A Workflow System for Data Processing on Virtual Resources

Rainer Schmidt, Christian Sadilek, and Ross King
AIT Austrian Institute of Technology
Department of Safety & Security
Digital Memory Engineering
Donau-City-Str. 1, 1220 Vienna, Austria
firstname.lastname@ait.ac.at

***Abstract*—This paper describes challenges and approaches that have been addressed during the development of a workflow environment for digital preservation. The system addresses the general problem of efficiently processing collections of binary data using commodity software tools. We present a prototype implementation of a job execution service that is capable of providing access to clusters of virtual machines based on standard grid mechanisms. The service allows clients to specify individual tools and execute them in parallel on large volumes of data. This approach allows one to utilize a cloud infrastructure that is based on platform virtualization as a scaling environment for the execution of complex workflows. Here, we outline the architecture of the workflow environment, introduce its programming model, and describe the service enactment. With this paper we extend work previously presented in [1].**

***Keywords*-data intensive computing; cloud computing; service-oriented, workflow; digital preservation;**

I. INTRODUCTION

Due to rapid changes in information technology, a significant fraction of digital data, documents, and records are doomed to become uninterpretable bit-streams within short time periods. Digital Preservation deals with the long-term storage, access, and maintenance of digital data objects. In order to prevent a loss of information, digital libraries and archives are increasingly faced with the need to electronically preserve large volumes of data while having limited computational resources in-house. However, due to the potentially immense data sets and computationally intensive tasks involved, preservation systems have become a recognized challenge for e-science [2]. Preservation systems must be scalable in order to cope with enormous data volumes, for example such as are produced in fields like science and the humanities. Here, we argue that grid and cloud technology can provide the crucial technology for building scalable preservation systems.

The Planets project [1] aims to provide a service-based solution to ensure long-term access to the growing collections of digital cultural heritage data. The system supports the development, evaluation, and execution of preservation processes based on atomic software components. Components that perform preservation actions often rely on third-party tools (e.g. a file format converter) that must be pre-installed on a specific hosting platform. Planets provides an integrated environment for seamlessy accessing those tools based on defined service interfaces. The workflow execution engine implements the component-oriented enactor that governs life-cycle operation of the various preservation components, such as instantiation, communication, and data provenance. It allows the user to create distributed preservation workflows from high-level components that encapsulate the underlying protocol layers.

A crucial aspect of the preservation system is the establishment of a distributed, reliable, and scalable computational tier. A typical preservation workflow may consist of a set of components for data characterization, migration, and verification, and may be applied to millions of digital objects. In principle, these workflows could be easily parallelized and run in a massively parallel environment. However, the fact that preservation tools often rely on closed source, third-party libraries and applications that often require a platform-dependent and non-trivial installation procedure prevents the utilization of standard high performance computing (HPC) facilities. In order to efficiently execute a preservation plan, a varying set of preservation tools would need to be available on a scalable number of computational nodes. The solution proposed in this paper tackles this problem by incorporating hardware virtualization, allowing us to instantiate sets of transient system images on demand, which are federated as a virtualized cluster. The presented Job Submission Service (JSS) is utilized as the computational tier of a digital preservation system. Jobs are capable of executing data-intensive preservation workflows by utilizing a MapReduce [3] implementation that is instantiated within a utility cloud infrastructure. The presented system is based on the Planets Interoperability Framework, Apache Hadoop [4], and a JSS prototype providing a grid middleware layer on top of the AWS [2] cloud infrastructure.

In this paper, we present on an execution service for

[1] Preservation and Long-term Access through Networked Services, http://www.planets-project.eu/

[2] Amazon Web Services

preservation tools which relies on standard grid mechanisms and protocols like the Job Submission Description Language [5] (JSDL) and the HPC basic web service profile (HPCBP) [6]. We outline the architecture of the Planets workflow environment and introduce an XML-based workflow language that is designed to integrate complex service interaction based on reusable software components. Finally, we present experimental results that have been conducted using the Amazon Simple Storage Service (S3) and Elastic Compute Cloud (EC2) services (AWS) [7]. The paper is organized as follows: In section II we provide an overview of related work in the area of cloud and virtual computing, grids, and digital preservation, section III outlines the problem domain, section IV presents the architecture of the workflow environment, in section V, we introduce the workflow model and language, section VI presents the Job Submission Service and its prototype implementation, section VII reports experimental results, and section VIII concludes the paper.

II. Background and Related Work

A. Cloud and Virtual Computing

The demand for storage and computational power of scientific computations often exceeds the resources that are available locally. Grid infrastructures, services and remote HPC facilities can provide a viable solution for scientists to overcome these limitations. However, many applications require dedicated platforms or need time-consuming adaptations in order to utilize a remote resource. Virtual machine technology provides software that virtualizes a physical host machine, allowing the deployment of platform-independent system images. The deployment of virtual computer instances is supported by a virtual machine monitor, also called a hypervisor. Cloud systems are consumable via Internet-based services offering IT-technology in the form of applications, hosting platforms, or access to computer infrastructures. Amazon's EC2 and S3 services, one of the most prominent commercial offerings, allow users to rent large computational and storage resources on-demand. EC2 is based on the Xen [8] [9] hypervisor allowing one to prepare and deploy virtual system instances that suit individual application needs. S3 provides access to a global, distributed, and replicated storage system. A detailed evaluation of Amazon's compute, storage, and coordination (SQS) web services and their suitability for scientific computing is given in [10] [11]. Deelman et al. provides cost-based analysis of utilizing the Amazon cloud infrastructure for scientific computing [12]. A proof-of-concept study that runs a complex nuclear physics application on a set of virtual machine nodes is presented in [13]. The Nimbus workspace cloud provides a service to scientific communities allowing the provisioning of customized compute nodes in the form of Xen virtual machines that are deployed on physical nodes of a cluster [14]. A study that compares differences of grid and cloud systems that is based on Amazon's EC2 and S3 services is given in [15]. An experiment were a large set of scanned newspaper articles haven been converted to PDF documents using the Amazon cloud infrastructure has been reported in [16].

B. Distributed Data Infrastructures

Research in fields like high-energy physics and earth science produce large amounts of irreplaceable data that must be accessed and preserved over time. For example, in earth observation, data is typically geographically dispersed over different archive and acquisition sites, using a multitude of data and meta-data formats [17]. Grid systems provide dependable access and the coordinated resource sharing across different organizational domains [18]. Data grids [19] focus on the controlled sharing and management of large data sets that are distributed over heterogeneous sites and organizations. In this context, an important aspect is the storage of data in a reliable, distributed, and replicated way. Preservation archives are systems that aim to implement long-term preservation in order to manage data integrity and technological evolution. This includes migrating digital objects to new technologies, maintaining their relationships and preservation metadata. Data grids can be used as the underlying technology to implement digital libraries and distributed preservation archives [20]. The Storage Resource Broker (SRB) [21] of the San Diego Supercomputer center implements a distributed data management environment for data collections based on a virtual file system, logical namespaces, and a metadata repository (MCAT). The iRODS system extends SRB by an adaptive rule system to enforce data management policies based on server-sided micro services [22]. The Transcontinental Persistent Archives Prototype (TPAP) [23] provides a testbed across a number of independent US sites that are linked by high-performance network (DREN), allowing the distribution of electronic records across multiple institutions based on SDSC's SRB. An effort to develop a service-oriented infrastructure for the automated processing of linguistic resources effort is undertaken by the Clarin project [3]. Computational grid systems provide a complimentary technology and are often combined with data grids. For example, the EGEE project [24], currently the world's largest production grid, provides large quantities of distributed CPUs and petabytes of storage. A survey of initiatives that focus on the integration of emerging technologies like digital libraries, grid, and web services for distributed processing and long-term preservation of scientific knowledge is given in [25].

III. Overview

The Planets infrastructure aims to provide an e-research and problem-solving environment for the development

[3] www.clarin.eu

of preservation workflows that supports flexible tool and workflow integration. It supports the planning as well as the execution and evaluation of repeatable preservation experiments. This preservation environment is implemented as a service-oriented architecture that is accessible by users via a portal server. The graphical end-user applications typically implement a scientific experimentation process and access the workflow execution engine (WEE) as part of the portal environment. A major challenge of the workflow execution engine is the enactment of a broad range of experiments that tremendously vary in complexity and scale. Experiments may be performed based on local desktop components, remote application services, as well as by incorporating large-scale compute and storage resources. The workflow environment and execution service presented in this paper addresses the following research issues:

- A grid service that provides access to a variety of third-party tools based on clusters of customized virtual images.
- The incorporation of data intensive computation mechanisms for the efficient processing of non-textual artifacts.
- A high-level workflow language for the task-parallel execution of (parallel) compute jobs on different middleware systems.

IV. The Workflow Environment

This section outlines the workflow execution engine, it's service interaction mechanisms, as well as the programming interface.

A. The Workflow Execution Engine

In the following, we outline the basic interaction pattern between the user application, the workflow environment, and the Job Execution Service. A detailed discussion of the Planets workflow system and its implementation is beyond the scope of this paper. The sequence diagram in Fig. 1 schematically depicts the interaction of a workflow client (*Preservation Application*), the workflow service API (*Workflow Execution Engine*), and the generic service proxy (*Execution Manager*) during workflow execution. The workflow service basically provides SOAP interfaces for the submission and monitoring of workflow processes. A workflow document provides an XML-based description of an executable process (section V), which is typically generated by a workflow editor and/or a domain specific graphical application that utilizes the workflow service. The workflow designer (application) is expected to lookup and select the required services, tools, and job parameters based on the Planets service and tool registries, which provide graphical as well as SOAP interfaces. In its current implementation, the workflow execution engine does not provide advanced resource management capabilities like on-demand service selection, dynamic resource allocations, or quality of service support. After a client has submitted a workflow description for execution, an identifier is returned and the control is handed over to the workflow execution engine. The WEE enqueues the workflow and starts the execution once all required preconditions are met. Resources are limited to the number of overall available cloud nodes and a maximum number of concurrently running workflows. A workflow preprocessing stage (*prepare Workflow*) validates the workflow document and evaluates the resource demand. During workflow execution, each activity is associated with an *Execution Context*, which provides a space that links an ongoing activity (and all its metadata) with the corresponding workflow instance. This includes information such as the service interface, endpoint, tool configuration, walltime, as well as a pointer to the result object. The implementation of the *Execution Context* is specific to the the *Execution Service* that is invoked. At this stage of development, three types of execution services are supported (see Fig. 2). The *LocalExecutionManager* executes local Java components which are typically used for implementing metadata operations and decision logic. The *WebServiceExecutionManager* is used to dynamically invoke remote preservation services. These services implement a predefined Web service profile, which is invoked by utilizing the Web Services Interoperability Technology (WSIT) [4] framework. Planets preservation services implement interfaces and messaging protocols for operations such as file characterization, modification, migration, validation, or comparison [26]. The *EC2ExecutionManager* implements the invocation and message exchange with the job submission service. This service implements a grid service profile and is used to execute long-running and data intensive jobs (section VII). Furthermore, the workflow execution engine provides a method for status inquiry and may send an email notification upon the completion of a workflow.

B. Programming Interface

Planets preservation workflows are build from Java components, allowing a workflow developer to assemble typical preservation cases from atomic services. The workflow API defines a set of functional interfaces that allow users to easily assemble and executable preservation workflows including preservation services like *migrate*, *characterize*, *compare*, or *validate*. The interfaces are compatible among each other and operate based on a minimal data abstraction, called a *digital objects*. Hence, on the API level each service consumes and produces a digital object. A digital object holds metadata like technical, provenance, or other preserva-

[4] https://wsit.dev.java.net/

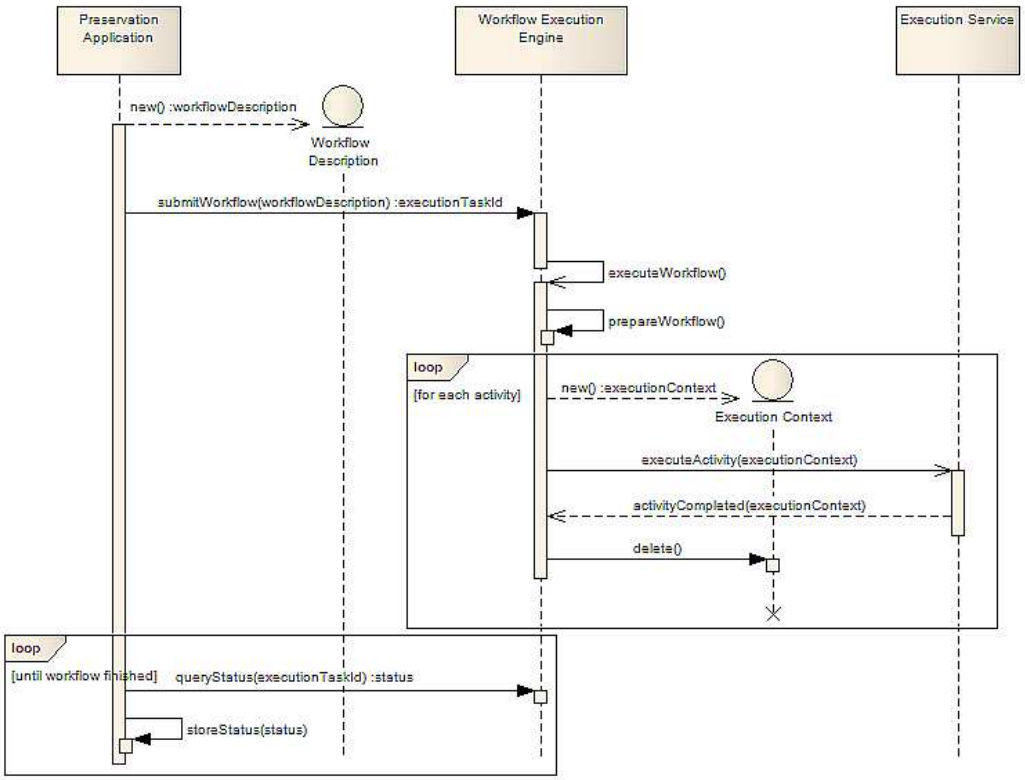

Figure 1. Sequence diagram showing the interactions between a Preservation Application, the Workflow Execution Engine, and the Execution Manager during workflow execution.

tion information about a digital resource including a handle to the actual data. Digital objects can be passed between different preservation services and point to different types of digital resources (e.g. files, collections, archives). The preservation metadata of processed digital objects must be be handled on the workflow level and is managed by trusted Java components.

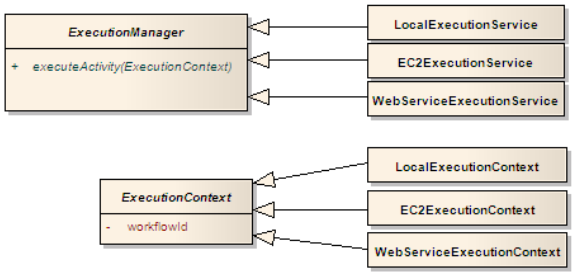

Figure 2. Class diagram showing different implementations of the abstract Execution Manager and Execution Context classes.

V. The Workflow Model

A. Objectives

In this section, we present a resource intensive preservation workflow that can be executed by employing the Planets Job submission service. Such a workflow requires a complex control logic, which must be defined and executed by the workflow system. In section IV-B, we outline a workflow API that abstracts away low-level details such as service interfaces and messaging protocols from the workflow developer. These components could be easily assembled into executable workflow based on the natural programming language (i.e. Java). However, for reasons like simplicity, robustness (e.g. checkpointing and restart), and platform independence, workflows should be defined in a declarative fashion. In section V-C, we introduce initial developments on an XML-based workflow language for orchestrating Planets preservation services, in particular the JSS. Work on this workflow environment is influenced by a number of existing web/grid service workflow systems including DAGMan [27], Triana [28], and GridAnt [29].

B. Use Case and Data Flow

The typical preservation use-case we are targeting is the processing of large data collections. A collection describes

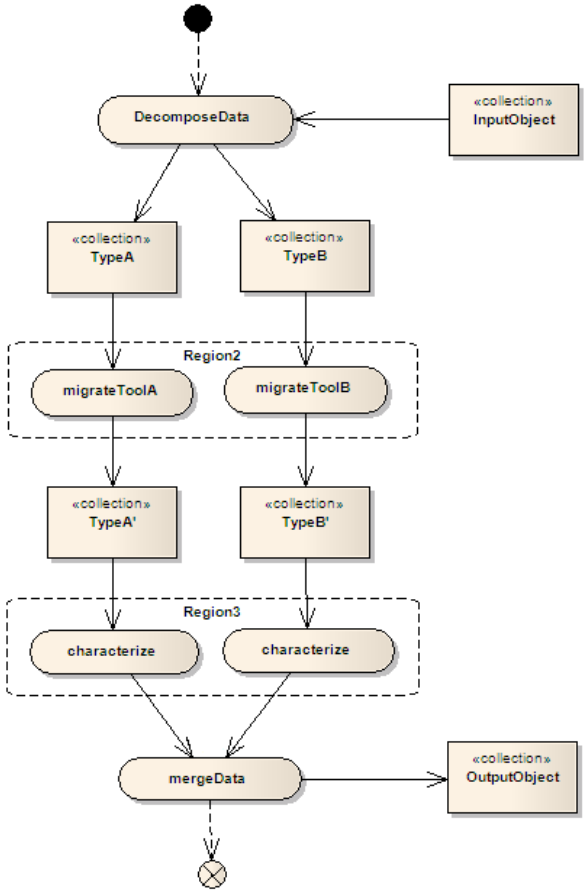

Figure 3. Data flow for a simple bulk processing application: data objects are physical and referenceable entities in a data store, activities are executed on parallel hardware, regions need to be co-scheduled. Messaging, metadata management, control flow and decision logic are not shown in the diagram.

data that is logically interrelated and described using some metadata language. If a collection is organized and curated within a digital repository system, it must be exported first before it can be processed by the presented preservation system. A major difficulty for data preservation in general arises from the diversity of digital data resources and methods to store, describe, and organize them. Examples of data collections we aim to preserve range from simple file collections, over data organized using some markup language (XML, HTML), to data organized in triple stores (RDF/XML).

Figure 3 shows the data flow graph for a simple preservation use case. Parallel *Regions* indicate that one or many tasks might be executed as data parallel jobs. Consider a collection of scanned book pages and associated text documents for a historic book collection, organized by a set of XML files. The idea of the preservation workflow is to convert all images into the JPEG 2000 format and all documents into the PDF/A format. The process flow works as follows; first separate the data into images and documents based on filename extensions. This can be done by running an application (script) for each desired output type. The activity takes a handle to the input data (for example pointing to an S3 bucket) as input and produces a data handle for each output type, represented as collection A and B in the diagram. Once the data is sorted, a migration task is started for each file set using a tool like ImageMagick [5] for the image migration and another tool for PDF/A document conversion. Both activities should be run as parallel jobs in order to minimize execution time. Therefore, each of the migration tasks is launched as a parallel job that executes on a specified number of (e.g. hadoop) cluster nodes. After both migration tasks are finished, collection A' and B' are created. In the next step, one needs to verify the format of the resulting files and extracts relevant properties like file size, image size, or the number of pages. This is done by starting two parallel jobs that invoke a characterization tool like jHove [6] using a handle to collection A' and B' as input. In the final step the data collections are merged and an updated version of the XML records linking to the new data manifestations are generated.

It is important to note that the dataflow graph does not represent the workflow programming model. The presented workflow execution engine follows a more service-oriented approach where the execution services are orchestrated by the WEE during execution time. Hence, a continuous message exchange between workflow execution engine and the preservation services is required. Such a model gives the workflow execution engine much more control over the execution during runtime as compared to batch submission of workflow graphs. This adds additional communication overhead to the overall system but allows one to implement much more complex workflow logic. This is for example required in order to implement decision logic that depends on metadata that is generated and evaluated during runtime.

C. Control Flow

Although a final data flow - as shown in figure 3 - results in a Directed Acyclic Graph (DAG), many workflows cannot be specified in this way. In order to define such processes, it is important to be able to express control logic like conditions or iterations. For a typical preservation workflow that is executed within this environment, it is for example required to evaluate intermediate results or implement error handling. In the following, we describe first results in defining an XML-based workflow language for data-intensive preservation workflows. These workflows can include activities that are local, distributed and/or executed on parallel hardware (i.e. through the JSS). A major design goal is to foster simplicity of the language based on reusable

[5] http://www.imagemagick.org/
[6] http://hul.harvard.edu/jhove/

software components. Therefore, our approach is to encapsulate the complexity of interacting with the system within an extensible set of high-level Java components. A workflow can be assembled by interlinking these component based on an XML document. We employ two abstraction layers: (a) reusable Java components for implementing complex logical tasks and (b) an XML schema for interlinking these components. This approach can be contrasted to the approach taken by low-level service orchestration languages like WS-BPEL. WS-BPEL [7] provides a very precise language that allows the specification of web service interactions at a messaging level using Web service standard languages like BPEL, XML, XPath. However, creating BPEL-based workflow documents can become a difficult and error prone task which is difficult to automate. The presented approach is less universal but designed with the idea in mind to be easily supported by a graphical editor.

D. Example

Figure 4 provides an example workflow snippet for the exection of two activities using the Job Submission Service (JSS). Both services are concurrently executed using the *execute* command. The command does not block the program execution until a corresponding *receive* operation is issued (similar to MPI [8] send/receive). The service is specified by its endpoint address as well as a proxy component (*class*) that implements the interaction with a certain service interface. Furthermore, the preservation service needs to be configured by a list of name-value pairs. The required parameters depend on the service implementation (published within the service registry), which specify the underlying application/tool, specific arguments, or the resource demands (e.g. number of nodes). In case of the execution service this information is required to automatically generate the job descriptor. The service execution is furthermore associated with a handle (*puid*) to the *digital object* representation of the input data. Digital objects contain provenance and other metadata about a physical data entity and are organized within a metadata repository. The *receive* operation blocks the workflow until the corresponding service execution has been completed and a resulting *digital object* has been created. The object represents the result of a preservation service, which might be enriched metadata (e.g. by a characterization) or the generation of new data items (e.g. migration, modification). Methods for evaluation and storing digital objects are implemented by the metadata repository API.

E. File Transfer

A significant research challenge in executing Grid workflows is the transfer of large files between activities. This is in particular true, when the data needs to be transferred between different sites during workflow execution. For the presented experiments, we exploit a utility cloud for running data-intensive experiments and only transfer metadata during workflow execution. The data resides within an virtual storage environment (S3) and is processed by a range of parallel applications.

```
<!-- executing two remote jobs -->
<!-- parallel -->
  <execute id="1">
    <service>
      <class>planets.JSS</class>
      <endpoint>http://...</endpoint>
      <parameters>
        <name>app</name>
        <value>cmdexec.jar</value>
        <name>tool</name>
        <value>/bin/tools/econv.sh</value>
        <name>data_in</name>
        <value>http://s3...</value>
        <name>data_out</name>
        <value>http://s3...</value>
        <name>nnodes</name>
        <value>4</value>
      </parameters>
    </service>
    <puid>puid://s3data/...</puid>
  </execute>
  <execute id="2">
    <!-- declare job2 here-->
  </execute>
  <receive id="1">
    <digObj>digObjA</digObj>
  </receive>
  <receive id="2">
    <digObj>digObjB</digObj>
  </receive>
<!-- /parallel -->
<!-- validating result digObjs. -->
```

Figure 4. XML workflow declaration for execution two concurrently running services. The workflow execution is blocked until both services complete by corresponding receive operations.

VI. The Job Submission Service

A. Motivation

In the context of grid computing and data grids, digital preservation archives are systems that can preserve the output of computational grid processes [20]. An important issue in the context of preserving existing digital content is the process of deriving metadata from digital assets like file collections in order to extract significant semantic information for their preservation (e.g. format characterization). Decisions in preservation planning [30] rely on information that needs to be generated by algorithms and tools for feature extraction, format identification, characterization, and validation [31]. Migrating digital entities between different

[7] www.oasis-open.org/committees/wsbpel/
[8] http://www.mpi-forum.org/docs/

formats typically relies on sequential, third-party libraries and tools that are not supported by scientific parallel and grid systems. Therefore, we propose a service that employs clusters of customizable virtual nodes in order to overcome these restrictions. The IF JSS implements a grid service that provides access to a virtual cluster of large numbers of individually tailored compute nodes that can process bulk data based on data-intensive computing mechanisms and that is integratable with computational and data grid systems.

B. Web Service Profile

Developing an infrastructure for digital preservation involves many grid-specific aspects including the processing of large volumes of data, conducting experiments in distributed and heterogeneous environments, and executing workflows that cross administrative and institutional boundaries. The service presented in this paper focuses on the aspect of submitting and executing data-intensive jobs as part of a digital preservation infrastructure. In order to be able to take advantage of existing grid solutions and to promote interoperability and integration, the IF JSS service is based on a standard grid service profile (HPCBP) for job scheduling (called the basic HPC use case) that is being well adopted by scientific and industrial systems [32]. The OGF Basic Execution Service (BES) [33] defines Web service interfaces for starting, managing, and stopping computational processes. Clients define computational activities in a grid based on JSDL documents. The OGF HPC Basic Profile (HPCBP) specification defines how to submit, monitor, and manage jobs using standard mechanisms that are compliant across different job schedulers and grid middlewares by leveraging standards like BES, JSDL, and SOAP. Our current implementation provides interfaces that support the BES base case specification and accept JSDL documents that are compliant with the HPCBP profile.

C. Basic Service Components

The Job Submission Service (JSS) prototype has been implemented based on a set of exchangeable core components, which are described below. The JSS is a stand-alone Web Service deployed in a Java EE Web Container as shown in Fig. 5. It is secured using HTTPS and SSL/TLS for the transport-layer and WS-Security based on X.509 server certificates and username/password client credentials for the message-layer. In order to submit a request to the JSS, username and password have to be provided that match a previously created account for the institution that utilizes the service. The individual accounts, utilization history, and potentially billing information are maintained by the *Account Manager* component. As HPCBP is used as the web service profile, JSDL documents are used to describe the individual job requests which need to be mapped to physical resources by the resource manager. The *JSDL parser* component validates the XML document and creates an object structure that serves as input for the *Execution Manager*. A *Session Handler* maps service requests based on activity identifiers to physical jobs and keeps track of their current status (e.g. pending, running, finished, failed). The *Execution Manager* interfaces with three components *the Handle Resolver*, *Input Generator*, and *Job Manager* that depend on the resource manager implementation, which is provided by Apache Hadoop in our case. The file handle resolver is used to validate a logical file handle (a URI) and resolve the physical and accessible data reference. The next step is the generation of an input file for a bulk of data that needs to be processed by a parallel application utilizing a particular preservation tool. Finally, the Job Manager prepares a job script and schedules a job using the resource manager.

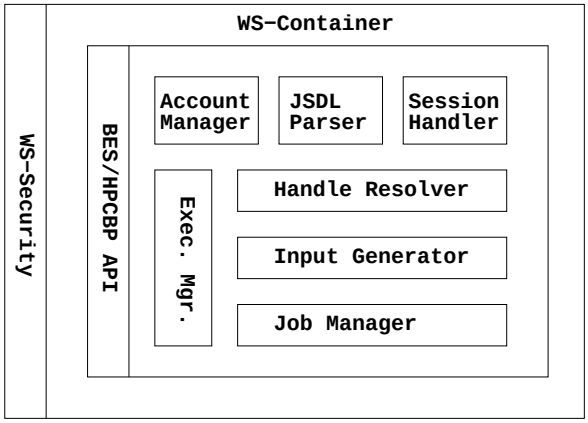

Figure 5. Job Submission Service Components

D. Implementation for MapReduce and Amazon's EC2, and S3 Services

The experimental results presented in section VII have been conducted using an *Execution Manager* implementation for (1) the Hadoop resource manager, (2) Amazon's EC2 compute cloud, and (3) the S3 storage infrastructure. In principle, each of the aforementioned components could be exchanged by different implementations and be connected to different resources, for example a local (e.g Condor [34] based) workstation cluster and network file system. In the following, we describe the functionality of the "cloud-enabled" execution manager. A file handle resolver is used to map a logical handle of a data collection to physical references that are meaningful for the application that needs to access the data (e.g. a file URI, a HTTP URL). Our file handle resolver is implemented in a way that it utilizes the S3 REST-based API to simply generate a list of URIs for files that are contained within an input bucket. The *Input Generator* uses this information to create an input file for the MapReduce application that processes the input data. *MapReduce* is a framework and programming model that has been introduced by Google to support parallel

data-intensive computations. Apache Hadoop is an open source MapReduce implementation that can be used to cluster commodity computers. Also, Hadoop provides built-in support for EC2 and S3. We use Hadoop's own distributed file system to store input files across the computing nodes. The *Job Manager* component passes the input file together with an MapReduce application (the *CommandExecuter*) and information extracted from the JSDL object to the Hadoop job scheduler. The *CommandExecuter* is responsible for handling the S3 bulk data i/o, processing the *input splits* based on pre-installed applications as specified by the user, and for output generation. Finally, the outputs produced by each node are merged to form the output data collection.

VII. Experimental Results

A. Preliminary Considerations

The experiments were carried out as a quantitative evaluation of utilizing a virtual, cloud-based infrastructures for executing digital preservation tools. For all experiments, a simple workflow was implemented that migrates one file collection into a new collection of a different format using the `ps2pdf` command-line tool. It is important to note that the selected tool is replaceable and not relevant for the presented experiments. Four dimensions have been analyzed and compared to sequential executions on local execution environments: the execution time, the number of tasks, the number of computing nodes, the physical size of the digital collections to migrate. As performance metrics we calculate Speedup and Efficiency [35] as formally described in equations $S_{s,n}$ (1) and E_p (2).

$$S_{s,n} = Tseq_{s,n}/Tp_{s,n} \quad (1)$$

$$E_p = S_{s,n}/p \quad (2)$$

where:
s - is the physical object size,
n - is the number of tasks,
p - is the number of computing nodes.
$Tseq$ - is the sequential execution time,
Tp - is the execution time with p computing nodes.

B. Experiment Setup

For the experiments, we utilized the Amazon Elastic Compute Cloud (EC2) as a cloud infrastructure, leasing up to 150 cluster nodes, each running a custom virtual images based on RedHat Fedora 8 i386, Apache Hadoop 0.18.0, and a set of pre-installed the migration tools. The used default system instances provide one virtual core with one *EC2 Compute Unit*, which is equivalent to the capacity of a 1.0-1.2 GHz 2007 Opteron or a 2007 Xeon processor. Bulk data was stored outside the compute nodes using Amazon's Simple Storage System (S3) due to scale and persistence considerations. We experienced an average download speed from S3 to EC2 of 32.5 MByte/s and an average upload speed from EC2 to S3 of 13.8 MByte/s at the Java level. At the time conducting the presented experiments, the per hour price for an EC2 default instance was $0.10.

C. Measurements and Results

For the experiments shown in Fig. 6 we executed all computations on a constant number of five virtual nodes. The number of migration tasks was increased using different sized digital collections to compare the execution time within EC2 to a sequential local execution (SLE) on a single node with identical hardware characteristics. Fig. 6 focuses on the intersection points of the corresponding curves for SLE and EC2 identifying the critical job size for which the parallel execution within EC2 is faster than the sequential execution on a local machine. The results including Speedup and Efficiency for jobs with a large task sizes outside the bounding box of Fig. 6 are shown in table I. For the experiments shown in Fig. 7 we held the number of tasks constant (migration of a set of one thousand 70kB files) and increased the number of computing nodes form 1 to 150 to evaluate scalability. The values for Speedup, Efficiency and execution time were calculated based on the sequential local execution time for a given parallel job. As shown in table II, Speedup increases significantly with an increasing number of nodes due to relatively small overheads of the data parallel application model (see VII-D).

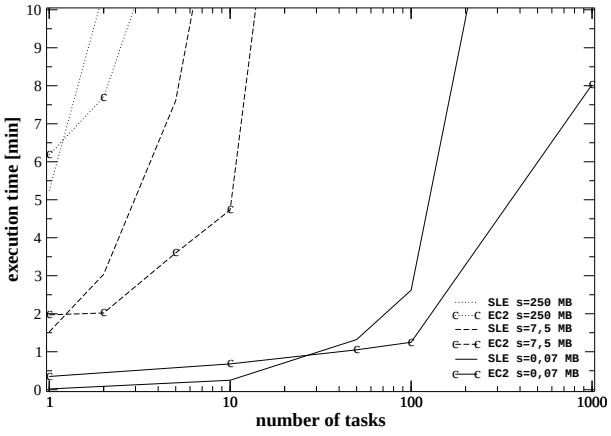

Figure 6. Execution time for an increasing number of migrations tasks and a constant number of computing nodes. The execution on five (EC2) nodes is compared to a sequential local execution (SLE) of the same task.

D. Interpretation of Results

Already for a small number of migration tasks the parallel execution within EC2 proved to be faster than the sequential execution on a single node (see Fig. 6). A Speedup of 4.4 was achieved for 5 nodes with n=1000 and s=7.5 MB (see table I) proving the suitability and potential of employing

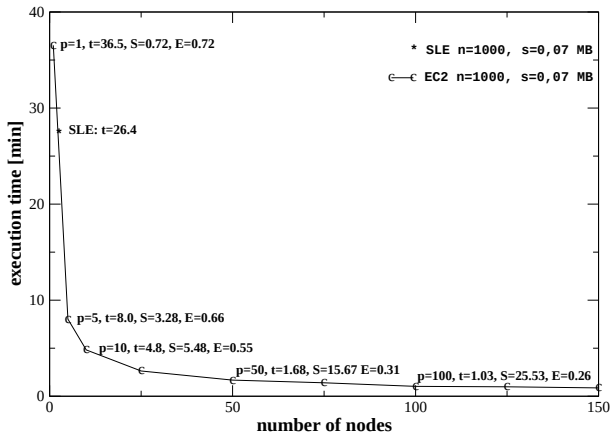

Figure 7. Execution time for 1000 constant migration tasks using an increasing number of computing nodes.

Tasks (n)	Size (s) [MB]	SLE exec. time [min]	EC2 exec. time [min]	$S_{s,n}$	E_p
1000	0.07	26.38	8.03	3.28	0.67
100	7.5	152.17	42.27	3.60	0.72
1000	7.5	1521.67	342.70	4.44	0.88
100	250	523.83	156.27	3.36	0.67
1000	250	5326.63	1572.73	3.37	0.68

Table I
RESULTS OUTSIDE THE BOUNDING BOX OF FIG. 6 INCLUDING SPEEDUP AND EFFICIENCY

(even small) clusters of virtual nodes for digital preservation of large data amounts. Results in Fig. II show that the system achieves good scalability when significantly increasing the number of utilized cluster nodes. However, following overheads which affect the efficiency of the described experiments have been identified: (1) Local execution (SLE) vs. cloud-based execution (p=1, n=1000). The master server for the Hadoop distributed file system which is running on a single worker node added 30% (8min) overhead on that node compared to an SLE (26min). We experienced less than 10% overhead introduced by S3 (compared to a local file system). (2) For a larger number of nodes (p $>$ 50, n=1000) efficiency decreases for various reasons, e.g. coordination. As all nodes are considered blocked until a job is processed, a large fraction of nodes are idle until the last process has finished. Also for short execution times per node, relatively small overheads like network delays and startup time have considerable impact on efficiency.

Number of nodes (p)	EC2 exec. time [min]	$S_{s,n}$	E_p
1	36.53	0.72	0.72
5	8.03	3.28	0.66
10	4.82	5.48	0.55
25	2.63	10.02	0.40
50	1.68	15.67	0.31
75	1.40	18.84	0.25
100	1.03	25.53	0.26
125	0.98	26.83	0.21
150	0.87	30.44	0.20

Table II
RESULTS SHOWN IN FIG. 7 COMPARED TO THE SEQUENTIAL LOCAL EXECUTION OF A GIVEN JOB (N=1000, S=0.07 MB) OF 26.38 MIN.

VIII. CONCLUSIONS AND FUTURE WORK

The emergence of utility cloud services introduced a novel paradigm for the provisioning of large-scale compute and storage resources [36]. Clouds allow their users to lease and utilize hard and software resources residing in large global data centers on-demand. This provides a generic model that can be exploited for business as well as for scientific applications. In the context of high-performance computing, it is obvious that such a model cannot replace dedicated clusters or other high-end and supercomputing facilities. However, it has been shown that applications in the area of data-intensive and high-throughput computing can be well applied to the cloud computing model [37]. Cloud infrastructures provide in general much less specific services than dedicated systems like compute clusters or Grid resources. The AWS EC2 service for example allows the user to control the software that is installed on the utilized virtual machines, commission and decommission computational resources on demand, and it does not require the user to wait for free instances/nodes before using them.

The integration of such resources into an infrastructure for distributed computing provides an important challenge in this context. It is important to identify the differences in orchestrating clouds compared to existing service-computing models. In this paper, we have presented a grid execution service that provides parallel processing of bulk data based on customizable virtual nodes as part of a digital preservation infrastructure. This service has been deployed and evaluated using Amazon's utility cloud infrastructure. We argue that building such computational services based on virtual images can provide a viable technology for the provisioning of domain-specific applications on a larger scale. Furthermore, we introduce work on a workflow system for the concurrent orchestration of cloud-based execution services. Future work will deal with the employment of a common authorization mechanism and protocol for secure web-based data access. In the area of digital libraries and archives, we feel that in particular, legal concerns, security policies, and SLAs will require extensive consideration. Another research goal will be the elaboration of resource management issues for on-demand computing. In particular, we will investigate in scheduling algorithms for distributing tasks across cloud nodes and clusters.

ACKNOWLEDGMENTS

Work presented in this paper is partially supported by European Community under the Information Society Technologies (IST) 6th Framework Programme for RTD - Project Planets (IST-033789).

REFERENCES

[1] R. Schmidt, C. Sadilek, R. King, "A Service for Data-Intensive Computations on Virtual Clusters," *First International Conference on Intensive Applications and Services (INTENSIVE09)*, 2009.

[2] Digital Preservation Coalition, "Digital Curation: digital archives, libraries, and e-science," Seminar report, London, UK, 19 October 2001, http://www.dpconline.org/graphics/events/ digitalarchives.html.

[3] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proceedings of OSDI'04*, 2004.

[4] Apache Hadoop, http://hadoop.apache.org/.

[5] A. Savva et. al, "Job Submission Description Language (JSDL) Specification, Version 1.0," Technical Report, 2005.

[6] B. Dillaway, M. Humphrey, C. Smith, M. Theimer, and G. Wasson, "HPC Basic Profile, v. 1.0. GFD-R-P.114," Technical Report, 2007.

[7] Amazon Web Services, http://aws.amazon.com.

[8] The Xen Project, http://xen.org/.

[9] Amazon Web Services, "Overview of security processes," June 2009, http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf.

[10] S. Garfinkel, "An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS," Technical Report TR-08-07, Tech. Rep., 2007.

[11] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science grids: a viable solution?" in *DADC '08*, 2008, pp. 55–64.

[12] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the Montage example," in *In Proceedings of SC'08*, 2008, pp. 1–12.

[13] K. Keahey, T. Freeman, J. Lauret, and D. Olson, "Virtual Workspaces for Scientific Applications," in *SciDAC 2007 Conference*, June 2007.

[14] Nimbus Cloud, http://workspace.globus.org/clouds/nimbus.html.

[15] "An EGEE Comparative Study: Grids and Clouds Evolution or Revolution," 2008, https://edms.cern.ch/file/925013/3/EGEE-Grid-Cloud.pdf.

[16] D. Gottfrid, http://open.blogs.nytimes.com/ 2007/11/01.

[17] J. V. Bemmelen, L. Fusco, and V. Guidetti, "Access to Distributed Earth Science Data Supported by Emerging Technologies," in *EnviroInfo 2005*, September 2005.

[18] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[19] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of Data Grids for distributed data sharing, management, and processing," *ACM Comput. Surv.*, vol. 38, no. 1, p. 3, 2006.

[20] R. Moore, A. Rajasekar, and M. Wan, "Data Grids, Digital Libraries, and Persistent Archives: An Integrated Approach to Sharing, Publishing, and Archiving Data," *Proceedings of the IEEE*, vol. 93, no. 3, pp. 578–588, March 2005.

[21] A. Rajasekar, M. Wan, and R. Moore, "MySRB & SRB: Components of a Data Grid," in *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002, p. 301.

[22] M. Hedges, A. Hasan, and T. Blanke, "Curation and Preservation of Research Data in an iRODS Data Grid," in *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, 2007, pp. 457–464.

[23] The U.S. National Archives and Records Administration, "The Transcontinental Persistent Archives Prototype (TPAP)." [Online]. Available: http://www.archives.gov/era/research/tpap.html

[24] Enabling Grids for E-sciencE (EGEE), http://project.eu-egee.org.

[25] L. Fusco, J. van Bemmelen, and V. Guidetti, "Emerging technologies in support of long-term data and knowledge preservation for the earth science community," in *PV 2005*.

[26] R. Schmidt, R. King, A. Jackson, C. Wilson, F. Steeg, and P. Melms, "A framework for distributed preservation workflows," in *Proceedings of The Sixth International Conference on Preservation of Digital Objects (iPRES)*, San Francisco, USA, 2009.

[27] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow Management in Condor," in *Workflows for e-Science*, 2007, pp. 357–375.

[28] I. Taylor, I. Wang, M. Shields, and S. Majithia, "Distributed computing with Triana on the Grid," *Concurrency and Computation:Practice and Experience*, vol. 17, no. 1–18, 2005.

[29] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi, "Gridant: a client-controllable grid workflow system," 2004, pp. 10 pp.+.

[30] *Reference Model for an Open Archival Information System (OAIS), Blue Book, Issue 1*, CCSDS - Consultative Committee for Space Data Systems, January 2002.

[31] C. Chou, "Format Identification, Validation, Characterization and Transformation in DAITSS," in *Proceedings of IS&T Archiving 2007*, May 2007, pp. 151–156.

[32] C. Smith, T. Kielmann, S. Newhouse, and M. Humphrey, "The hpc basic profile and saga: standardizing compute grid access in the open grid forum," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 8, pp. 1053–1068, 2009.

[33] I. Foster et al., "OGSA Basic Execution Service Version 1.0." OGF, GFD-R-P.108, August 2007.

[34] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[35] D. Eager, J. Zahorjan, and E. D. Lozowska, "Speedup Versus Efficiency in Parallel Systems," *IEEE Trans. Comput.*, vol. 38, no. 3, pp. 408–423, 1989.

[36] I. Foster, "There's Grid in them thar Clouds," personal blog, January 08, 2008, http://ianfoster.typepad.com/blog/2008/01/theres-grid-in.html.

[37] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science Clouds: Early Experiences in Cloud Computing for Scientific Applications," *First Workshop on Cloud Computing and its Applications (CCA'08)*, 2008.