies for workflow-based applications in grids,” in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid*, vol. 2, May 2005, pp. 759–767.

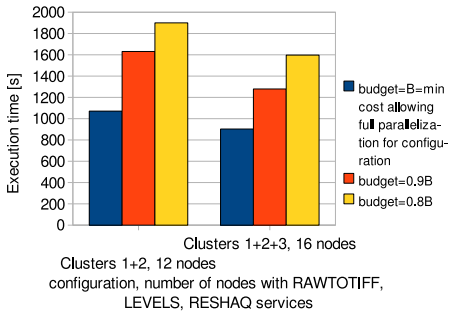

Figure 14: Digital Photography Workflow: Execution Time [s] under Cost Constraints: Day

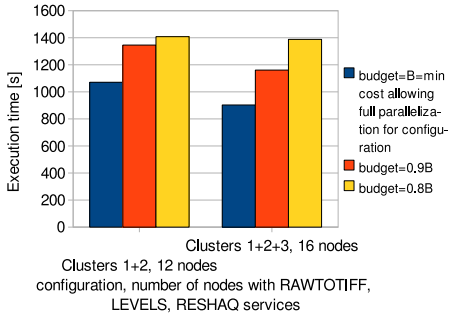

Figure 15: Digital Photography Workflow: Execution Time [s] under Cost Constraints: Night

[2] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, September 2005. [Online]. Available: http://dx.doi.org/10.1007/s10723-005-9010-8

[3] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos, "Scheduling workflows with budget constraints," in *Integrated Research in GRID Computing*, ser. CoreGRID, S. Gorlatch and M. Danelutto, Eds. Springer-Verlag, 2007, pp. 189–202. [Online]. Available: http://www.cs.man.ac.uk/ rizos/papers/coregrid2005a.pdf

[4] D. M. Quan and D. F. Hsu, "Mapping Heavy Communication Grid-Based Workflows Onto Grid Resources Within an SLA Context Using Metaheuristics," *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 330–346, 2008. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/22/3/330

[5] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming Journal*, 2006, iOS Press, Amsterdam.

[6] Yingchun, X. Li, and C. Sun, "Cost-effective heuristics for workflow scheduling in grid computing economy," in *GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 322–329.

[7] J. Yu and R. Buyya, "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 2006)*, Paris, France, June 2006.

[8] J. Yu, R. Buyya, and C.-K. Tham, "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 2005), IEEE CS Press*, Melbourne, Australia, December 2005.

[9] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality driven web services composition," in *Proceedings of WWW 2003*, Budapest, Hungary, May 2003.

[10] C. Patel, K. Supekar, and Y. Lee, "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)*, ser. LNCS, Prague, Czech Republic, September 2003, pp. 826–835.

[11] G. Canfora, M. D. Penta, R. Esposito, and M. Villani, "A Lightweight Approach for QoS-Aware Service Composition," iCSOC 2004 forum paper, IBM Technical Report Draft.

[12] J. Cardoso, A. Sheth, and J. Miller, "Workflow quality of service," LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA 30602, USA, Tech. Rep., March 2002.

[13] G. Canfora, M. D. Penta, R. Esposito, and M. Villani, "Qos-aware replanning of composite web services," in *Procs. of 2005 IEEE International Conference on Web Services*, vol. 1. Res. Centre on Software Technol., Sannio Univ., Italy, July 2005, pp. 121–129.

[14] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows*, 2005.

[15] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus : Mapping Scientific Workflows onto the Grid," in *Across Grids Conference*, Nicosia, Cyprus, 2004, `http://pegasus.isi.edu`.

[16] S. Majithia, M. S. Shields, I. J. Taylor, , and I. Wang, "Triana: A Graphical Web Service Composition and Execution Toolkit," in *IEEE International Conference on Web Services (ICWS'04)*. IEEE Computer Society, 2004, pp. 512–524.

[17] *Parallel Grid Runtime and Application Development Environment, User's Manual, ver. 8.4.2*, Laboratory of Parallel and Distributed Systems, MTA SZTAKI, Hungary.

[18] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint driven web service composition in meteor-s," in *Proceedings of IEEE International Conference on Services Computing (SCC'04)*, 2004, pp. 23–30.

[19] R.-Y. Ma, Y.-W. Wu, X.-X. Meng, S.-J. Liu, and L. Pan, "Grid-enabled workflow management system based on bpel," *Int. J. High Perform. Comput. Appl.*, vol. 22, no. 3, pp. 238–249, 2008.

[20] Gridbus Project, "Workflow language (xwfl2.0)," `gridbus.cs.mu.oz.au/workflow/2.0beta/docs/xwfl2.pdf`.

[21] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR," *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 347–360, 2008. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/22/3/347

[22] "Kepler user manual," May 2008.

[23] Z. Du, M. Wang, Y. Chen, Y. Ye, and X. Chai, "The triangular pyramid scheduling model and algorithm for pdes in grid," *Simulation Modelling Practice and Theory*, vol. 17, no. 10, pp. 1678 – 1689, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/B6X3C-4WYJSNC-2/2/ae37494233580d168fdaadd2ec4e5b76

[24] P. Czarnul, "A JEE-based Modelling and Execution Environment for Workflow Applications with Just-in-time Service Selection," in *proceedings of Grid and Pervasive Computing*, Geneva, Switzerland, May 2009.

[25] M. M. Syslo, N. Deo, and J. S. Kowalik, *Discrete Optimization Algorithms*. Prentice-Hall, 1983.

[26] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Dynamic web service composition in meteor-s," LSDIS Lab, Computer Science Dept., UGA, Technical Report, May 2004.

[27] P. Czarnul, M. Bajor, M. Fraczak, A. Banaszczyk, M. Fiszer, and K. Ramczykowska, "Remote task submission and publishing in beesycluster : Security and efficiency of web service interface," in *Proc. of PPAM 2005*, Springer-Verlag, Ed., vol. LNCS 3911, Poland, Sept. 2005.

[28] P. Czarnul, "Integration of compute-intensive tasks into scientific workflows in beesycluster," in *Computational Science – ICCS 2006*, ser. LNCS, vol. 3993. Springer, 2006, pp. 944–947.

[29] A. Caracas and J. Altmann, "A pricing information service for grid computing," in *MGC '07: Proceedings of the 5th international workshop on Middleware for grid computing*. New York: ACM, 2007, pp. 1–6.

[30] R. Buyya, D. Abramson, and J. Giddy, "A case for economy grid architecture for service oriented grid computing," in *IPDPS '01: Proceedings of the 10th Heterogeneous Computing Workshop*. Washington, DC, USA: IEEE Computer Society, 2001.

[31] E. Afgan and P. Bangalore, "Computation cost in grid computing environments," in *ESC '07: Proceedings of the First International Workshop on The Economics of Software and Computation*. Washington, DC, USA: IEEE Computer Society, 2007, p. 9.