International Transactions on Systems Science and Applications
Volume 8 ▪ December 2012 ▪ pp. 57-69
sai: itssa.0008.2012.019

ISSN 1751-1461 (print)
ISSN 2051-5642 (online)

Realizing the Flexible Integration of Cloud Resources into Workflows

Michael Gerhards [1], Marco Jagodzinska [1], Volker Sander [1] and Adam Belloum [2]

[1] Faculty of Medical Engineering & Technomathematics
FH Aachen, University of Applied Sciences, Jülich, Germany
[2] Institute of Informatics, University of Amsterdam
Amsterdam, Netherlands
Email: {M.Gerhards; Jagodzinska; V.Sander}@fh-aachen.de; A.S.Z.Belloum@uva.nl

Abstract: The hosting of large on-premise computational resources is common practice. Cloud Computing offers a promising, alternative infrastructure for using scalable on demand off-premise resources. However, outsourcing whole applications is not a cost optimal solution in some scenarios, because the already existing on-premise resources are not considered. A flexible integration of additional resources from the cloud to compensate a shortage of suitable on-premise resources is a tradeoff between costs and efficiency. This article presents a realization of this approach in a cross-cloud environment using the concept of workflows. The realization is evaluated by the development of a prototype.

Keywords: dynamic resource allocation, cloud computing, cross-cloud workflows, service oriented architecture, workflow, workflow orchestration.

1. Introduction

The hosting of large on-premise computational resources by a computer center is common practice. However, it is not reasonable to provide a solution for all requested resource types in such a center. First of all, the initial purchase costs are very high. For small and medium enterprises (SME) it is nearly impossible to bear these costs alone. Even after a purchase the disadvantages still occur, mainly due to the operational costs. The hosting company is bound to the resources for many years, even if the computational power is no longer required. The old hardware does not benefit from new technologies which were developed in the meantime. If specific resources are used with unbalanced load, there is the risk of idling. An overprovisioning is also required for load peaks which also increase the costs. It is important to emphasize that the rare use of licensed software leads to opportunity costs.

Cloud computing offers a promising alternative infrastructure for using scalable on demand resources. Providers such as Amazon allow users to allocate virtualized computational resources. Of course, those providers allow for porting the full application. However, this might not be the most cost-effective solution, because the already existing on-premise resources are not considered. Therefore, for many scenarios it appears to be opportune to integrate cloud resources with easy-scale and dynamic provisioning into the local environment for the execution of computation intensive application parts whereas the other application parts are executed on local available general-purpose computational resources. An example is a highly parallelized application which could use a Graphics Processing Unit (GPU) in the cloud, while the remainder of the program is executed locally.

This article describes a realization of this approach in a cross-cloud environment using the concept of workflows. The realized ideas were taken from [1] and then extended to a prototype development. The motivation scenario can therefore be viewed as an example for a concept that applies to a much broader application domain.

2. Theoretical Development

The integration of cloud resources when a shortage of suitable on-premise resources occurs is a cost-effective solution. Figure 1 illustrates this approach for an example: some parallel parts of one application are executed off-premise, whereas the other parallel parts of the same application are executed on-premise. The parts which run on on-premise resources are called "local tasks" in the rest of this article whereas the parts which run on integrated cloud resources are called "cloud tasks". A condition for this approach is an application which is divisible into parts.

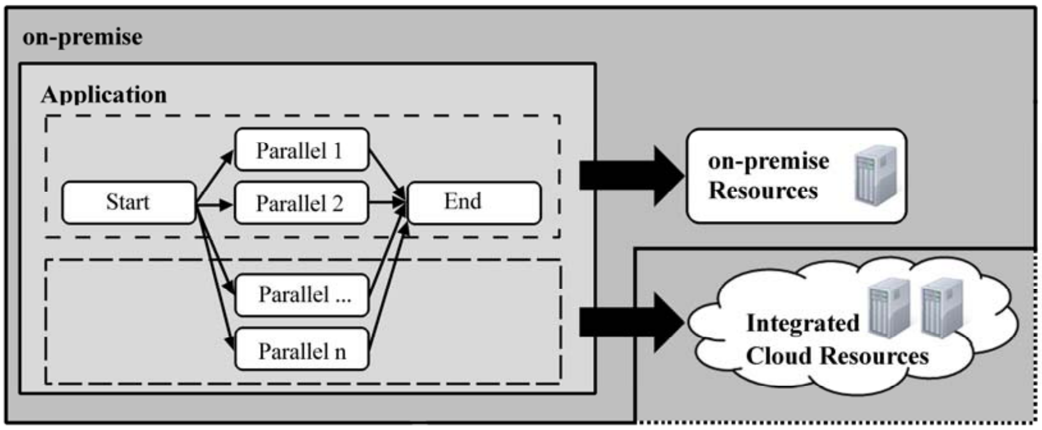

Figure 1. Approach of integrating cloud resources to compensate a shortage of on-premise resources for a single application.

2.1 Basic Concept of Workflows

Modeling a complex application as workflow supports its division into simpler individual parts that are executed as interacting tasks by a workflow management system. The reason why this article will focus on workflows to realize the above presented solution is that the division of applications into parts is natively supported. The basic ideas apply to a much broader application domain.

A workflow can be illustrated as directed graph composed of tasks as nodes and task dependencies as directed edges. Directed edges connect the predecessor task with its successor task. A task can only start its execution after its predecessor has finished its own execution. The application in Figure 1 can be seen as a workflow. A so-modeled workflow is called a workflow template that describes the behavior of a process; thus, it can be referred to as a general workflow definition. It is comparable with a program's source code and binary. Workflow templates are described using a specific workflow modeling language. Such templates are deployed, instantiated, and executed on a workflow management system [2] [3] that takes care of the individual tasks' progress and dependencies. Workflow instances follow the behavior of their assigned workflow template for a particular incident. It is comparable to a program's execution.

2.2 Workflows in Clouds

Workflow templates are modeled independently of specific resources because new resources can be established or existing ones can be omitted or blocked. The binding of workflow tasks to resources is done at runtime. The concept of considering only physical resources is gone in the cloud vision of infinite resources that just have to be activated. Theoretically, any number of cloud resources can be instanced on demand. To execute a workflow task in a cloud, the software must be deployed on a cloud instance and be accessible from the workflow management system via a remote procedure call (RPC) mechanism like a web service. Cloud computing per se does not impose any specific limitations with respect to the usage Application Programming Interface (API). "With the emerging of the latest cloud computing paradigm, the trend for distributed workflow systems is shifting to cloud computing based workflow systems [4]."

2.3 Cloud Service Models and Deployment Models

The National Institute of Standards and Technology (NIST) [5] distinguishes the three cloud service models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). SaaS providers often focus on standard applications like text processing or customer relation management and will not cover the whole variety of possible tasks. PaaS offers preconfigured deployment environments. IaaS offers only a rudimentary system which gives the most freedom to the developer but also forwards the maintenance of the system. The rest of the article will focus on IaaS resources to assume the minimum of requirements. This should not limit the generality since SaaS or PaaS offerings can be used instead with less effort.

NIST [5] also distinguishes four different deployment models: Private Cloud, Community Cloud, Public Cloud, and Hybrid Cloud. Since the example scenario assumes that the specific hardware is not used frequently, a Private Cloud providing such hardware is not feasible. However, the Private Cloud can be used to provide general on-premise resources. Sharing the resources of a Community Cloud is only possible if such a community exists but this cannot be assumed. Since the article focuses on integrating off-premise resources, a Public Cloud fulfills all requirements. A Hybrid Cloud is used in a combination of two or more previous deployment models.

This article presents how on-premise resources can be effectively combined with rented Public Cloud IaaS resources.

2.4 First Approach

A simple possibility to execute a workflow task in the cloud is the usage of a service-oriented approach by deploying the task software as web service on the cloud instance and binding the workflow task to this web service. Web services provide standardized uniform interfaces which support interoperability of heterogeneous systems. The data to be processed are typically pushed as parameter from the workflow task to the assigned web service. An alternative approach for passing larger data sets is that the web service pulls the requested data itself using a more efficient transmission protocol together with a onetime access ticket granted by the workflow management system.

IaaS and PaaS resources are frequently provided following a pay-per-time billing structure. The time is billed when the resources are available even when the resources are not used. Therefore any cloud instance should be terminated after each use to avoid unnecessary costs while the resource is idling. The consequence is that the cloud instance has to be started again before a re-use is possible. The dynamic allocation of cloud instances at runtime forms a service oriented architecture (SOA). The task execution idles during the integration of the cloud instance. Preconfigured machine images contain only the required software to speed up the instantiation. Each abstract cloud task uses its own machine image which is identified evaluating the abstract task's description in the workflow template. An alternative is the use of one basic machine image which is customized dynamically on startup time.

Procedure 1 summarizes all steps that have to be performed for each cloud task. The number indicates the order of performance. Dependend on the service model, some steps can be let skipped. The usage of on-premise resources or SaaS requires only Step 4, the software call, whereas the usage of IaaS requires all 5 steps. In the following Steps 1 to 3 are combined and named "integration".

2.5 Required Software for Prototype Development

The development of a prototype to realize this approach requires a workflow management system and a cloud system. The next subsections describe the used open-source software. During the prototype description in Section 3 these technologies are hidden in the background.

1. Start cloud resource
2. Chose & deploy machine image
3. Chose & deploy software } IaaS
4. **Call software / invoke service** } on-premise & SaaS } PaaS
5. Terminate cloud resource

Procedure 1. Steps which have to be performed for each cloud task.

2.5.1 Workflow Management System

The open-source flexible Business Process management (BPM) Suite jBPM of the JBoss community was used for the prototype. It provides an application server, a workflow engine to run workflows, an Eclipse Integrated Development Environment (IDE) with a Business Process Model and Notation 2.0 (BPMN 2.0) conform editor as plugin to model workflows, a data base to persist workflow runs, and a WS-HumanTask implementation to integrate human interactions into workflows in a standard conform way.

2.5.2 Cloud Virtualization System

OpenNebula [6] is an open-source software toolkit for cloud computing in recent version 3.8. It enables the creation of Private, Public, and Hybrid Clouds. The prototype uses OpenNebula for local tests to simulate a Public Cloud service provider without expenses on local resources. The fielding operating system is GNU/Linux. Binary OpenNebula packages are for example distributed with Ubuntu. Base for OpenNebula is a virtualization possibility like kernel based virtual machine (KVM). For a minimal sample OpenNebula only needs one computer, which is both administration unit and node. The processing power of the node is used by the virtual machines. The cloud resource instances to test the prototype run the free machine image Ubuntu Server 12.04 from the OpenNebula-marketplace as operating system so that configuring an own image is not necessary. OpenNebula provides commandline-tools for direct user interaction, but there are also high-level APIs for interacting from different programming languages like Ruby and Java, and the more low-level XML-RPC API.

Since the prototype should be extensible for other cloud service providers with their own APIs, the OpenNebula API is wrapped to provide only the rudimentary functionality and to be immune against interface changes from the perspective of the workflow. In future the proprietary prototype wrapper will be replaced using a cloud unification layer or a cloud agnostic API like the Open Cloud Computing Interface (OCCI) [7].

2.6 Example Workflow for On-Premise Execution

This section introduces an example workflow which is referenced in the rest of this article. The workflow is designed for execution on on-premise resources. The vision is to execute one task on integrated cloud resources. The procedure is described step by step in Section 3.

The example BPMN 2.0 workflow illustrated in Figure 2 consists of four script tasks, two control flow elements: split and merge, and the start as well as the end. The script tasks are Java dialect programs which can be customized. The control flow elements have a defined behavior. The arrows indicate the task dependencies which define the execution order of the tasks. The two control flow elements split and merge the two execution branches of the workflow. That means that the script tasks “Gauss” and “LuDecomposition” can be executed independent of each other in an arbitrary order with no dependencies between them or even in parallel on different computers. The merge waits until both parallel braches finished their execution.

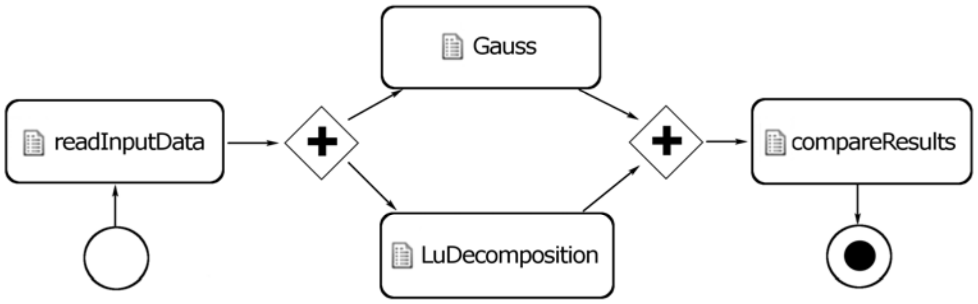

Figure 2. The example workflow consists of four script tasks and two control flow elements. It solves a linear equation with two different algorithms and compares the results.

The workflow solves the linear equation system $A\vec{x} = \vec{b}$ where matrix A and vector $\vec{b}$ are given but the solution vector $\vec{x}$ must be calculated. The first script task "readInputData" reads the input data in form of matrix A and vector $\vec{b}$. Then the data is forwarded to the two script tasks "Gauss" and "LuDecomposition" which calculate the solution vector $\vec{x}$ using their eponymous algorithm. Both solution vectors are compared by the last script task "compareResulst".

The data flow can be seen analogous to the control flow. Every time a task produces data, it will be consumed by its successor task. Every time a task consumes data, it was produced by its predecessor task. So the data is hand over from task to task to be modified like in a pipeline

Listing A shows the BPMN 2.0 workflow template XML source code of the four script tasks. The template source code line numbers occur in the following explanation text in brackets, e.g. (A3). The script syntax is Java based: scriptFormat=http://www.java.com/java in lines (A1), (A6), (A10), (A14). Each script task forwards its execution to static Java methods which are implemented in the class "Tasks" (2) (3) (7) (11) (15). The class "Data" stores the data which are globally available only in the workflow template. The data are not visible outside of the workflow template, especially not in the script task methods of the class "Tasks". These data are used to store input and output values of the script task methods. The workflow is designed for execution in a defined on-premise environment without cloud integration. Therefore, only the service invocation is done (Step 4 of Procedure 1).

```
    ...
 1  <scriptTask name="readInputData" id="_3" scriptFormat="..."><script>
 2  Data.inputA = Tasks.readInputA();
 3  Data.inputB = Tasks.readInputB();
 4  </script></scriptTask>
 5
 6  <scriptTask name="LuDecomposition" id="_4" scriptFormat="..."><script>
 7  Data.resultX1 = Tasks.solve(Data.inputA, Data.inputB, "ludecomp");
 8  </script></scriptTask>
 9
10  <scriptTask name="Gauss" id="_5" scriptFormat="..."><script>
11  Data.resultX2 = Tasks.solve(Data.inputA, Data.inputB, "gauss");
12  </script></scriptTask>
13
14  <scriptTask name="compareResults" id="_6" scriptFormat="..."><script>
15  Data.comparison = Tasks.compareResults(Data.resultX1, Data.resultX2);
16  </script></scriptTask>
    ...
```

Listing A. BPMN source code of all four script tasks in the template of the example workflow.

The computational resources have neither to be integrated nor to be terminated (Steps 1 to 3 and Step 5 of Procedure 1).

The two computation intensive script tasks in this workflow are “Gauss” and “LuDecomposition”. The workflow management system calls the static Java method “solve” during workflow execution of both of these tasks. Listing B shows the Java source code of this method. The implementations of the other methods are not relevant for the context of this article and will be skipped therefore. The readability of all Java source codes in this article was improved by removing all exception handling and all Java imports. The computation intensive algorithms will not be executed on the workflow management system computer. Instead the method invokes a web service which is indicated by an endpoint. It is possible that both script tasks call the same Java method because both different web services provide the same interface only on different endpoints (A7) (A11). The correct web service endpoint is fetched from the ServiceRegistry lookup table by evaluating the parameter value of “alg” (B8). The ServiceRegistry (B3) assigns all service names to web service endpoint URLs independent if the web service is running on-premise or off-premise in the cloud. A possible assignment is given in line (B2). The workflow management system feeds the ServiceRegistry during workflow instantiation. The implementations of the mathematical algorithms behind the web services are out of scope of this article. It is assumed that the services simply exist. The client for the equation solver web service is created by Java API for XML Web Services (JAX-WS) using the EquationSolver interface, the web service endpoint, and the web service description language (WSDL) file (B9) – (B11). The invocation of the corresponding equation solver web service generates the result (B12). After the result is returned to the workflow management system the script task’s execution is finished and the workflow execution will continue with the merge control flow element (B13).

```
      public class Tasks {
 1    // the workflow management system fills this map
 2    // ServiceRegistry.put(“gauss”, “http://149.201.206.241:8000/gauss”
 3    public static Map<String, String> ServiceRegistry;
 4
 5    //Data.resultX1 = Tasks.solve(Data.inputA, Data.inputB, “ludecomp”);
 6    //Data.resultX2 = Tasks.solve(Data.inputA, Data.inputB, “gauss”);
 7    public static double[] solve(double[][] A, double[] b, String alg) {
 8        String serviceUrl = ServiceRegistry.get(alg);
 9        Service service = Service.create(new URL(serviceUrl + “?wsdl”),
10                new QName(“http://webservice/”, alg + ”Service”));
11        EquationSolver solver = service.getPort(EquationSolver.class);
12        double[] resultX = solver.solveEquation(A, b);
13        return resultX;
14    }
15    ... // additional methods for other script tasks
16    }
```

Listing B. Java source code four the cloud interaction and web service deployment.

3. Prototype

A prototype which withstands the challenges indicated in Section 2 was developed in close relation to the example workflow of Sub-section 2.6 to evaluate the concept. Sub-section 3.1 describes how the example workflow is manually extended to prepare the cloud integration. Sub-section 3.2 describes the reusable workflow independent software to manage the cloud integration into arbitrary workflows. Sub-section 3.3 provides extensions to automate and optimize cloud integration.

3.1 Extension of the Example Workflow To Prepare Cloud Integration

The example workflow of Figure 2 described in Sub-section 2.6 will now be extended to prepare the cloud integration. Additional computational power is needed if the available on-premise resources are insufficient. The two computation intensive tasks are “LuDecomposition” and “Gauss” which can be executed in parallel. Let assume that there is only enough on-premise computational power to execute one of these tasks, e.g. “LuDecomposition”. That means that the “Gauss” task must be executed in the cloud which resources will be allocated and integrated into the workflow on runtime. Tasks with short runtimes will be ignored in this description.

Listing C shows the XML workflow template source code extension to integrate the cloud resources for script task “Gauss” execution. The remaining template source code stays unchanged like in Listing A. The only modifications are two source code lines which perform Steps 1 to 3 from Procedure 1 to integrate the cloud resource instance and Step 5 to terminate the cloud resource (C10.5) (C11.5). The class “Cloud” is used to handle interactions with cloud resources. The method “solve” stays unchanged because it only depends on the dynamic web service end point assignment of the ServiceRegistry. The script task “LuDecomposition” still uses the same method “solve” for on-premise execution.

```
10     <scriptTask name="Gauss" id="_5" scriptFormat="..."><script>
10.5   Data.gaussCloud = Cloud.integrateCloud(“gauss”);
11     Data.resultX2 = Tasks.solve(Data.inputA, Data.inputB, “gauss”);
11.5   Data.gaussCloud.stopVM();
12     </script></scriptTask>
```

Listing C. BPMN source code extension to integrate cloud resources for a single task.

3.2 Reusable Workflow Independent Software for Cloud Integration

The previous section has shown how an existing workflow has to be extended to integrate cloud resources. Listing D shows the Java source code of the used method “integrateCloud”. This method is designed to be independent of the example workflow to be reusable for other workflows. The method first fetches the corresponding virtual machine image from the MachineImageRegistry lookup table by evaluating the service name (D4) (D8). This registry works similar to the ServiceRegistry and assigns service names to machine images. The machine image contains the required software in form of a web service. Then the method creates a cloud resource instance and starts the virtual machine (D9) (D10). The cloud resource instance provides the web service endpoint which is published at the ServiceRegistry (D11) (D12). This step guaranties that the “Gauss” web service invocation addresses the correct end point in the cloud without modification of the method “solve” source code in Listing B. The cloud resource handler is returned to the workflow management system in line (D13) to be saved in line (C10.5). The billing period of a Public Cloud service provider would start now together

```
 1   public class Cloud {
 2   // the workflow management system fills this map
 3   // MachineImageRegistry.put(“gauss”, “/images/gauss.img”);
 4   public static Map<String, String> MachineImageRegistry;
 5
 6   // method call: Data.gaussCloud = Cloud.integrateCloud(“gauss”);
 7   public static CloudInstance integrateCloud(String serviceName) {
 8       String machineImage = MachineImageRegistry.get(serviceName);
 9       CloudInstance cloudInstance = new OpenNebulaImpl(machineImage);
10       cloudInstance.startVM();
11       String serviceUrl = cloudInstance.getServiceEndpoint();
12       ServiceRegistry.put(serviceUrl);
13       return cloudInstance;
14   }
15   }
```

Listing D. Java source code to integrate clouds.

with the instantiation of the cloud resource instance. After the result is returned by the web service in line (C11), the method “stopVM()” in line (C11.5) terminates the cloud resource instance to stop the billing period. Now the “Gauss” script task execution is finished and the workflow engine will continue with the merge control flow element. In this example only one cloud resource instance was started to execute the Gauss algorithm. The Workflow management system could directly start additional cloud resource instances. Alternatively the first cloud resource instance could do this.

3.3 Implemented and Envisaged Extensions

The simple prototype presented above provides only the rudimentary functionality to integrate cloud resources for individual workflow tasks. The design was chosen to lower the entry barrier for a simple usage of the basic functionality. The framework is open for extensions which brings new powerful opportunities. Some extensions are presented in the next sub-sections together with their benefits and implementation proposals.

3.3.1 Dynamic Allocation of Cloud Resources

The extension of the example workflow in Sub-section 3.1 to prepare cloud integration was done by a manual insertion of new method invocations into the source code of a script tasks in the workflow template. These insertions can be automated by software to dynamically allocate cloud resources to flexibly compensate on-premise resource shortages. The workflow management system has to check the scheduler if enough on-premise resources are available. If not it modifies only the tasks in the workflow template which should run on integrated cloud resources. This new workflow template is than executed instead of the original one.

Because the workflow management system cannot decide which individual tasks are allowed for execution on integrated cloud resources, this must be configured. One possibility is the usage of task annotations in the workflow template. This is similar to MAUI [8] where developers annotate which methods of an application can be offloaded for remote execution. If annotations are not supported, another possibility is the usage of workflow or task dependent configuration.

Beside the integration into the workflow task, there are other locations where the cloud administration can be integrated: the called functions, the workflow template, and the workflow management system. A comparison of the different administration locations is done in [1].

The concept of the workflow template extension has the benefit of being interoperable with other workflow management systems without individual source code modifications. This makes it even usable for proprietary systems. The same template extension application can be used by different workflow management systems if the same modeling language is supported. Standard workflow modeling languages like XPDL [9] and WS-BPEL [10] benefit most of this approach. Since the usage of automated template modifications has been already validated in [11], no implementation advices are given in this article.

3.3.2 Flexible Selection of Cloud Resources

The cloud itself was abstracted using the wrapper class “OpenNebulaImpl” together with the wrapper interface “CloudInstance” in Listing D in line (D9). The source code is static and always instantiates resources of the same cloud service provider for all tasks. But many other cloud service providers exist. The flexible integration of resources of the most suitable cloud service provider for each individual task is an optimization to form a cross-cloud workflow. The cloud service provider selection could be arranged as Step 0 in Procedure 1.

The selection process can be modeled similar to the three-phase cross-cloud federation model described in [12]. In the discovery phase, the Cloud Service Broker creates a table in a database which provides information about Assured Properties offered by the cloud service providers like in the first three columns of Table 1. Possible properties are special hardware like general purpose GPUs, best performance, lowest price, performance/price ratio, available volume resources of non-pay-as-you-go contracts, and location of the cloud for liable reasons or for data nearness as well as data sensitiveness. This table must always be kept up to date. Each abstract cloud task specifies its requirements. In the example in Table 2, “Gauss” has the requirements “a” and “b” whereas “LuDecomposition” has the requirement “c”. These requirements are sent by the workflow management system to the Cloud Service Broker before the integration of the cloud resource. Now in the match-making phase, the Cloud Service Broker compares the cloud task’s requirements with the cloud service providers’ assured properties. The cloud service providers that assure all requirements of the requesting task are potential task owners. The last two columns of Table 1 indicate which resources are the potential owner of which cloud task. In the authentication phase, the Cloud Service Broker selects the cheapest potential owner as the current owner for each cloud task: “EC2” for “Gauss” and “Azure” for “LuDecomposition”.

Table 1. Assured properties of cloud service providers.

Provider Name	Properties	Price	Potential Owner of	
			Gauss	LuDecomp.
App Engine	a	3	×	×
EC2	a, b	4	✓	×
Azure	a, b, c	6	✓	✓
force.com	b, c	7	×	✓

Table 2. Requirements of individual cloud tasks.

Task Name	Requirements
Gauss	a, b
LuDecompostion	c

One challenge arises if the workflow execution depends on large data because the data movement costs and time have to be considered. It will be a performance bottleneck if tasks are scattered to cloud data centers over the whole world. But different tasks could always be executed on different instance sizes (small, medium, large) in the same cloud data center because the data transmission from one instance to another is normally limited by the network and not by the instance size. Therefore, tasks that process the same data should integrate resources of the same cloud data center. This is addressed by the special possible requirement: “integrate resources of the same cloud data center as task XYZ”. A suitable property is added at runtime to the cloud data center which executes task XYZ. If the selection of the cloud resource should be done fully automatic at runtime, a special selection metric has to be developed to create the optimal configuration of the system. Ideas could be borrowed from the domain of machine learning.

3.3.3 Provenance

The importance of validating and reproducing the outcome of computation processes is fundamental to many application domains. It is exposed that there is a need to capture extra information in a process documentation that describes what actually occurred. The automated tracking and storing of provenance information during workflow execution could satisfy this requirement [13]. The amount and the kind of data to be stored are always user and implementation dependent. Provenance traces enable the users to see what has happened during the execution of the workflow. This also enables failure analysis and future optimization. Provenance becomes even more important in distributed environments because workflow tasks are loosely bound to computational resources. Using provenance in the cloud-workflow domain enables the identification of task to cloud assignments so that it is visible where the cloud task has been executed and where its data have been stored.

Provenance also shows at which time the cloud instance was running and therefore causing costs. Based on provenance traces, statistics can be created showing which workflows cause which costs, which users cause which costs, which clouds cause which costs, which users instantiate which workflows, which clouds execute which cloud task, etc.

A provenance model describes how the gathered provenance data are interpreted and stored in the provenance trace. Several provenance models exist and two of them are described briefly in the following. A detailed comparison is done in [14]. The Open Provenance Model (OPM) [15] is very prominent in the e-Science domain. It provides a comprehensive set of concepts to capture how things came out to be in a given state and is designed to achieve inter-operability between various provenance systems. Another provenance model is the so-called History-tracing XML (HisT) [11]. It was developed within the HiX4AGWS project [16] and provides provenance following an approach that directly maps the workflow graph to a layered structure within an XML document. HisT directly supports the integration of digital signatures and is therefore optimized for the e-Business and cross-organizational domain where responsibility and liability play an important role.

3.3.4 Reuse of Web Services by Keeping Alive Cloud Instances

In scenarios like parameter studies, the same workflow task is executed frequently. Other examples of reusing the same task are loops, multiple workflow instances, and different workflows instances using the same cloud task. The simple prototype introduced above integrates a new cloud instance for each cloud task instance and terminates the cloud instance after the execution of the web service. The cloud instance integration overhead slows down the workflow's execution but can be reduced for future invocations by keeping alive the cloud instance for reusability. A single cloud web service is then used multiple times by different instances of the same abstract cloud task.

The implementation is described in the following: cloud instances are only integrated and terminated using a factory. The stop virtual machine command in Listing C (C11.5) is replaced by a notification of the factory that the web service is no longer needed by the cloud task. The factory keeps alive the cloud instance if it expects future web service invocations. Otherwise, the factory terminates the cloud instance as usual. The prediction is possible by evaluating the instantiated workflows in the workflow management system or simply by setting a keep-alive flag before first workflow instantiation.

3.3.5 Reuse of Cloud Instance by Providing Multiple Web Services

To reduce cloud instance integration overhead, additional web services can be deployed on the same integrated cloud instance. This optimization is most suitable for workflows with different cloud tasks that can then be executed in a pipeline on the same cloud instance. Using this optimization, static machine images cannot be instantiated because additional software must be installed during the uptime of the cloud instance. The installation can be done using SSH in a shell script like in Listing E. The third line copies the program encapsulated in a Java archive file (JAR) from a local repository via secure copy (SCP) to the cloud instance. The fourth line uses secure shell (SSH) to start the remote program. The program will publish its web service as new endpoint on the cloud instance. The password prompts for SCP and SSH are suppressed using public/private key based authentication. The required keys are stored on the workflow management system's computer and in all machine images. Lines 6 to 10 present the usage with example values. Listing F shows the Java main method of the “gauss.jar”. Line 3 sets the program's parameter as web service endpoint. Line 4 publishes the JAX-WS web service class “GaussService” as endpoint.

The shell script and the Java program should be seen as the simplest example. The deployment of web services which run in the context of an application server like JBoss is more complicated but still possible.

3.3.6 User Privileges for Cloud Integration

Many different users instantiate workflows but not each of them should be able to integrate arbitrary cloud resources. Otherwise it would not be possible to map caused costs to individual cloud usages and an abuse of resources would be possible. Therefore, an authentication service is required on workflow side. This service maps the authentication mechanism of the organization to the authentication mechanism of the cloud service provider. The user privileges can be assigned considering many strategies, e.g. a user could have access only a limited time to a cloud or she/he could have only access to specific clouds or for specific workflows. Security Assertion Markup Language (SAML) assertions [17] can be used for this. A standard based security system like WS-

Trust [18], Simple Authentication and Security Layer (SASL) [RFC 4422], oAuth [RFC 5849], or OpenID can be integrated into the workflow management system.

```
#/bin/bash
user=$1; instanceip=$2; serviceendpoint=”http://$2:$3”; program=$4;
scp ~/$program $user@$instanceip:~/$program
ssh $user@$instanceip java -jar ~/$program $serviceendpoint

# usage example:
# ./startws.sh nebula 149.201.206.241 8000/gauss gauss.jar
# scp ~/gauss.jar nebula@149.201.206.241:~/gauss.jar
# ssh nebula@149.201.206.241
# java -jar ~/gauss.jar http://149.201.206.241:8000/gauss
```

Listing E. Linux shell script to copy and start the software on the cloud instance.

```
// args[0] = “http://149.201."206.241:8000/gauss”
public static void main(String[] args) throws Throwable {
  String serviceEndpointUrl = args[0];
  Endpoint.publish(serviceEndpointUrl, new GaussService());
}
```

Listing F. Java main method to deploy the web service out of the “gauss.jar”.

The selection of the corresponding cloud service provider could be done by a human being. The WS-HumanTask specification enables the integration of human interactions into workflows. According to the specification, tasks are assigned to humans depending on their role. These human roles can be mapped on the general security infrastructure. An example where LDAP users were mapped to WS-HumanTask roles is given in [19].

4. Related Work

Cloud computing is the greatest IT hype of the last ten years. Therefore, many publications deal with cloud computing. Surprisingly the combination of cloud computing with workflows is little addressed. The automatically flexible integration of cloud resources to execute tasks of on-premise workflows is not supported yet. In comparison to the mobile smartphone domain, approaches like CloneCloud [20] already exists to dynamically partition applications between weak devices and clouds. Some workflow management systems claim to be ready for the cloud but they are mostly ported from the Grid domain and only support running in the cloud as extension to running in the Grid. The flexible selection and interaction with cloud resources is not implemented in the workflow management systems. One approach is presented in the following and then delimited to the approach presented in this article.

The Generic Workflow Execution Service (GWES) [21] is an open source workflow management system and was developed by Frauenhofer-Gesellschaft for the management and the automation of complex workflows in heterogeneous environments. The service orchestration goes through five abstraction levels: User Request, Abstract Workflow, Service Candidates, Service Instances, and Resources. The formal described User Request represents an abstract operation and is automatically composed into an infrastructure independent non-executable Abstract Workflow. This Abstract Workflow is mapped at runtime down to available Resources. During this process Service Candidates web services are searched and optimally selected to become Service Instances. GWES was originally developed basing on Grid technologies like Globus Toolkit as Grid Workflow Execution Service (also GWES) and was then adjusted to the cloud domain.

The proposed approach of this article differs from the basic GWES concept. GWES is a specific workflow management system with an own workflow description language. In contrast the interoperable approach of this article bases on an extension for existing modeling languages of arbitrary workflow management systems by the simple automatically integration of cloud administration which connect the workflow instance with the Cloud Service Broker to select, start, and terminate cloud instances. By choosing a workflow management system independent approach the benefit of using the already known system is given for the end-user.

5. Evaluation and Conclusion

This article presented a general concept for the hybrid execution of workflows by allowing the off-premise execution of specific tasks in the cloud whereas the remaining tasks stay on-premise to avoid unnecessary costs.

The proposed prototype has the advantage that it neither depends on a particular workflow engine nor on a particular workflow description language. It follows the approach of automatically modifying workflow templates to incorporate the steps for dynamically allocating the appropriate off-premise resources in a flexible manner. The Cloud Service Broker automatically selects the most suitable cloud resource to guaranty the fulfillment of all task requirements. The end users' interfaces are not changed so that workflows can be used the same way as before.

The security, trust, and privacy of the approach plays an important role which is minor addressed in this article. The opponents of cloud computing often criticize the privacy issue. The proposed approach does not consider these critics. It focuses on companies which are already willing to use the cloud and provides possibilities to do this in a more cost-effective way. Security is important on different levels. Cloud resource instantiations and cloud web service invocations must be protected against unauthorized requests to avoid a misuse of resources. The workflow management system has to check and grant assertions for requests. Also the data transmission between workflow management system and the web service on the cloud instance must be encrypted which could decrease the performance.

Next steps of work will be an analysis of an according selection metric for the Cloud Service Broker. The occurred costs of a partial off-premise execution will be compared with the costs of a full off-premise execution to calculate a costs reduction ratio. The time overhead for migrating tasks across cloud and organizational boundaries has to be measured for different providers and set it into relation with the avoided costs. Efficient data movement strategies have to be implemented to reduce migration overheads.

Acknowledgements

This work was carried out in the context of HiX4AGWS. HiX4AGWS is supported of the Federal Ministry of Education and Research in Germany. Grant No.: 17N3409.

References

[1] M Gerhards, V Sander, and A Belloum, About the flexible Migration of Workflow Tasks to Clouds: Combining on- and off-premise Executions of Applications. The Third International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2012), France, July 2012, pp. 82-87.

[2] E Deelman, D Gannon, M Shields, and I Taylor, Workflows and e-science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst., Vol. 25, 2009, pp. 528-540.

[3] J Yu and R Buyya, A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing, Vol. 3, No. 3-4, 2005, pp. 171-200.

[4] X Liu, D Yuan, G Zhang, W Li, D Cao, Q He, J Chen, and Y Yang, The Design of Cloud Workflow Systems. SpringerBriefs in Computer Science.

[5] P Mell and T Grance, National Institute of Standards and Technology (NIST), The NIST Definition of Cloud Computing. Special Publication 800-145, September 2011.

[6] OpenNebula Enterprise Cloud and Datacenter Virtualization http://www.opennebula.org

[7] Open Grid Forum (OFG), Open Cloud Computing Interface (OCCI), June 2011.
[8] H T Dinh, C Lee, D Niyato, and P Wang, A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches. Wireless Communications and Mobile Computing.
[9] R M Shapiro, Workflow Management Coalition Working Group One, XPDL 2.1 Integrating Process Interchange & BPMN. January 2008.
[10]D Jordan and J Evdemon, Web Services Business Process Execution Language Version 2.0 (BPEL). OASIS Standard, April 2007.
[11]M Gerhards, A Belloum, F Berretz, V Sander, and S Skorupa, A History-tracing XML-based Provenance Framework for Workflows. The 5th Workshop on Workflows in Support of Large-Scale Science (WORKS), November 2010, pp. 1-10.
[12]A Celesti, F Tusa, M Villari, A Puliafito, How to Enhance Cloud Architectures to Enable Cross-Federation. 3rd International Conference on Cloud Computing (CLOUD), 2010, pp. 337-345.
[13]Y L Simmhan, B Plale, and D Gannon, A Survey of Data Provenance in e-Science. SIGMOD RECORD, Vol. 34, 2005.
[14]M Gerhards, V Sander, T Matzerath, A Belloum, D Vasunin, A Benabdelkader, Provenance Opportunities for WS-VLAM: An Exploration of an e-Science and an e-Business Approach. The 6th Workshop on Workflows in Support of Large-Scale Science (WORKS), November 2011, pp. 57-66.
[15]L Moreau, B Clifford, J Freire, J Futrelle, Y Gil, P Groth, N Kwasnikowska, S Miles, P Missier, J Myers, B Plale, Y Simmhan, E Stephan, and J Van den Bussche, The Open Provenance Model Core Specification (v1.1). Future Generation Computer Systems, Vol. 27, No. 6, pp. 743-756, June 2011.
[16]History-tracing XML for an Actor-driven Grid-enabled Workflow System (HiX4AGWS), http://www.fh-aachen.de/en/research/projekt-hixforagws/
[17]S Cantor, J Kemp, R Philpott, and E Maler, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, 15 March 2005.
[18]K Lawrence and C Kaler, WS-Trust 1.3 OASIS standard. March 2007.
[19]M Gerhards, S Skorupa, V Sander, P Pfeiffer, and A Belloum, Towards a Security Framework for a WS-HumanTask Processor. 7th International Conference on Network and Service Management (CNSM 2011), Paris, France, October 2011.
[20]B Chun, S Ihm, P Maniatis, M Naik, A Patti, CloneCloud: Elastic Execution between Mobile Device and Cloud. Proceedings of the sixth conference on Computer systems (EuroSys'11), 2011, pp. 301-314.
[21]Generic Workflow Execution Service (GWES) http://www.gridworkflow.org/kwfgrid/gwes/docs/