An Implementation Approach for Inter-Cloud Service Combination

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

***Abstract*—Increasing cloud platforms have been developed in the recent time and are provisioning different services to customers. The existence of different cloud provides offers the users an opportunity of combining different cloud services to run their scientific workflows with lower cost and higher performance. In this work we developed a framework to allow combining individual services of different clouds to solve complex problems with efficiency. The framework contains a workflow management system that processes users workflow descriptions and starts the related services automatically on the target clouds. A data management component takes care of the data exchange between the underlying clouds. Moreover, a prediction model computes the potential cost and performance of running a workflow on existing clouds, giving users help in selecting the execution target for better performance/cost effect.**

***Keywords*-Service Composition, Cloud Computing, Workflow Engine, Cloud Interoperability**

I. INTRODUCTION

Cloud Computing is an emerging technology for providing IT capacities as services. Increasing number of customers is using the resources, such as computational power, software, storage, and network, offered by various cloud providers for their daily computation requirement. However, inter-cloud computing is still a novel topic. We developed a workflow framework to enable the interaction across different cloud infrastructures [1].

The service concept has been proposed for several decades. However, the term of Cloud Computing was known first in 2008 as Amazon published its Elastic Compute Cloud (EC2) [2] and Simple Storage Service (S3) [3]. Cloud Computing became thereafter a hot topic in both industrial and academic areas. There exist different definitions of Cloud Computing, including our earlier contribution [4]. 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 [5].

A specific feature of Cloud Computing is that computation and storage resources are provided as services. In this way, applications/software can be executed or maintained on the cloud without the necessity of operating an own local infrastructure. Such a computing model significantly reduces the cost for resource and software management, which is clearly an attractive benefit for small business companies and research groups. In addition to cost-efficiency, Cloud Computing shows other advantages such as on-demand resource provision, supporting legacy codes, service reliability, easy management, and so on. Therefore, more and more cloud infrastructures are established and increasing numbers of users are joining the cloud world.

The fact that different cloud providers exist brings a chance to users: combining different cloud services to efficiently solve a complex problem. However, it also introduces difficulties for deciding to choose which cloud in case of multiple identical services. Depending on the size and requirement of tasks as well as the hardware configuration of clouds the tasks may perform better on some platforms than on others.

The goal of this work is to design and prototypically implement a management system that combines different cloud services to run a user-defined workflow with an additional functionality of helping users select the cloud providers.

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 [6], 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 to benefit from their diverse functionality and to achieve a better performance/cost ratio.

Currently, it is possible to run a workflow on the cloud [7]. However, running workflows is still limited to a single cloud platform. The partitions (called tasks) of a workflow, however, may have different behavior on different clouds, indicating a requirement of running workflows across several cloud platforms.

To enable this execution mode we developed a workflow management system with a workflow engine, a data management component, and a mathematical model for performance and cost prediction. In difference to Grid workflow

implementations that target on a unified interface [8], 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 layer to deliver common access interfaces. The developed framework was tested with several sample workflows.

The remainder of the paper is organized as following. Section II gives a short introduction of Cloud Computing, together with some related work. Section III analyzes the requirement on a cloud workflow framework and presents the designed software architecture. Section IV describes our initial prototypical implementation of the workflow framework in detail. Section V shows the experimental results with sample workflows. The paper concludes in Section VI with a short summary and several future directions.

II. Background and Related Work

Cloud Computing introduces a new computing paradigm. This paradigm uses virtualization as its fundamental technology. A traditional computer system runs applications directly on the complete physical machine. Using virtualization, applications are executed on virtual machines (VM), with each VM typically running a single application and a different operating system.

Cloud Computing distinguishes itself from other computing paradigms, like Grid Computing, Global Computing, and Internet Computing, 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 like CPU speed, I/O bandwidth and memory size. The computing cloud renders QoS in general by processing Service Level Agreement (SLA) with users.
- Autonomous system: the computing cloud is an autonomous system and it is managed transparently to users. Hardware, software and data inside clouds can be automatically reconfigured, orchestrated and consolidated to present a single platform image, finally rendered to the users.
- Security: the host system monitors the communication to the VMs, restricting the number of successful attacks. Even if an attack is effective, the attack could only compromise one VM, while the other applications and operating systems maintain a secure state.
- Availability: VMs can be easily migrated increasing the system's fault tolerance and availability.

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 [9] 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 [5]. 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 [10] and Microsoft Azure [11] are examples of PaaS-featured clouds.

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. Table I shows several major features of both computing paradigms. A detailed comparison between Grid and Cloud Computing can be found in [12].

One utility implemented on the Grid is the workflow management system. Production Grids, such as WLCG [13], TeraGrid [14], and EGEE [15], 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 [16], Unicore [17], Kepler [18], GridAnt [19], Pegasus [20], and GridFlow [21]. An overview of these workflow systems is presented in [22].

The research work on workflow management systems on the cloud has been started. A well-known project is the Cloudbus Toolkit [23] 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 [24]. 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.

In the sense of service combination, there are several efforts on automated processes [25]–[27]. For service composition on the cloud research issues have been discussed [28], [29] and practices have been conducted as well [30]. In addition, there are also activities on preparing and executing workflows [31] on such composite services. Recently, scientists proposed the idea of "federated clouds" [32], which can effectively utilize several clouds to enhance the QoS.

Existing work on cloud workflow systems is either in the research phase or an implementation work within a single

Table I
CLOUD VS. GRID: A FUNDAMENTAL COMPARISON.

	Cloud Computing	Grid Computing
Objective	Provide desired computing platform via network enabled services	Resource sharing Job execution
Infrastructure	One or few data centers heterogeneous/homogeneous resources under central control Industry and business	Geographically distributed, heterogeneous resources, no central control, VO Research and academic organization
Middleware	Proprietary, several reference implementations exist, e.g., Amazon	Well developed, maintained, and documented
Application	Suited for general applications	Special application domains like High Energy Physics
User interface	Easy to use/deploy, no complex user interface required	Difficult to use and to deploy Need new user interface, e.g., commands, APIs, SDKs, services
Business model	Commercial: pay-as-you-use	Publicly funded: use for free
Enabling technology	Virtualization, SaaS, Web 2.0, Web service, ...	HPC, Grid infrastructure, middleware
QoS	Possible	Little support
On-demand provisioning	Yes	No

cloud platform. The work presented in this paper aims at a prototypical implementation of a workflow engine that enables the execution of a workflow across different cloud platforms thereby using their individual services. Such a tool is currently still not available. Our goal is to simply provide a new functionality rather than to investigate a comprehensive solution. Therefore, we majorly focus on the design of an architecture and a prototypical implementation with basic functionalities. Our main contributions are the workflow engine that enables inter-cloud service combination and the proposal of a prediction model for estimating the execution time of tasks on different clouds.

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) [33], has been proposed and several implementations are currently available. However, this standard is only supported by a few cloud infrastructures. To link the services of different clouds, an abstraction layer is therefore required for providing an identical view with the data and interfaces of the target cloud infrastructures.

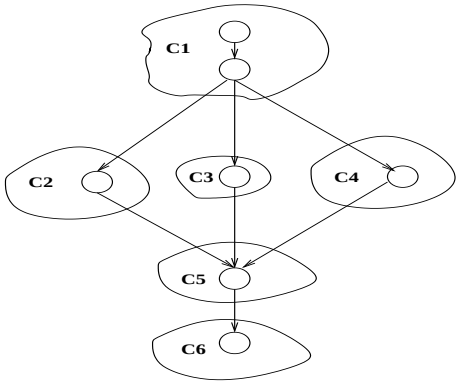

Figure 1. A sample execution scenario of cloud workflows.

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 number of cloud infrastructures is 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 an appropriate 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. It contains a

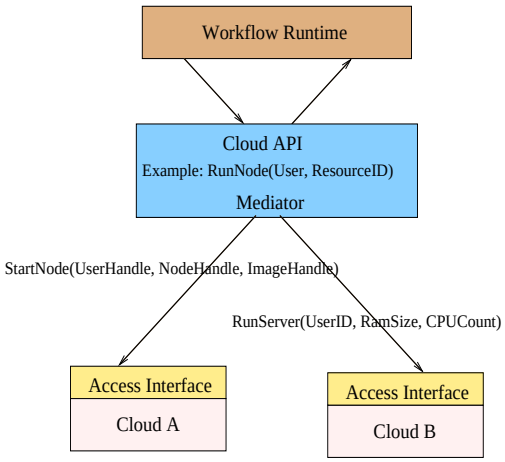

Figure 2. Software architecture of the workflow engine.

workflow runtime system, the cloud API, and the underlying cloud infrastructures. Users submit their workflows using a client side interface. Based on the description of tasks in a workflow, the execution time of the tasks on different clouds is predicted. Using this information users extend the workflow description by specifying a target cloud for each task. In the following, the workflow runtime handles the execution of tasks on the clouds and delivers the results to the users.

An important component in the architecture is the cloud abstraction layer, shown in the middle of Figure 2. 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:

$$EoW = EoT_1 + DT_1 + EoT_2 + DT_2 + ... + EoT_n \quad (1)$$

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:

$$EoT = f(S_{comp}, F_{cpu}, S_{mem}, S_{I/O}) \quad (2)$$

where the parameters are size of the computation, frequency of CPU, size of memory and cache, and size of the 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:

$$CoT = f(EoT, \$/h) \quad (3)$$

The cost of executing a workflow is then calculated as following:

$$CoW = CoT_1 + CoT_2 + ... + CoT_n \quad (4)$$

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 next section.

IV. Initial Implementation

Based on the designed architecture described above, we implemented a prototype of the cloud workflow framework. This initial implementation focused on the following components:

- Cloud abstraction
- Runtime execution environment and data management
- 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 [34] 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. There are other IaaS cloud

abstraction libraries, such as libcloud [35] and deltacloud [36]. We use jCloud as the base of this work due to its feature of supporting SSH and a number of cloud providers.

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) [37] 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 usually 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.

SaaS/PaaS services are started via HTTP-based protocols. A service request is sent to the service URL via GET/POST. Information like Cookies, SOAP actions and other parameters can be wrapped in the message head. The content of the call is either contained in the body of the HTTP message in case of POST calls or coded in the URL in case of GET calls.

The process of invoking a SaaS or PaaS service with the developed S+P abstraction 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 message, while the body of the message 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 coded, an XSLT document is defined to extract the useful information.

B. Workflow Execution and Inter-Cloud Data Transfer

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 eight tasks, T-a to T-f, 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 the following XML file:

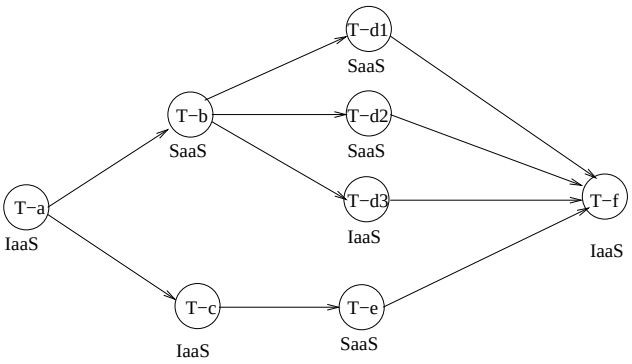

Figure 3. A simple cloud workflow.

```
<workflowdefinition>
  <task name="a">
    <input filetype="some/type" name="in" />
    <output filetype="some/type" name="oa" />
  </task>
  <task name="b">
    <input filetype="some/type" name="oa" />
    <output filetype="some/type" name="ob" />
  </task>
  <task name="c">
    <input filetype="some/type" name="oa" />
    <output filetype="some/type" name="oc" />
  </task>
  <task name="d1">
    <input filetype="some/type" name="ob" />
    <output filetype="some/type" name="od1" /
        >
  </task>
  <task name="d2">
    <input filetype="some/type" name="ob" />
    <output filetype="some/type" name="od2" /
        >
  </task>
  <task name="d3">
    <input filetype="some/type" name="ob" />
    <output filetype="some/type" name="od3" /
        >
  </task>
  <task name="e">
    <input filetype="some/type" name="oc" />
    <output filetype="some/type" name="oe" />
  </task>
  <task name="f">
    <input filetype="some/type" name="od1" />
    <input filetype="some/type" name="od2" />
    <input filetype="some/type" name="od3" />
    <input filetype="some/type" name="oe" />
    <output filetype="some/type" name="of" />
  </task>
```

```
  <workflow from="a" to="b" />
  <workflow from="a" to="c" />
  <workflow from="b" to="d1" />
  <workflow from="b" to="d2" />
  <workflow from="b" to="d3" />
  <workflow from="c" to="e" />
  <workflow from="d1" to="f" />
  <workflow from="d2" to="f" />
  <workflow from="d3" to="f" />
  <workflow from="e" to="f" />
</workflowdefinition>
```

The workflow definition mainly describes the input and output of each task and thereby also specifies the data flow between the tasks. In the concrete example the root element *workflowdefinition* is specified by the tasks to be executed, where each task is combined with an input node and an output node. The last task *f*, for example, has four input nodes describing the source of its input as shown in Figure 3. The last few lines of the XML document define the data flow between individual tasks, with help of the *workflow* element. The data flow from task a to task b and task c, from task b to task d1, task d2, and task d3, and so on.

The tasks, however, are not defined in the workflow document but described in a separate configuration file. An example is a task for extracting texts from a video. This task is used in a language translation workflow that will be introduced in the next section. The following XML file defines this task:

```
<task name="videototext" type="IaaS">
  <binary name="julius" parallel="
      sequential">
    <input type="video/flv" param="" />
    <output type="text/plain" param="" />
  </binary>
</task>
```

In the example, the IaaS task *videototext* is defined with a binary called *julius* that will be sequentially executed on an IaaS cloud. The type of its input and output is also specified in the XML file.

The primary work of the workflow execution engine is to interpret the workflow definition and starts each of the tasks on a cloud, based on the configuration XML files. In addition, the engine is also responsible for transferring the result of one task to its successor and downloading the final results to the user. The first task 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.

To deliver the output of one task to the next task as input, the workflow execution engine has to interact with both participating clouds and to handle the inter-cloud communication. We implemented mechanisms for the following data transfers:

- 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 defined as tasks executing the same program. The execution history is stored in a user database that contains the following main data structures:

- tasks: identifies each individual task with a unique ID and the associated program.
- I/O: defines the size of input and output files of tasks.
- 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.

We use an SQLite [38] database system to store the data structures. Figure 4 shows a screenshot of the SQLite database browser, where the data structures and the stored data are presented. The database browser allows us to simply modify the data items and to view the data. As examples, Figure 5 and 6 show the node and execution data collected during our tests. The browser depicts the data in the form of tables.

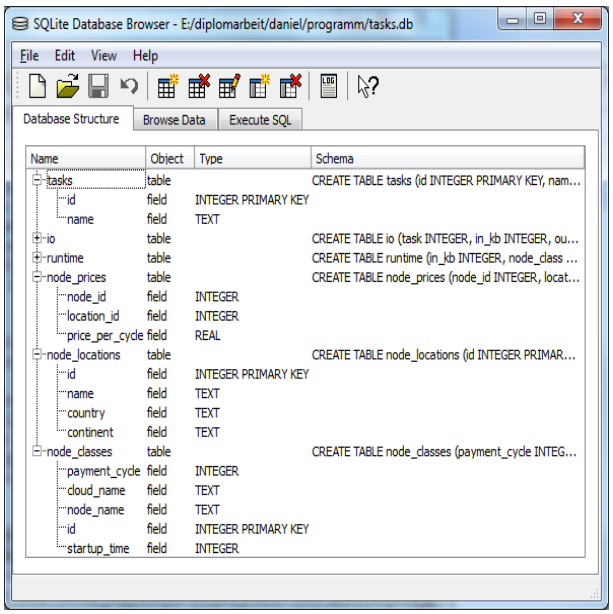

Figure 4. The data structures in an SQLite database.

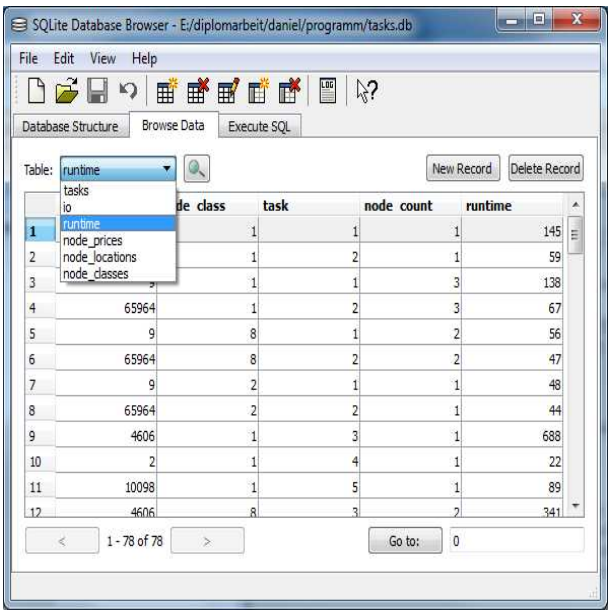

Figure 6. Task execution information stored in the SQLite database.

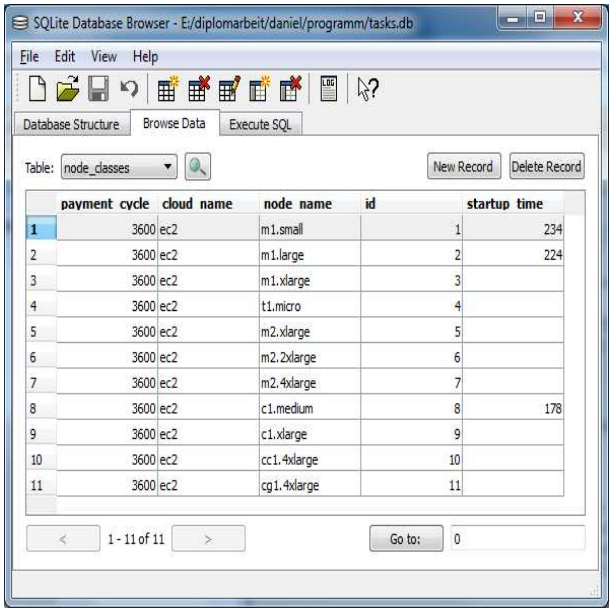

Figure 5. Node class information stored in the SQLite database.

The concrete table in Figure 5 lists all possible computing nodes on the Amazon EC2. EC2 uses a unit of one hour to calculate the cost of running instance leading to a payment cycle of 3600 (in seconds). We tested three different EC2 nodes and the startup time of these nodes can be seen in the last column of the table.

We also tested different tasks on various computing nodes. The table in Figure 6 depicts the execution information of the first twelve tests, with the data size of the task on the first column, the ID of the target computing node on the second, the task ID on the third, the number of nodes used for the test on the fourth, and the execution time on the last column. The sixth line, for example, shows that task 2 with an input data of 65964 bytes was executed on two EC2 "c1.medium" nodes (node ID 8) in 47 seconds.

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 the following equation:

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

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, and n is the number of executions. 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

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

Table II
AN EXAMPLE OF EXECUTION HISTORY.

Tests	Size of data (KB)	Execution time (seconds)
1	300	15
2	100	12
3	200	13
4	250	14
5	300	14

We introduce a weight variable W_{data} to represent the influence degree of the input size on the execution time. The value of this variable may vary from task to task and shall be choosen based on experimental results.

Now we show an example for a better understanding of the prediction model. The sample data is depicted in Table II, which contains five history executions on a computing node. The size of data is presented in Kbytes while the execution time in seconds.

According to the data in the table, we acquire the calculation results as:

$$t_{(p,n_s)} = \frac{15+12+13+14+14}{5} = 13.6$$

$$s_{(p,n_s)} = \frac{300+100+200+250+300}{5} = 230$$

The execution time of task $t_{(p,150)}$ on the related computing node is:

$$\frac{\frac{150}{230} \cdot 13.6}{0.7} = 12.67$$

where we assume an I/O influence of weight 0.7 to the execution of the program.

V. EXPERIMENTAL 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, depicted in Figure 7, contains two main tasks, *3dscenetopictures* (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 [39] 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 [40] is applied. We run this task on a single IaaS node. The workflow is defined in the following XML file:

```
<workflowdefinition>
  <task name="3dscenestopictures">
    <input filetype="*.dat" name="scene" />
    <input filetype="*.cam" name="camera" />
    <output filetype="image/ppm" name="images
        " />
  </task>
  <task name="picturestovideo">
    <input filetype="image/ppm" name="images"
        />
    <output filetype="video/avi" name="video"
        />
  </task>
  <workflow from="3dscenestopictures"
        to="picturestovideo" />
</workflowdefinition>
```

The second workflow, as shown in Figure 8, 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 [41] 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 to SaaS 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 [42]. Finally, the aforementioned FFmpeg program combines the audio with the video. The workflow definition is as following:

```
<workflowdefinition>
  <task name="videototext">
    <input filetype="video/flv" name="video"
        />
    <output filetype="text/plain" name="
        jatext" />
  </task>
  <task name="translatejatode">
    <input filetype="text/plain" name="jatext
        " />
    <output filetype="text/plain" name="
        detext" />
  </task>
  <task name="translatedetoen">
    <input filetype="text/plain" name="detext
        " />
    <output filetype="text/plain" name="
        entext" />
  </task>
  <task name="texttospeech">
    <input filetype="test/plain" name="entext
        " />
```

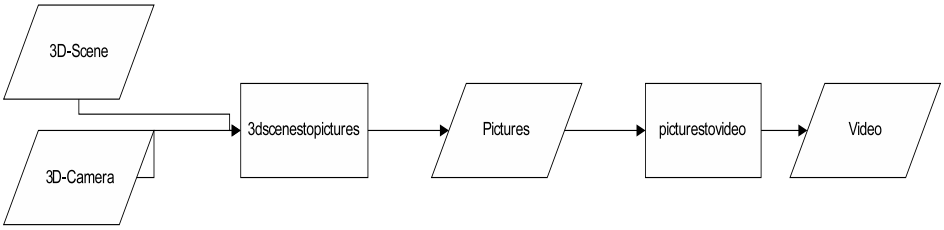

Figure 7. Data flow of the 3D-render workflow.

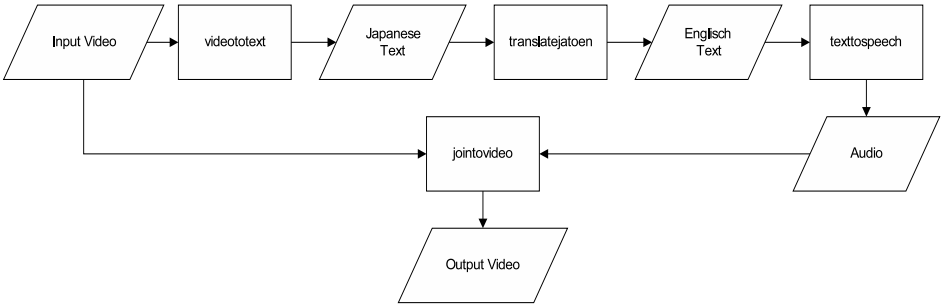

Figure 8. Data flow of the workflow film synchronization.

```
    <output filetype="audio/wave" name="audio
        " />
  </task>
  <task name="jointovideo">
    <input filetype="video/flv" name="video"
        />
    <input filetype="audio/wave" name="audio"
        />
    <output filetype="video/flv" name="
        resultvideo" />
  </task>
  <workflow from="videototext" to="
      translatejatoen" />
  <workflow from="translatejatode" to="
      translatedetoen" />
  <workflow from="translatedetoen" to="
      texttospeech" />
  <workflow from="texttospeech" to="
      jointovideo" />
</workflowdefinition>
```

For the experiments we requested an account on EC2. The test results are shown in Table III and Table IV 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 by measuring the duration of a task from submission to termination 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 III 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 and Future Work

As various cloud platforms are emerging, it is possible for users to involve several clouds to run a complex job,

Table III
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	12.40
	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 IV
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	116.30
	m1.large	257	271	5.4	87.20
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	7.60
	c1.medium	87	94	8.0	29.60
	m1.large	97	75	-12.4	33.02

for example, a workflow. For this functionality, users need a framework to manage the execution of single tasks on different clouds and to deliver the result of one task to another task that may run on a different infrastructure.

We designed and implemented such a workflow management system for cloud users. The main components of the framework includes a workflow engine for task execution, a cloud abstraction to enable the interoperability of different cloud platforms, a data management component that handles inter-cloud communications, and a prediction model for estimating the cost and performance of running the workflow tasks on different cloud nodes. Initial experiments on the prototypical implementation showed the functionality of the framework.

In the next step of this work we will improve the prediction model with involvement of the runtime information of the cloud platforms. We will also provide the users with a more friendly interface to describe their workflows. A resource broker is planed as well to serve as a mediator for users to detect the best cloud services.

REFERENCES

[1] D. Franz, J. Tao, H. Marten, and A. Streit, "A Workflow Engine for Computing Clouds," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011). ISBN: 978-1-61208-153-3*, Roma, Italy, September 2011.

[2] "Amazon Elastic Compute Cloud," [Online], http://aws.amazon.com/ec2/ (accessed: 2012-06-20).

[3] "Simple Storage Service," [Online], http://aws.amazon.com/s3/ (accessed: 2012-06-20).

[4] 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.

[5] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," [Online], http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf (accessed: 2012-06-20).

[6] 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.

[7] Y. Wei and M. B. Blake, "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities," *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, 2010.

[8] 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.

[9] 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, available: http://eucalyptus.cs.ucsb.edu/wiki/Presentations (accessed: 2012-06-20).

[10] "Google App Engine," [Online], http://code.google.com/appengine/ (accessed: 2012-06-20).

[11] "Windows Azure Platform," [Online], http://www.microsoft.com/windowsazure/ (accessed: 2012-06-20).

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Proceedings of the Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1–10.

[13] WLCG, "Worldwide LHC Computing Grid," [Online], http://lcg.web.cern.ch/lcg/ (accessed: 2012-06-20).

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

[15] EGEE, "Enabling Grids for E-sciencE," [Online], project homepage: http://www.eu-egee.org/ (accessed: 2012-06-20).

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] S. McIlraith, T. C. Son, and H. Zeng, "Semantic web services," *IEEE Intelligent Systems*, vol. 16, pp. 46–53, 2001.

[26] D. Fensel and C. Bussler, "The web service modeling framework WSMF," *Electronic Commerce Research and Applications*, vol. 1, pp. 113–117, 2002.

[27] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint driven Web service composition in METEOR-S," in *Proceedings of the IEEE International Conference on Service Computing*, 2004, pp. 23–30.

[28] F. Tao, L. Zhang, and Y. Hu, *Cloud Manufacturing: Development and Commerce Realization of MGrid, in Resource Service Management in Manufacturing Grid System*. John Wiley & Sons, Inc, 2012, ch. 15, doi: 10.1002/9781118288764.

[29] C.-H. Hsuand and H. Jin, "Services Composition and Virtualization Technologies," *IEEE Transactions on Services Computing*, vol. 4, no. 3, pp. 181–182, 2011.

[30] S. Wang, Q. Sun, H. Zou, and F. Yang, "Particle Swarm Optimization with Skyline Operator for Fast Cloud-based Web Service Composition," *Mobile Networks and Applications*, April 2012, online available: DOI: 10.1007/s11036-012-0373-3.

[31] "The OASIS committee, Web Services Business Process Execution Language (WS-BPEL)," [Online], http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (accessed: 2012-06-20).

[32] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A.Maraschini, P. Massonet, H. Munoz, and G. Toffetti, "Reservoir - When One Cloud Is Not Enough," *IEEE computer*, vol. 44, no. 3, pp. 45–51, March 2011.

[33] "Open Cloud Computing Interface," [Online], http://occi-wg.org/ (accessed: 2012-06-20).

[34] "jclouds," [Online], http://www.jclouds.org/ (accessed: 2012-06-20).

[35] "Apache Libcloud – a unified interface to the cloud," [Online], http://libcloud.apache.org/ (accessed: 2012-06-20).

[36] "Deltacloud – an API that abstracts the difference between clouds," [Online], http://incubator.apache.org/deltacloud/ (accessed: 2012-06-20).

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

[38] "SQLite," [Online], http://www.sqlite.org/ (accessed: 2012-06-20).

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

[40] "FFmpeg," [Online], http://www.ffmpeg.org/ (accessed: 2012-06-20).

[41] "Open-Source Large Vocabulary CSR Engine Julius," [Online], http://julius.sourceforge.jp/en_index.php (accessed: 2012-06-20).

[42] "eSpeak text to speech," [Online], http://espeak.sourceforge.net/ (accessed: 2012-06-20).