A Workflow Engine for Computing Clouds

Daniel Franz, Jie Tao, Holger Marten, and Achim Streit
Steinbuch Center for Computing
Karlsruhe Institute of Technology, Germany
daniel2712@gmx.de, {jie.tao, holger.marten, achim.streit}@kit.edu

***Abstract*—This work developed a workflow engine that enables the execution of workflows on existing Cloud platforms. The workflow engine automatically delivers the computation of each individual task to the selected Cloud and transfers the input/output data across different platforms. Additionally, it predicts the execution time and payment of the tasks, helping users select the best Cloud services with respect to the performance vs. cost tradeoff.**

***Keywords*-Cloud Computing, Workflow Management System, Grid Computing**

I. Introduction

Since Amazon published its Elastic Compute Cloud (EC2) [1] and Simple Storage Service (S3) [2] in 2008, Cloud Computing became a hot topic in both industrial and academic areas. There exist different definitions of Cloud Computing, including our earlier contribution [3]. Recently, the National Institute of Standards and Technology (NIST) provides a specific definition: Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [4].

Cloud computing distinguishes itself from other computing paradigms in the following aspects:

- Utility computing model: Users obtain and employ computing platforms in computing Clouds as easily as they access a traditional public utility (such as electricity, water, natural gas, or telephone network).
- On-demand service provisioning: Computing Clouds provide resources and services for users on demand. Users can customize and personalize their computing environments later on, for example, software installation, network configuration, as users usually own administrative privileges.
- QoS guaranteed offer: The computing environments provided by computing Clouds can guarantee QoS for users, e.g., hardware performance. The computing Cloud renders QoS in general by processing Service Level Agreement (SLA) with users.

As a result of these advantages, Cloud Computing is gaining more and more customers. Currently established Cloud infrastructures mainly deliver three kinds of services: Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service. IaaS targets on an on-demand provision of the computational resources. The commercial computing Cloud Amazon EC2 and its non-commercial implementation Eucalyptus [5] are well-known examples of IaaS-featured Cloud platforms. SaaS allows the consumers to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface [4]. An example of SaaS is Web-based email. PaaS targets on an entire platform including the hardware and the application development environment. Google App Engine [6] and Microsoft Azure [7] are examples of PaaS-featured Clouds.

The goal of this work is to combine different Clouds to run a user-defined service workflow. A workflow is a methodology that splits the computation of a complex problem into several tasks. A well-known scenario is to run scientific experiments on the Grid [8], where an entire computation is partitioned and distributed over several computing nodes with a result of being able to process large data sets. This scenario can also occur on the Cloud when scientific applications move to them. Furthermore, there are other scenarios on the Cloud, where users require the workflow support. For example, users may compose the services provided by different Clouds for an overall goal.

We developed an execution engine for workflow management on Clouds. In difference to Grid workflow implementations that target on a unified interface [9], a Cloud workflow system has to cope with different interfaces and features of individual Clouds. In order to enable the combination of single workflow tasks running on various Clouds, we implemented a Cloud abstraction and designed mechanisms for inter-Cloud data transfer. We also established a prediction model to estimate the execution time and cost of the individual tasks on different Cloud nodes, therefore helping users achieve maximum performance at lowest payment.

The remainder of the paper is organized as following. Section II describes the related work. Section III analyzes the requirement on a Cloud workflow framework and presents the designed software architecture. Section IV gives the details of an initial prototypical implementation, followed by the evaluation results in Section V. The paper concludes in Section VI with a brief summary and several future directions.

II. Related Work

The concept of resource sharing in Cloud Computing is similar to Grid Computing. Cloud Computing allows on-demand resource creation and easy access to resources, while Grid Computing developed standards and provides various utilities. A detailed comparison of these two computing paradigms can be found in [10]. One utility implemented on the Grid is the workflow management system. Production Grids, such as WLCG [11], TeraGrid [12], and EGEE [13], commonly support the execution of scientific workflows on the underlying resources. There are also various implementations of workflow engines on the Grid. Examples are ASKALON [14], Unicore [15], Kepler [16], GridAnt [17], Pegasus [18], and GridFlow [19]. An overview of these workflow systems is presented in [20].

The research work on workflow management systems on the Cloud has been started. A well-known project is the Cloudbus Toolkit [21] that defines a complete architecture for creating market-oriented Clouds. A workflow engine is also mentioned in the designed architecture and described in detail in [22]. The authors analyzed the requirement and changes needed to be incorporated when moving scientific workflows to Clouds. They also described the visions and inherent difficulties when a workflow involves various Cloud services. The work presented in this paper aims at a prototypical implementation of a workflow engine that executes a workflow composed of different Cloud services, because such a tool is currently still not available. The goal is to simply provide a new functionality rather than to investigate a comprehensive solution.

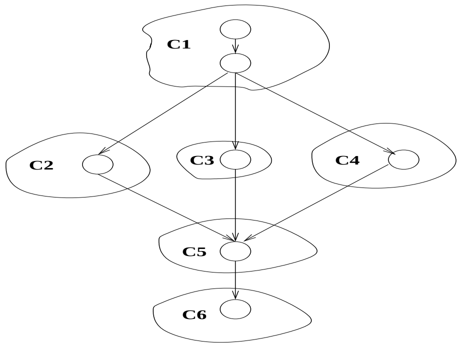

Figure 1. A sample execution scenario of Cloud workflows.

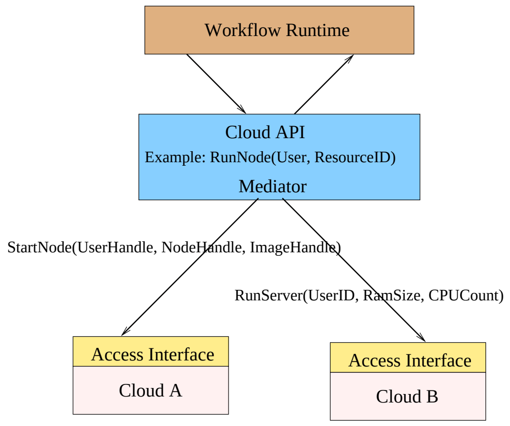
Workflow Runtime
Cloud API
Example: RunNode(User, ResourceID)
Mediator
StartNode(UserHandle, NodeHandle, ImageHandle)
RunServer(UserID, RamSize, CPUCount)
Access Interface
Cloud A
Access Interface
Cloud B

Figure 2. Software architecture of the workflow engine.

III. Architecture Design

Grid Computing has been investigated for more than a dozen of years and established standards. Cloud Computing, in contrast, is a novel technology and has not been standardized. The specific feature of each Cloud brings additional challenges to implementing a workflow engine on Clouds.

A. Design Challenges

Grid workflows may be executed in several resource centers but the involved resources are contained in a single Grid infrastructure and hence can be accessed with the same interface. Cloud workflows, however, run usually on different Clouds.

Figure 1 shows a sample scenario of running workflows on Clouds. While some tasks may be executed on the same Cloud, e.g., Cloud C1, some others may run on different Cloud platforms. The data are transferred from one Cloud to another in order to deliver the output of one task to other tasks. Unfortunately, different Clouds use also different data format. Furthermore, existing Clouds have their own access interfaces. A standard, called Open Cloud Computing Interface (OCCI) [23], has been proposed but no implementation is currently available. To link the services of different Clouds, an abstraction layer is required for providing an identical view with the data and interfaces of the target Cloud infrastructures.

Additionally, the service price varies across Cloud providers. Cloud users usually expect an optimal performance vs. cost tradeoff: i.e., acquiring the best service with the lowest payment. While increasing Cloud infrastructures are emerging, there may be several choices to run a workflow task. A prediction model, which is capable of estimating the performance and cost of an execution on a specific Cloud, can help users select the best Cloud for their tasks.

Based on the aforementioned observations, we designed a software architecture for the proposed Cloud workflow engine and defined a performance-cost model. The following two subsections give some details.

B. Software Architecture

Figure 2 demonstrates the software architecture of the proposed workflow engine for Cloud Computing. An important component in the architecture is the Cloud abstraction layer, shown in the middle of the figure. The task of this layer is to implement a unified API for accessing different Clouds. The runtime environment of the workflow engine uses this

API to run the tasks in a workflow.

The abstraction layer defines common functions for Cloud activities. It also contains a mediator that translates the functions in the unified API to concrete calls to the underlying Cloud platforms. For example, the function RunNode() is provided for running a virtual machine instance on any IaaS-featured Cloud. During the runtime the mediator replaces the function by a Cloud specific one, in this example, either StartNode for Cloud A or RunServer for Cloud B. It also maps the function parameters in the functions of the unified API to the functions of the APIs of individual Clouds. Furthermore, the mediator handles the authentication/security issues.

C. Prediction Model

Cloud users not only take care of the execution performance but pay more attention to the payment for using resources on the Clouds. As an initial design, we bring the two most important metrics, application execution time and the cost, into the prediction model. Workflows in this work are defined as: A workflow is comprised of several tasks, each is combined with an application/software that is either executed on an IaaS-Cloud or hosted as a Web service on a SaaS/PaaS-Cloud.

The execution time of a workflow (EoW in short) can be calculated with the following mathematical form:

$$EoW = EoT_1 + DT_1 + EoT_2 + DT_2 + ... + EoT_n$$

where EoT_i is the execution time of task i and DT_i is the time for transferring data from T_i to T_{i+1}. Note that we ignore the time to start a service on the Cloud as well as data transfers from and back to the customer environment.

The execution time of a single task depends on the features of the host machine on which the task is running. Roughly, it can be presented with:

$$EoT = f(S_{comp}, F_{cpu}, S_{mem}, S_{I/O})$$

where the parameters are size of the computation, frequency of CPU, size of memory and cache, and size of input/output data. For parallel applications, an additional parameter, the communication speed, has to be considered.

The price of a service on a Cloud is usually determined by the node type and the location of the resource. Each Cloud provider maintains a price table, where concrete payment (in US$ per hour) is depicted. Based on this table, we calculate the cost of a workflow task with:

$$CoT = f(EoT, \$/h)$$

The cost of executing a workflow is then calculated with:

$$CoW = CoT_1 + CoT_2 + ... + CoT_n$$

The functions for computing the execution time of a task can be designed differently with a tradeoff between complexity and accuracy. We implemented a simple model, which is detailed in the following section.

IV. Prototypical Implementation

Our initial implementation of a Cloud workflow management system focused on the following components:

- Cloud abstraction
- Runtime execution environment
- Prediction model

A. Cloud Abstraction

To run a workflow on diverse Clouds, an abstraction layer is required for the purpose of hiding the different access interface each Cloud presents to the users. We use jClouds [24] as the base of this work. jClouds provides a framework for transferring programs and their data to an IaaS-Cloud and then starting an instance to execute the program on the Cloud. The current release of jCloud can connect several IaaS-Clouds including Amazon EC2.

jClouds defines an API for accessing the underlying IaaS platforms. For SaaS/PaaS-featured Clouds, however, there exists currently no implementation for an abstraction layer. Our main task in extending jClouds is to develop an S+P abstraction that interacts with SaaS-featured and PaaS-featured Clouds.

The S+P abstraction contains two kinds of functions, GET and POST, for transferring data and service requests. Their input and output are defined in XML documents. This is identical to all Clouds. Each Cloud, however, requires specific input and output formats as well as different parameters for service requests. Our solution is to use XSL Transformation (XSLT) [25] to map the input and output of the service functions to the required data format and service parameters.

XSLT is a part of the Extensible Stylesheet Language (XSL) family and often adopted to transform XML documents. An XSLT file describes templates for matching the source document. In the transformation process, XSLT uses XPath, an interface for accessing XML, to navigate through the source document and thereby to extract information or to combine/reorganize the separate information elements. For this work an XSLT document is introduced for some data formats, like SOAP. For others, such as binary and JSON (JavaScript Object Notation), a data transformation is not needed.

The process of invoking a SaaS or PaaS service with the developed S+P abstract contains the following steps:

- Processing the input data of the service request.
- Constructing a URL for the service. Information about Cookies, SOAP actions and other parameters, is contained in the head of the protocol (HTTP), while the content of the protocol defines the request.
- A service request is sent to the aforementioned URL, together with the data.
- The results of the service are downloaded as raw data. For the data formats like SOAP, where the results are

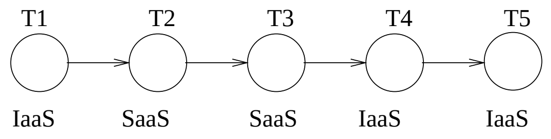

Figure 3. A simple Cloud workflow.

coded, an XSLT document is defined to extract the useful information.

B. Workflow Execution

In order to allow an easier understanding of the tasks for a Cloud workflow execution engine, we take a simple workflow as an example. Figure 3 demonstrates the sample workflow consisting of five tasks, T1 to T5, which are combined through a respective data flow. A task can be a program or an available Web service on a SaaS or PaaS Cloud. For the former, the program is executed on an IaaS Cloud, while for the latter the Cloud provides resources for running the software. The workflow and its tasks are defined by the user in an XML file. .

The workflow execution engine is responsible for running each task on the selected Cloud, transferring the result of one task to its successor, and downloading the final results to the user. The first job is performed within a single Cloud and contains the following steps, which are all covered by the Cloud abstraction described above:

- Transferring data (Program or service parameters) to the target Cloud.
- Executing the program on an IaaS Cloud or invoking the Web service on the SaaS or PaaS Cloud. In the case of IaaS, a virtual machine instance has to be started and some scripts are executed for configuration and program installation.
- Extracting the results out of the Cloud.

Another task of the workflow runtime engine is to deliver the output of one task to the next task as input. This involves an inter-Cloud communication. We implemented mechanisms for the following data transfer:

- IaaS to SaaS/PaaS: We use SSH to transfer data from the IaaS node to the local host and then use HTTP to deliver the data further to the SaaS/PaaS request;
- SaaS/PaaS to SaaS/PaaS: Data are extracted from the HTTP stream, stored temporally on the host, and then applied to the next HTTP request;
- SaaS/PaaS to IaaS: Locally storing the data, which are again extracted from an HTTP stream, and then transferring them to the IaaS node via SSH;
- IaaS to IaaS: We transfer the data directly from one IaaS node to the other that is potentially located on a different Cloud. This is an optimization for removing the overhead caused by an intermediate storage.

Finally, the result of the entire execution is downloaded to the user or stored on the last Cloud.

C. Performance & Cost Prediction

The proposed prediction model, as described in the previous section, involves several hardware parameters that can be only acquired at the runtime by accessing the Cloud resources. For the prototypical implementation, we developed a simple model without using the runtime resource information of the underlying infrastructures.

Our model is based on the execution history of similar tasks, which are tasks executing the same program. The execution history is stored in a user database, which contains the following main data structures:

- node_class: describes a computing node with node ID, node name, Cloud name, payment cycle, and startup time.
- execution: describes an execution of a task on a node with several attributes including program name, node_class, size of I/O, and execution time.
- node_price: gives the per-cycle-price of the computing nodes.
- node_location: gives the country and continent the node is located.

For each task in a new user-defined workflow, the potential execution time is calculated for all registered Clouds and their associated computing nodes. The payment is then calculated according to the price published by the Cloud providers. The first five {Cloud, node} pairs with the best performance vs. cost tradeoff are shown to the users to help them select the optimal target platforms.

We use the following algorithm to predict the execution time of a new task presented with $t_{(p,s)}$, where the first attribute is the program to be executed and s is the size of the input data.

First, the average execution time of the program on a node n_s is calculated with

$$t_{(p,n_s)} = \frac{\sum_{i=1}^{n} t_{i(p,n_s,s_i)}}{n}$$

where $t_{i(p,n_s,s_i)}$ is the time measured with the recorded ith execution of program p on node n_s with a data size of s_i. Here, $t_{(p,n_s)}$ is associated with the average data size $s_{(p,n_s)}$, which is calculated in a similar way. The execution time of the new task $t_{(p,s)}$ can then be estimated with

$$t_{(p,s)} = \frac{s}{s_{(p,n_s)}} \cdot t_{(p,n_s)} \cdot W_{data}$$

We introduce a weight variable W_{data} to represent the influence degree of the input size on the execution time.

Table I
EXPERIMENTAL RESULTS WITH THE 3D RENDER WORKFLOW (85 CAMERA POSITIONS).

Task	Node	Execution time			Performance vs. Cost
		Measured	Predicted	Difference (%)	
3dscenetopictures	m1.small	145	138	-4.8	12.4
	c1.medium	56	52	-7.1	19.03
	m1.large	48	42	-12.5	17,97
picturetovideo	m1.small	59	48	-18.6	5.01
	c1.medium	47	37	-21.2	15.97
	m1.large	44	36	-18.2	14.96

Table II
EXPERIMENTAL RESULTS WITH THE WORKFLOW OF SYNCHRONIZING A FOUR MINUTES VIDEO.

Task	Node	Execution time			Performance vs. Cost
		Measured	Predicted	Difference (%)	
videototext	m1.small	665	688	3.4	168.04
	c1.medium	341	355	4.1	116.3
	m1.large	257	271	5.4	87.2
translatejatoen		45	40	-11.1	0
texttospeech	m1.small	26	22	-15.4	1.87
	c1.medium	22	20	-9.1	7.47
	m1.large	19	17	-10.5	6.46
jointovideo	m1.small	89	104	16.8	7.6
	c1.medium	87	94	8.04	29.6
	m1.large	97	75	-12.4	33.02

V. EVALUATION RESULTS

To evaluate the developed framework, several workflows were tested. In this section, we present the results with two examples. The first workflow processes 3D scenes with a result of creating a video. The second workflow performs film synchronization whereby to translate the spoken text from Japanese to English.

The first workflow contains two main tasks, *3dscene-topictures* (the raytracer) and *picturetovideo*. The raytracer acquires a scene file and a camera file as input and splits the scene into single pictures based on the position defined in the camera file. The single pictures are then processed by the second task to produce a continuous video. We apply the Tachyon [26] raytracer for the first task, which needs an MPI cluster on an IaaS Cloud because the software is parallelized with MPI. To combine the pictures to a video, the program FFmpeg [27] is applied. We run this task on a single IaaS node. Hence, the first workflow involves only IaaSs.

The second workflow is comprised of four components: the language identifier (task *videototext*), a translator (task *translatejatoen*), the text synthesizer (task *texttospeech*), and the task *jointovideo*. The language identifier acquires a video file as input and outputs its text in Japanese. The output is then delivered to the language translator, where an English text is produced. In the following, the text synthesizer converts the text to speech, which is combined with the video via the last task of the workflow. We apply the language identifier Julius [28] to process the audio that is extracted from the video by FFmpeg. In order to speed up the process, an audio is first partitioned and the partitions are then processed in parallel. Hence, an MPI cluster is required for this task. For language translation, the translation service of Google is applied. In order to model a SaaS/PaaS to SaaS/PaaS data transfer and to verify our Cloud abstraction, the Japanese text is first translated to German and then to English. The task *texttospeech* is implemented using the speech synthesizer eSpeak [29]. Finally, the aforementioned FFmpeg program combines the audio with the video.

For the experiments we requested an account on EC2. The test results are shown in Table I and Table II for each workflow. The tables show the execution time of tasks of a single workflow on different nodes of EC2. In the case of Google, the Web service is executed on a Google machine, which cannot be specified by the user.

The execution time of a task is presented with the measured time and the predicted one, where the former was acquired at runtime and the latter was calculated using the developed prediction model. It can be seen that the accuracy of our model varies between the tasks, where the value with the second workflow is relative better. For the 3D render, the model underestimates the execution time in most cases, while an alternating behavior can be seen with the second workflow. Altogether, we achieved the best case with a difference of 3.4% between the real execution time and the predicted one, while the worse case shows a value of -21.2%. The difference is caused by the fact that the time for executing a program can vary significantly from one execution to the other, even though the executions are performed successively. This indicates that a more accurate model is required for a better prediction, which shall be our future work.

The values in the last column of the tables are calculated by multiplying the real execution time by the payment. It is expected that both the execution time and the payment are low. Hence, we use the values in the last column to represent the performance vs. cost tradeoff, where a lower value indicates a better behavior. Observing Table I it can be seen that the nodes m1.small have a better behavior. This may be associated with the concrete tasks, which do not demand a high computation capacity. With larger programs, e.g., the task *videototext* in the second workflow, a node with higher capacity, m1.large in this case, behaves better. However, the best choice is to use the free services provided by some Clouds, such as the translation service on Google.

VI. Conclusions

This paper described a workflow engine, which are designed and implemented for Cloud Computing. To enable the execution of a service workflow we developed a Cloud abstraction that mediates between different Cloud platforms. We implemented a runtime engine to execute the single tasks in the workflow and transfer data among them. Additionally, a prediction model was designed to estimate the execution time of the tasks on different Cloud nodes. Currently we implemented a simple model that will be improved in the next step of this work. Furthermore, we plan to develop a search engine that automatically detects Cloud services for a user-specified task. A graphical interface is also planned to allow the user to define the workflows in a more intuitive way. In addition, the workflow engine will be extended to handle the exception/errors of the Cloud services.

References

[1] “Amazon Elastic Compute Cloud,” [Online], June 2011, http://aws.amazon.com/ec2/.

[2] “Simple Storage Service,” [Online], June 2011, http://aws.amazon.com/s3/.

[3] L. Wang, M. Kunze, and J. Tao, “Performance evaluation of virtual machine-based Grid workflow system,” *Concurrency and Computation: Practice & Experience*, vol. 20, pp. 1759–1771, October 2008.

[4] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” [Online], January 2011, http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf.

[5] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus Open-source Cloud-computing System,” in *Proceedings of Cloud Computing and Its Applications*, October 2008. [Online]. Available: http://eucalyptus.cs.ucsb.edu/wiki/Presentations

[6] “Google App Engine,” [Online], June 2011, http://code.google.com/appengine/.

[7] “Windows Azure Platform,” [Online], June 2011, http://www.microsoft.com/windowsazure/.

[8] B. Asvija, K. V. Shamjith, R. Sridharan, and S. Chattopadhyay, “Provisioning the MM5 Meteorological Model as Grid Scientific Workflow,” in *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems*, 2010, pp. 310–314.

[9] G. Fox and D. Gannon, “Special Issue: Workflow in Grid Systems,” *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1009–1019, 2006.

[10] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared,” in *Grid Computing Environments Workshop, 2008. GCE’08*, 2008, pp. 1–10.

[11] WLCG, “Worldwide LHC Computing Grid,” [Online], June 2001, http://lcg.web.cern.ch/lcg/.

[12] P. H. Beckman, “Building the TeraGrid,” *Philosophical transactions - Royal Society. Mathematical, physical and engineering sciences*, vol. 363, no. 1833, pp. 1715–1728, 2005.

[13] EGEE, “Enabling Grids for E-sciencE,” [Online], June 2011, project homepage: http://www.eu-egee.org/.

[14] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Jr, and H. L. Truong, “ASKALON: a tool set for cluster and Grid computing,” *Concurrency and Computation: Practice & Experience*, vol. 17, pp. 143–169, February 2005.

[15] M. Riedel, D. Mallmann, and A. Streit, “Enhancing Scientific Workflows with Secure Shell Functionality in UNICORE Grids,” in *Proceedings of the IEEE International Conference on e-Science and Grid Computing*. IEEE Computer Society Press, December 2005, pp. 132–139.

[16] D. Barseghian, I. Altintas, M. B. Jones, D. Crawl, N. Potter, J. Gallagher, P. Cornillon, M. Schildhauer, E. T. Borer, E. W. Seabloom, and P. R. Hosseini, “Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis,” *Ecological Informatics*, vol. 5, pp. 42–50, 2010.

[17] G. von Laszewski, K. Amin, M. Hategan, N. J. Z. S. Hampton, and A. Rossi, “GridAnt: A Client-Controllable Grid Workflow System,” in *37th Hawaii International Conference on System Science*. IEEE CS Press, January 2004.

[18] S., D. Karastoyanova, and E. Deelman, “Bridging the Gap between Business and Scientific Workflows: Humans in the Loop of Scientific Workflows,” in *IEEE International Conference on eScience*, 2010, pp. 206–213.

[19] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, “GridFlow:Workflow Management for Grid Computing,” in *Proceedings of the International Symposium on Cluster Computing and the Grid*, May 2003, pp. 198–205.

[20] 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.

[21] R. Buyya, S. Pandey, and C. Vecchiola, “Cloudbus Toolkit for Market-Oriented Cloud Computing,” in *Proceeding of the 1st International Conference on Cloud Computing*, December 2009, pp. 978–642.

[22] S. Pandey, D. Karunamoorthy, and R. Buyya, *Cloud Computing: Principles and Paradigms*. Wiley Press, February 2011, ch. 12, pp. 321–344.

[23] “Open Cloud Computing Interface,” [Online], June 2001, http://occi-wg.org/.

[24] “jclouds,” [Online], June 2001, http://www.jclouds.org/.

[25] M. Kay, *XSLT 2.0 Programmer’s Reference*. Wrox, 3 edition, August 2004.

[26] J. Stone, “An Efficient Library for Parallel Ray Tracing and Animation,” In Intel Supercomputer Users Group Proceedings, Tech. Rep., 1995.

[27] “FFmpeg,” [Online], June 2001, http://www.ffmpeg.org/.

[28] “Open-Source Large Vocabulary CSR Engine Julius,” [Online], June 2001, http://julius.sourceforge.jp/en_index.php.

[29] “eSpeak text to speech,” [Online], June 2001, http://espeak.sourceforge.net/.