Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 227–238. http://www.scpe.org
ISSN 1895-1767

PARALLELIZATION OF COMPUTE INTENSIVE APPLICATIONS INTO WORKFLOWS BASED ON SERVICES IN BEESYCLUSTER

PAWEL CZARNUL*

Abstract. The paper presents an approach for modeling, optimization and execution of workflow applications based on services that incorporates both service selection and partitioning of input data for parallel processing by parallel workflow paths. A compute-intensive workflow application for parallel integration is presented. An impact of the input data partitioning on the scalability is presented. The paper shows a comparison of the theoretical model of workflow execution and real execution times. The execution of this distributed workflow is compared to a highly parallel approach using MPI. Finally, results for an integrated workflow/MPI approach are shown.

Key words: workflow management, service selection, data partitioning, parallel computing, numerical integration

1. Introduction. Parallel and distributed processing is now possible thanks to a variety of architectures, application models and technologies. First, these are available for both shared memory and distributed memory processing within HPC clusters [30]. Secondly, approaches such as grid, sky and cloud computing [5] allow integration of services at a high level.

Several models and frameworks have been proposed for complex distributed applications. Quality of Service (QoS) needs to be considered where resources are usually shared among various users requesting their own jobs [13, 29, 35]. Such approaches need to offer enough flexibility in constructing a distributed application and provide the ability to process input data in parallel. Ideally, the user should not be involved in low level details such as selection of services for particular tasks or selection of resources to execute the code. The user should define functional and QoS goals and rely on automatic selection to meet the goals.

2. Related Work. Parallel processing is currently being implemented at various levels within:

1. computing nodes using GPU programming [2], programming many processor cores,
2. local systems such as HPC clusters in which computing nodes are interconnected with a reasonably fast interconnect such as Infiniband, Gigabit Ethernet etc. [6, 30],
3. at a distributed level using e.g. grid [17], cloud computing [5].

One of practical approaches for modeling complex distributed processing based on services is done using workflow applications. A workflow application is modeled as an acyclic graph in which vertexes denote tasks to be executed while directed edges dependencies between the tasks. For *scheduling in utility grids* [34]/*workflow scheduling in grids* [31] for each task t_i there is a set of services S_i out of which one service is selected to execute t_i. Other attributes such as service costs are considered [33]. As proposed by [34, 35] 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 using service s_k. Execution of t_i and t_j on one s_k must not overlap. The workflow execution time should be minimized while keeping the cost of selected services below a predefined minimum [7, 8]. In the context of typical business interactions, many more quality attributes are considered such as execution time, cost, availability [7, 27, 36], accessibility [27], fidelity [9] or conformance [27], security [27], reputation [36] is minimized. Usually no dependencies between or overlapping of services executing different tasks are considered in these applications.

From the infrastructure point of view, several MPI [26] implementations have been traditionally used for parallel programming. OpenMPI supports threads and MPI simultaneously. Additionally, several tools can be used for parallel programming within nodes such as CUDA [2], OpenCL, Pthreads [30] etc.

For the distributed workflow model, there are several workflow management systems for grid computing. Paper [32] provides a review and comparison of Gridbus, Kepler [23], Pegasus [16], Triana [25], P-GRADE [22], Directed Acyclic Graph Manager (DAGMan), ICENI, GridFlow, GrADS, Askalon, UNICORE, Taverna, GridAnt. These grid-oriented systems rely mainly on middlewares such as Globus Toolkit, Grid Application Toolkit or other resource management systems for starting and management jobs. Business oriented systems such as Meteor-S [3] incorporate support for BPEL for modeling workflows and use semantic service discovery and composition. As suggested by [24], BPEL can also be used for grid environments. Input data can often be

*Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland, (pczarnul@eti.pg.gda.pl) http://fox.eti.pg.gda.pl/∼pczarnul.

defined as values, files or data streams [19], as in Gridbus. Input data can drive workflows as in Kepler [1]. The input data from one or several preceding workflow tasks can be treated and processed in several ways [18].

3. Motivations. While this paper builds on the model presented in [14], the contribution of this paper is as follows:

1. assessment of the impact of input data partitioning of input data on the scalability for a workflow with parallel paths,
2. comparison of the theoretical model of processing in a workflow and real results,
3. assessment of the overhead of the parallelization using workflows with distributed services and a highly parallel solution using MPI,
4. integration of parallel processing by parallel workflow paths and parallelization within services using MPI.

4. Model of the Workflow Scheduling with Data Distribution. The model proposed by the author is based on the workflow model with service selection [13, 35] and is extended to consider data distribution for parallel paths of the workflow.

A directed graph $G(V, E)$ represents a workflow in which nodes V correspond to tasks while edges E represent task dependencies. At least one starting node with initial data and one termination node which terminates computations are distinguished. The starting nodes do not have predecessors. Each node should have a successor apart from the termination node. The model allows to define:

1. a sequence – a service assigned to the second task in a sequence successor is executed only after the service selected for the predecessor has completed (Figure 4.1a),
2. fork – services assigned to the tasks following the forked task are executed in parallel provided these were installed on separate processors (Figure 4.1b),
3. join – the service selected for the task to which other tasks are connected are executed only after each of the predecessors has finished (Figure 4.1b).

The following parameters are distinguished following [14]:

- $S_i = \{s_{i0}, s_{i1}, ..., s_{i(|S_i|-1)}\}$ – a set of services out of which one is selected to execute task t_i,
- c_{ij} – the cost of processing a unit of data by service s_{ij},
- N_{ij} – the node on which service s_{ij} was installed,
- sp_n – the speed of node n,
- P_{ij} – the provider of service s_{ij},
- d_{ij}^{in} and d_{ij}^{out} denote the sizes of the input and output data accepted and produced by service s_{ij}. These are linked with formula $d_{ij}^{out} = f_{t_i}(d_{ij}^{in})$ where f_{t_i} defines the size of output data for task t_i based on the input data size.
- 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}. d_{ij}^{out} can be sent and/or partitioned into input files of successors. In particular, all the data can be sent to all the successors or it can be partitioned into non-overlapping parts that will be distributed for parallel processing by the following tasks.
- $t_{ij}^{exec}(d_{ij}^{in})$ – 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.
- $t^{workflow}$ – the workflow execution time i.e. $t^{workflow} = t_{termination}$ $not\exists_q(v_{termination}, v_q) \in E$.

Traditionally, several optimization goals can be considered such as minimization of $t^{workflow}$ with a bound on the total cost of selected services i.e. $\sum d_{ij}^{in} c_{ij} < B$ where B is the budget (problem MIN_T_C_BOUND) and $t^{workflow}$ is the time when the last service finishes. Problem MIN_TC is minimization of $v_{MIN\_TC} = \alpha t^{workflow} + \sum d_{ij}^{in} c_{ij}$. $\alpha > 0$.

It should be noted that some descriptions of services with respect to e.g. execution times or reliability may not be accurate. An adaptive technique for learning of service reliability were proposed by the author and his

team in [15].

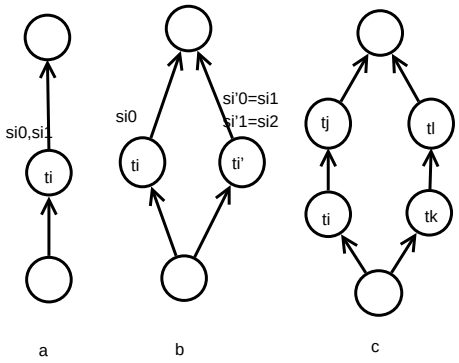

Fig. 4.1: Selection and Parallelization in the Model

4.1. Selection of Solutions and Data Parallelization. The proposed formulation allows to model several issues typical of workflows composed of independent services offered and managed by various users:

1. *selection of alternative solutions* (services) $s_{i0}, ..., s_{i(|S_i|-1)}$ to particular task t_i. This is possible thanks to the well known service selection concept i.e. selecting only one service out of the ones defined for the given task (one of several services is selected for task t_i in Figure 4.1a).
2. *data parallelism* – partitioning of input data for parallelization of computations on this data. This can be achieved in the model in several configurations, depending on the needs:
   (a) parallelization of task t_i where input data is divided and possibly processed by services s_{i0} and s_{i1} or s_{i0} and s_{i2} in parallel. Note that s_{i1} and s_{i2} are modeled as exclusive services which combines selection with parallelization. Task t_i is split into t_i and t'_i which are functionally equivalent (Figure 4.1b) and initial services are assigned to these two tasks.
   (b) parallelization of a complex task (composed of more than one task) by possibly execution of various sub-solutions in parallel. Input data is divided into flows so that $t_i \rightarrow t_j$ and $t_k \rightarrow t_l$ are executed in parallel. It can be that $t_i \neq t_k$ and $t_j \neq t_l$ (Figure 4.1c).

Assuming the user has access to services $s_{i0}, ..., s_{i(|S_i|-1)}$ for task t_i, parallelization where the algorithm determines partitioning of data between possibly all services, can be modeled as splitting services for many tasks (in fact functionally equivalent) as in Figure 4.2 where $\forall_{i,k} s^{'n}_{i0} = s_{in}$.

4.2. Real and Integer Data Sizes. Additional constraints on partitioning of data could be set e.g. $d_i \in Z$ $d_{ijkl} \in Z$ for integer values suitable for partitioning of a set of e.g. pictures of same size for processing in parallel. This makes the problem harder to solve but the algorithms proposed and discussed by the author in Section 5 can handle this case.

4.3. Synchronization. In Figure 4.2 all services are synchronized on the following task t_j which means that even after $s^{'2}_{i0}$ has finished processing its portion of data, t_j will not start until all $s^{'n}_{i0}$ have finished. If there are tasks t_i and t_j which need to process the input data subsequently, the proposed model can allow pushing a portion of data to the following task even if the previous steps on the other portions of data have not completed yet. This can be accomplished as in Figure 4.3. Namely, as soon as t'_i has finished processing its portion of data, it can send results for processing by t'_j without waiting for completion of processing by e.g. t_i.

The question arises into how many virtual tasks processing of the given task should be split. This should cover the number of services which can process the given task. In such a case, potentially all services for the original task t_i and t_j could participate in parallel execution of the corresponding task with the possibility of pushing parts of data from task t_i to task t_j even when execution of other parts of t_i's data is still in progress. Furthermore, this could be set up to the potential number of partitions the initial data could be divided into to allow pushing smaller portions of data sooner.

In one workflow there may be both tasks with multiple services (for selection of one service for task) and virtual tasks with one service assigned to each of them meant for data parallelization. The former may be used

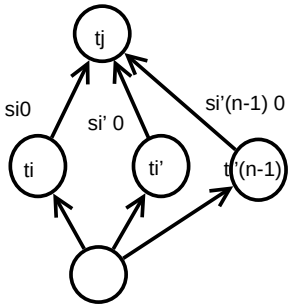

Fig. 4.2: Data Parallelization

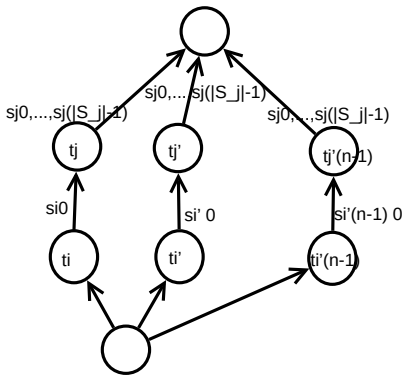

Fig. 4.3: Data Parallelization without Synchronization between Tasks

when there is a task to be executed on a portion of the input data where it is not possible to split data among services or it would be too costly compared to the execution by one service (e.g. a conversion of an image to another format). Data partitioning can be used when it is possible and beneficial to split the data and there are enough services to execute in parallel.

5. Algorithms. If the graph contains only tasks with one service for each, the problem becomes how to distribute data with possibly cost constraints. In this case and assuming that data can be divided into chunks of any size, fast linear programming [28] can be used where variables denote the sizes of data processed by particular services. This also assumes that the execution times for the serviceare linear functions of input data size which does not have to be the case. For service selection, the literature suggests introducing integer variables [3, 4, 36] that denote which service is selected for the particular task.

The author has proposed and implemented three different algorithms to solve the problem [14]:

1. genetic algorithm (GA) – in this case a solution is represented by a chromosome. It encodes both selection of particular services for the tasks and also order of execution for pairs of parallel tasks in the graph for which the order is not determined by the workflow graph.
2. mixed genetic algorithm and linear programming (MGALP) – in this case services are assigned to the tasks by a genetic algorithm and data distribution for the given schedule is determined by linear programming [28],
3. mixed integer linear programming (MILP) – similarly to the known approaches [36, 3, 4], integer variables mean which service is selected for a particular task. Additionally though, real or integer variables denote sizes of data flowing from task to task.

6. Management, Optimization and Execution of Workflow Applications in BeesyCluster. The author has created a workflow management environment for modeling, scheduling and execution of workflow applications, both for the proposed model [14] and also for dynamic selection of services at runtime [13].

The environment uses BeesyCluster as a middleware to access distributed services and is available at `https://lab527.eti.pg.gda.pl:10030/ek/Main` at Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland.

BeesyCluster allows its users to access accounts on distributed resources through SSH and compile and run applications, among others (Figure 6.1). Furthermore, such applications can be published as BeesyCluster services to which access may be granted to other users. In particular, costs of running services can be defined. BeesyCluster users have virtual purses from which can pay for services published by others. Similarly, users may earn from others running their own services.

Furthermore, the workflow management environment allows:

1. definition of a workflow using a graphical interface (Figure 6.2),
2. selection of services and defining data flows between tasks [10, 12],
3. actual execution of the workflow in a distributed environment [10, 12].

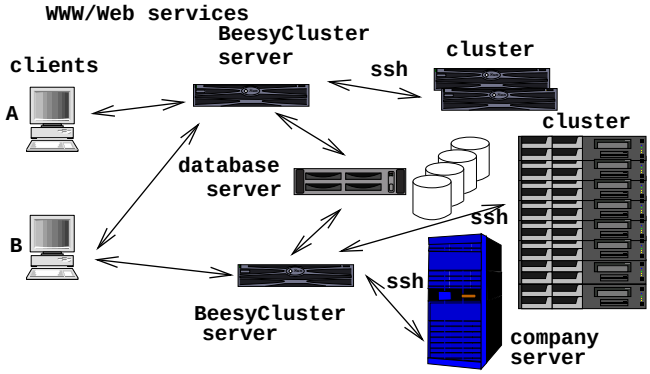

Fig. 6.1: BeesyCluster Architecture

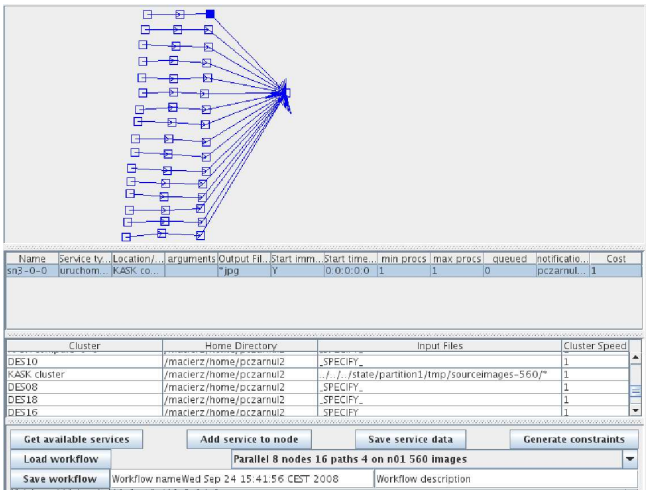

Fig. 6.2: Workflow Editor in BeesyCluster

6.1. Compute Intensive Scientific Workflow: Adaptive Quadrature Distributed Integration. As an example for demonstration of the goals defined in Section 3 a compute intensive application was implemented by the author and analyzed. This application computes a numerical integrate of a given function on the given range with a certain accuracy using the adaptive quadrature integration algorithm presented in [30].

6.1.1. Testbed Environment and Services. The testbed environment consists of 16 nodes installed at the Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology. Each node along with services installed on them was accessed by BeesyCluster using SSH. The relative speeds of 8 nodes were 1 while the speeds of the other nodes were 0.575 for the application. Each node uses its local disks to store input and output data. As an example function $f(x) = sin(x)cos(sin(x)) + \frac{1}{x^2}$ was integrated on given ranges. The following services were developed:

partition *a b rpf filecount* – partitions range $[a, b]$ recursively into the given number of files *filecount* and subranges per file *rpf* so that the execution time per subrange is approximately similar. At the given step, for each subrange [a,b] 10 points $c_0, ..., c_9$ are selected so that $a < c_i < b$. Then the maximum of areas of the triangles formed by $(a, f(a)), (c_i, f(c_i)), (b, f(b))$ is selected for each subrange $[a, b]$. Out of all available subranges the one with the largest area is selected and partitioned into two subranges $[a, \frac{a+b}{2}]$, $[\frac{a+b}{2}, b]$. The procedure is repeated until the desired number of subranges is generated.

integration – an application written in C that integrates all subranges from input files in its directory and produces a single output file with extension `out` with the total integrate of the given subranges. The al-

gorithm divides the given subrange recursively like described above until the maximum area of triangles is smaller than 0.000000001 in which case the integrate is approximated by a sum of rectangles.

adder – adds the values supplied in files with `out` extension supplied in its directory

Figure 6.3 presents the workflow in which the initial range $[a, b]$ is partitioned into $filecount$ files each of which contains rpf subranges. Then the initial files with subranges are sent to separate nodes for parallel processing. It is important to note that filecount must be sufficiently large to distribute files with input subranges among parallel tasks for integration especially if services assigned to them run on nodes of different speeds. Similarly, rpf can be set larger than 1 to further improve load balance since it increases the number of initial subranges and makes their computation times more even. The model assumes that the processing time of each input file is the same.

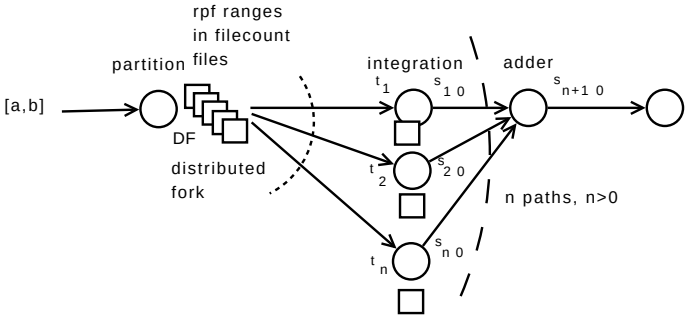

Fig. 6.3: A Parallel Integration Workflow

6.1.2. Workflow Configurations and Results. The algorithm is able to adapt data distribution to various speeds of processing nodes and also does it under budget constraints. The following configurations were tested (optimization goals are noted):

MIN_TC: granularity test (Figure 6.3) – for up to 8 nodes it is best to select 1 file per node as the nodes have same speeds but vary the number of subranges. On the other hand, for 12 and 16 processors we must increase the number of files to let the algorithm balance work between processors of different speeds. Figure 6.4 presents execution times for these settings for range [1,1000]. Setting the number of files or subranges too large causes too much overhead related to opening files, loop overhead with no further gain in load balance. Finally, best settings were selected for following tests.

MIN_TC:parallel (Figure 6.3) – execution times and corresponding speed-ups are shown in Figures 6.5 and 6.7 for three different ranges of $[1, 250]$, $[1, 1000]$ and $[1, 2000]$. It can be seen that the algorithm achieves very good speed-ups for larger ranges even though there are various node speeds. For a small run the overhead of the solution including preparation of directories on the nodes, copying of data take their toll. In fact the speed-up for range [1,1000] is slightly better than for range [1,2000] presumably because of some temporary load on one of the nodes during the runs. It also suggests that this is the upper limit on the speed-up for this configuration as increasing the size does not increase the speed-up.

MIN_T_C_BOUND: parallel under budget constraints (Figure 6.6) – we define the costs of services depending on the clusters they are installed on and the time of day according to Table 6.1. Three logical clusters are distinguished with 8, 4 and 4 nodes (with different costs) respectively. Figures 6.8 and 6.9 show the impact of decreasing the budget on the execution time of the workflow for two configurations: clusters 1 and 2 (total of 12 nodes), clusters 1, 2 and 3 (total of 16 nodes) for day and night. Cluster costs are the same as for the digital photography workflow since the same clusters were used. However, for this particular integration code clusters 2 and 3 (the speed of each node is 0.575) are slower than cluster 1 (the speed of each node is 1). The budget is varied from the minimum budget allowing full parallelization, then 0.9 and 0.8 of this value. Clearly tightening the budget increases execution times as the algorithm does not allow to pass input data to more expensive services causing the use of a smaller number of cheaper but slower paths.

6.2. Comparison of Theoretical Model and Simulation Results. The image processing workflow analyzed and tested in [14] is a good example for comparing theoretical execution times of the adopted model and

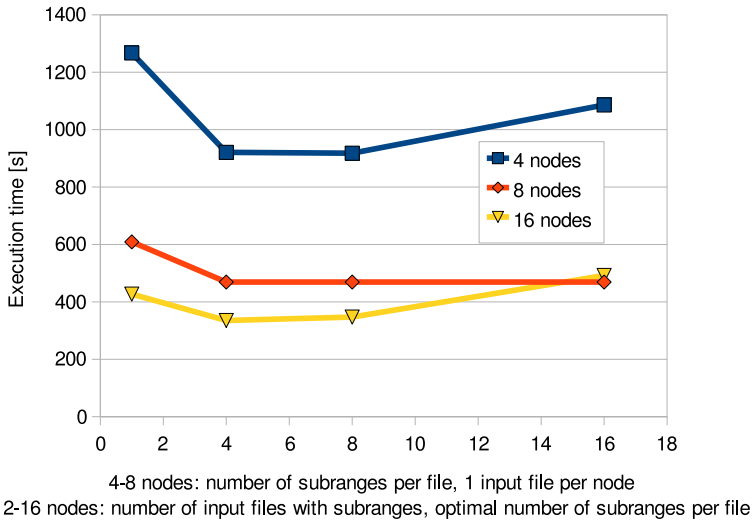

Fig. 6.4: Adaptive Integration on range [1,1000]: Execution Times [s] vs Number of Subranges per File (1-8 nodes) or Files Per Node (16 nodes)

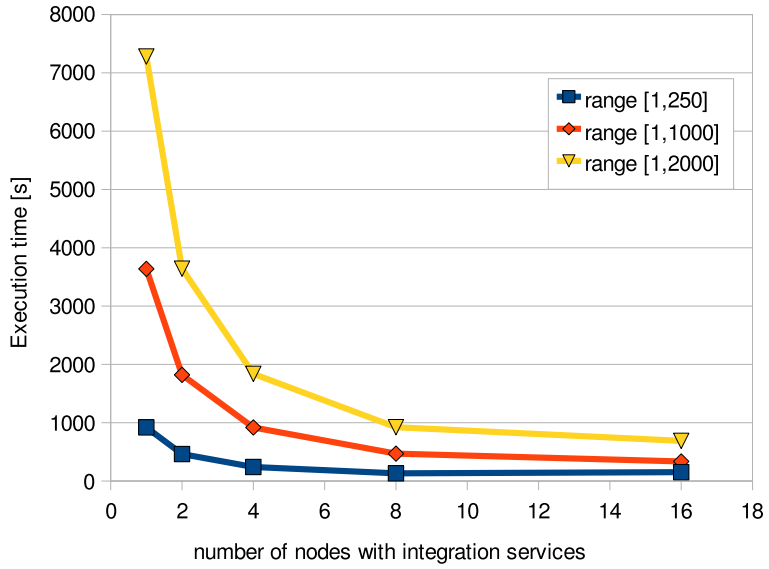

Fig. 6.5: Adaptive Integration: Execution Times [s]

			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

Table 6.1: Services and Cost per Processor Second for Testbed Clusters

simulation results as it involves transfers of large portions of data of different sizes for the three configurations tested. The simulation results shown in [14] were compared to the theoretical model in Figure 6.10. The parameters of the model in this case were computed as follows. The execution time of each path of the workflow

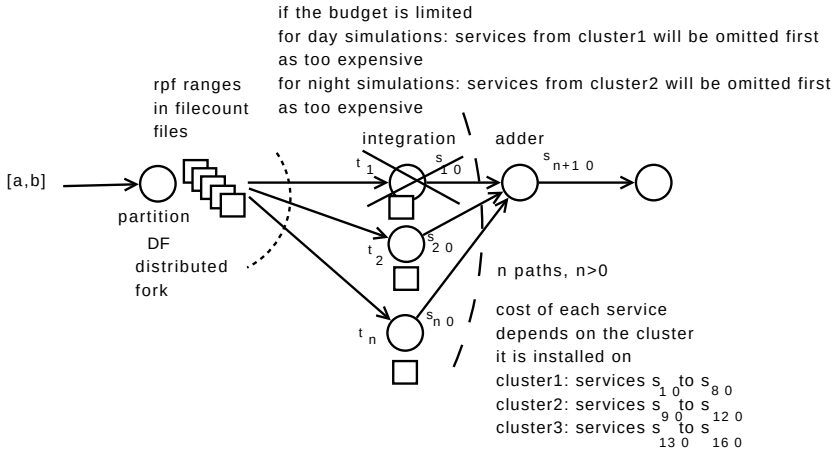

Fig. 6.6: A Parallel Integration Workflow with Budget Constraints

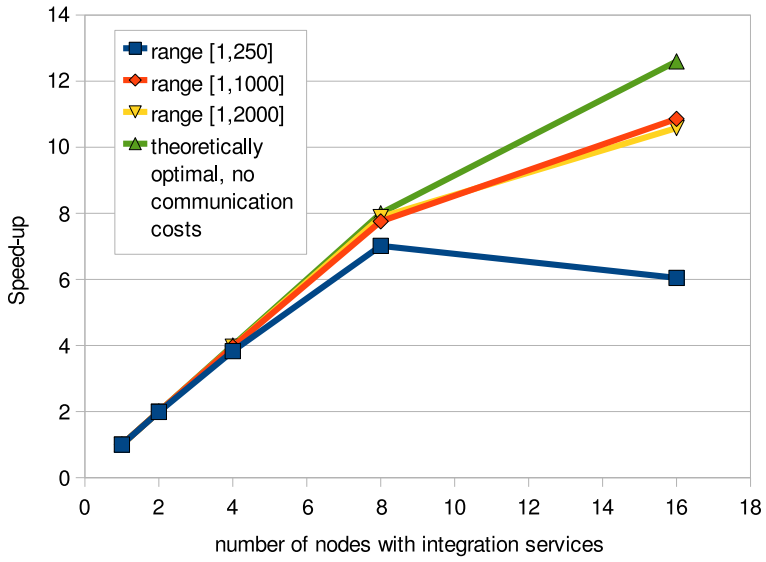

Fig. 6.7: Adaptive Integration: Speed-up

including web album generation as the last workflow task is as follows: execution time $= a + \frac{d}{p}(c + e) + df$ where d is the number of initial images, p the number of paths, a is an accumulated constant from startup times, overhead for preparation of directories for each workflow node, e and c correspond to computation and communication times and df corresponds to the execution time of Web album generation.

The execution times are known as are data sizes d. The execution times of the Web generation phase were read from the system logs and thus it was possible to determine f. The execution time of web album generation is always the same as requires the same number of images. Then linear regression was used to determine a and $c + e$.

The simulation results are very close to the model (Figure 6.10) being slightly too optimistic regarding performance. The source node is a bottleneck in copying data to several following nodes.

6.3. Integration of BeesyCluster and MPI and Overhead of the Solution. It is possible to assess the overhead of the workflow support in BeesyCluster mechanism by comparing execution times to highly dedicated parallel solutions run in the same environment. A C+MPI based implementation can be regarded as a lower bound on the execution time of the service-based BeesyCluster workflow. Obviously neither the

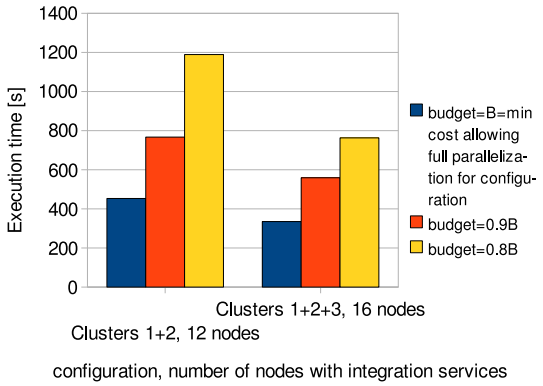

Fig. 6.8: Adaptive Integration: Execution Time [s] under Cost Constraints: Day

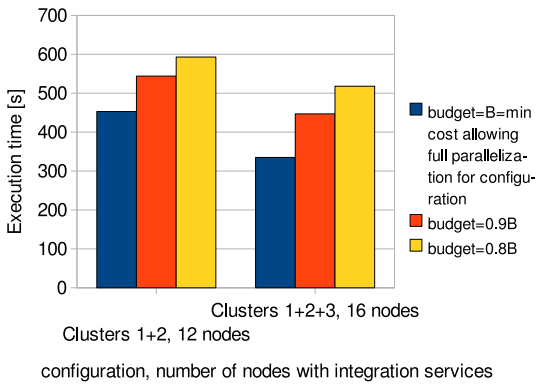

Fig. 6.9: Adaptive Integration: Execution Time [s] under Cost Constraints: Night

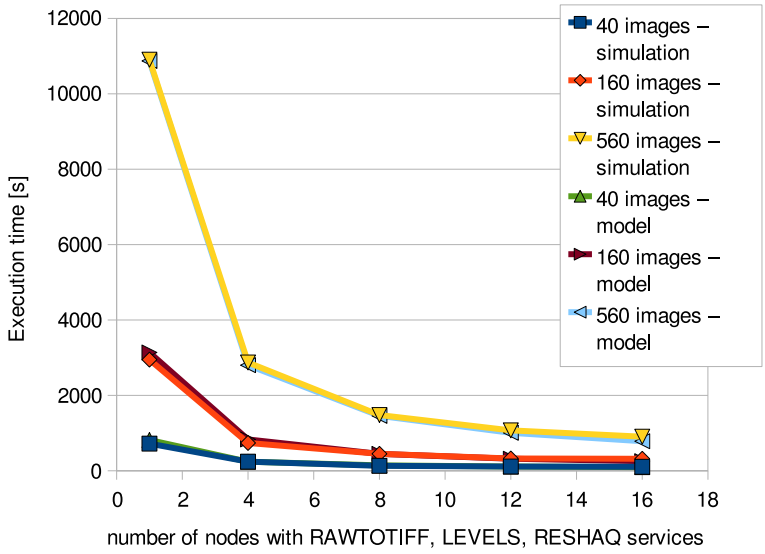

Fig. 6.10: Comparison of Theoretical Model and Simulation Results

standard MPI nor grid-enabled versions such as MPICH-G2 [20], PACX-MPI [21] or BC-MPI [11] offer easy integration of services with performance and cost aware optimization.

Distributed integration was used as an example in the following configurations:

MPI on 2 clusters. Two MPI environments each with 8 nodes were configured. The partitioning application divides the initial range into a predefined number of subranges and saves them to the number of files equal to the number of distinct MPI environments used (1 up to 8 nodes, 2 for 16 nodes in this case) considering relative speeds of the latter. Subsequently, MPI applications launched on the clusters divide the ranges between processes using MPI and compute results for their parts.

workflow in BeesyCluster with services using MPI. Two services were used each of which was implemented as a parallel C+MPI application. For 1-8 nodes one service implemented by one MPI application using from 1 to 8 nodes was used. For 16 nodes, two BeesyCluster services each of which ran an MPI application on 8 nodes just like in the previous example. It allows to assess the overhead of the BeesyCluster layer. Initial data was prepared as in the previous example.

workflow in BeesyCluster. The configuration considered in Paragraph 6.1 was used where the number of services is equal to the number of nodes. Each service is a sequential application.

Figure 6.11 presents a comparison of speed-ups between these three solutions. The reference run used 1

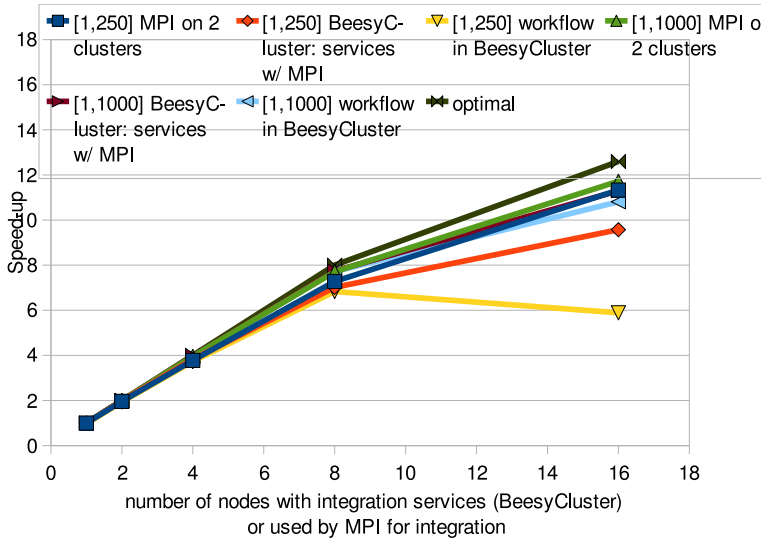

Fig. 6.11: Performance Comparison of BeesyCluster Workflow and MPI: Speed-up

processor.

It can be seen that the MPI implementation is not far below the theoretical best result while the BeesyCluster based implementations introduce slightly more overhead. It is not significant for the [1,1000] range but is visible for short runs for range [1,250] and 16 nodes. This is because the overhead introduced by the Java EE implementation is significant compared to the short execution time in this case. Nevertheless, for longer runs the performance of the workflow solution is very good. Secondly, as expected the overhead of the workflow with MPI-based services is much smaller than for a larger number of sequential services. The overhead of the workflow with MPI-based services compared to an MPI-only implementation on 16 nodes for range [1,1000] was 13 seconds. This encourages to use workflows with MPI-based services as it allows very easy integration of services with scheduling and data partitioning incorporating performance and costs which is not available in MPI implementations.

7. Summary. The paper formulated a problem on service selection and scheduling with data distribution encountered in the integration of distributed services offered by various providers. Both the model and the algorithm were implemented in BeesyCluster allowing to consume distributed services from various providers. Additionally, an easy-to-use workflow editor and an execution engine were implemented. As an example, a distributed numerical integration was constructed as a workflow application were prepared and executed a distributed service-based environment. The solution achieves good speed-ups. It is able to minimize the execution time and keep the total cost of selected services below a threshold. The paper demonstrates the impact of input data partitioning on the scalability of the approach. Secondly, simulation results are compared to the theoretical model which confirms its correctness. Finally, the overhead of the implementation is analyzed compared to a pure parallel MPI implementation. Results for an integrated workflow/MPI solution are also presented.

Acknowledgment. Research partially sponsored by research grant N N516 383534 "Strategies for management of information services in distributed environments".

REFERENCES

[1] *Kepler user manual*, May 2008.
[2] *CUDA Programming Guide 3.1.* `http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf`, June 2010.
[3] 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.

[4] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, *Dynamic web service composition in meteor-s*, technical report, LSDIS Lab, Computer Science Dept., UGA, May 2004.
[5] S. A. Ahson and M. Ilyas, eds., *Cloud Computing and Software Services: Theory and Techniques*, CRC Press, 2011. ISBN 978-1-4398-0315-8.
[6] R. Buyya, ed., *High Performance Cluster Computing, Programming and Applications*, Prentice Hall, 1999.
[7] 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.
[8] ———, *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.
[9] J. Cardoso, A. Sheth, and J. Miller, *Workflow quality of service*, tech. report, LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA 30602, USA, March 2002.
[10] P. Czarnul, *Integration of compute-intensive tasks into scientific workflows in beesycluster*, in Computational Science – ICCS 2006, vol. 3993 of LNCS, Springer, 2006, pp. 944–947.
[11] ———, *Bc-mpi: Running an mpi application on multiple clusters with beesycluster connectivity*, in Proceedings of Parallel Processing and Applied Mathematics 2007 Conference,, Springer Verlag, May 2008. Lecture Notes in Computer Science, LNCS 4967.
[12] ———, *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.
[13] P. Czarnul, *Modeling, run-time optimization and execution of distributed workflow applications in the JEE-based BeesyCluster environment*, The Journal of Supercomputing, (2010), pp. 1–26. 10.1007/s11227-010-0499-7, http://dx.doi.org/10.1007/s11227-010-0499-7.
[14] P. Czarnul, *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, 2010, pp. 629–636. Wisla, Poland.
[15] P. Czarnul, M. Matuszek, M. Wjcik, and K. Zalewski, *BeesyBees — agent-based, adaptive & learning workflow execution module for BeesyCluster*, in Faculty of ETI Annals, Information Technologies vol. 18, Gdansk, Poland, 2010.
[16] 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`.
[17] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, in Open Grid Service Infrastructure WG, June 22 2002. Global Grid Forum, `http://www.globus.org/research/papers/ogsa.pdf`.
[18] 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, 22 (2008), pp. 347–360.
[19] Gridbus Project, *Workflow language (xwfl2.0)*. `gridbus.cs.mu.oz.au/workflow/2.0beta/ docs/xwfl2.pdf`.
[20] N. Karonis and B. Toonen, *Mpich-g2 – a grid-enabled implementation of the mpi v1.1 standard*. `http://www.hpclab.niu.edu/mpi/`, Department of Computer Science at Northern Illinois University and Mathematics and Computer Science Division (MCS) at Argonne National Laboratory.
[21] R. Keller and M. Mller, *The Grid-Computing library PACX-MPI: Extending MPI for Computational Grids*. `www.hlrs.de/organization/amt/projects/pacx-mpi/`.
[22] Laboratory of Parallel and Distributed Systems, MTA SZTAKI, Hungary, *Parallel Grid Runtime and Application Development Environment, User's Manual, ver. 8.4.2*.
[23] 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).
[24] 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., 22 (2008), pp. 238–249.
[25] 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.
[26] Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface Standard*, July 1997.
[27] 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), LNCS, Prague, Czech Republic, September 2003, pp. 826–835.
[28] M. M. Syslo, N. Deo, and J. S. Kowalik, *Discrete Optimization Algorithms*, Prentice-Hall, 1983.
[29] M. Wieczorek, A. Hoheisel, and R. Prodan, *Towards a general model of the multi-criteria workflow scheduling on the grid*, Future Generation Comp. Syst., 25 (2009), pp. 237–256.
[30] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, 1999.
[31] 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, 2007, IEEE Computer Society, pp. 322–329.
[32] J. Yu and R. Buyya, *A taxonomy of workflow management systems for grid computing*, Journal of Grid Computing, 3 (2005), pp. 171–200.
[33] 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.
[34] ———, *Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms*, Scientific

Programming Journal, (2006). IOS Press, Amsterdam.
[35] 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.
[36] 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.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011