An Automated Approach to Cloud Storage Service Selection

Arkaitz Ruiz-Alvarez
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
arkaitz@virginia.edu

Marty Humphrey
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
humphrey@virginia.edu

ABSTRACT

We present a new, automated approach to selecting the cloud storage service that best matches each dataset of a given application. Our approach relies on a machine readable description of the capabilities (features, performance, cost, etc.) of each storage system, which is processed together with the user's specified requirements. The result is an assignment of datasets to storage systems, that has multiple advantages: the resulting match meets performance requirements and estimates cost; users express their storage needs using high-level concepts rather than reading the documentation from different cloud providers and manually calculating or estimating a solution. Together with our storage capabilities XML schema we present different use cases for our system that evaluate the Amazon, Azure and local clouds under several scenarios: choosing cloud storage services for a new application, estimating cost savings by switching storage services, estimating the evolution over time of cost and performance and providing information in an Amazon EC2 to Eucalyptus migration. Our application is able to process each use case in under 70 ms; it is also possible to easily expand it to account for new features and data requirements.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*distributed systems, performance evaluation (efficiency and effectiveness)*

General Terms

Algorithms, Measurement, Economics

Keywords

Cloud computing, cloud storage services, matching algorithms

ScienceCloud'11, June 8, 2011, San Jose, California, USA.

1. INTRODUCTION

Scientists are increasingly relying on computational resources, both compute and storage, to expand scientific knowledge. Simulation is now a key component of the scientific method, but it requires compute capabilities beyond the single workstation. The amount of data gathered from scientific instruments is also quickly overcoming the capacity of storage systems; a phenomenon called "data deluge" [9]. Because of these two reasons, scientists are looking to expand their local resources (workstations and clusters) with highly available and scalable systems. In this paper, we focus on the storage of data in cloud computing [5]. Each cloud provider offers different storage abstractions: Amazon provides the S3, EBS, SimpleDB and Relational Database storage services [1]; similarly Windows Azure provides the Blob, Table, Drive and SQL Azure services [12]. For local storage resources users have traditionally used NFS drives and sometimes high performance distributed systems such as the GPFS and Hadoop.

In this new context, with multiple and very different options for storage, users are facing a daunting task when trying to select a storage service. There are multiple aspects that have to be taken into account: performance, cost, specific features, long term management issues, etc. For example, a user may decide to store some data in the Blob service of the Azure cloud; read throughput is strongly dependent on the number of concurrent clients and going from 1 client to 11 concurrent ones the throughput per client decreases by 40% [10]. Application performance is affected by the selection of the storage and several variables (number of clients, size of data and requests, etc.). Users would also like to select the cheapest storage service that meets the data requirements: storing data in Amazon's Reduced Redundancy Storage is cheaper than the regular service, but comes at the cost of reduced durability. Amazon's RRS is appropriate for intermediate results and would reduce the storage bill for these data by 33%. Selecting a storage service can also be limited to specific characteristics, such as support for data versioning. To further complicate this selection, the requirements for existing data change over time and new datasets and applications are added to the cloud so choices need to be reconsidered periodically.

We believe that this complex decision should be automated, and we present our approach in this paper. First, we identify the multiple characteristics and features of storage systems and propose an XML schema that includes them. Second, we present examples of current storage systems that

are fully described using our schema. Third, we take several use cases drawn from our experience in the field of scientific computing and algorithmically match their storage needs to concrete storage services. The final result of our system is a match of services and requirements, which ensures that data storage meets users' requirements (durability, availability, etc.), performance expectations (latency and throughput at scale) and provides cost estimates. For example, a migration from the public Amazon cloud to a private Eucalyptus deployment would usually involve the movement of data out of the Amazon cloud. However, we find that for one of our use cases continuing using the SimpleDB service from our local cluster in Virginia offers good performance and the data transfer and storage cost is negligible. We summarize the contributions of this paper in the following list:

- We devise a XML schema capable of fully describing cloud and local storage systems. Our goal is to algorithmically process this information to match users' requirements. We present several descriptions of storage services, including the most commonly used services from Amazon and Windows Azure.
- We describe several use cases in distributed applications that benefit from our system. The possible benefits for the users are multiple: our results can guide application design, estimate costs, calculate cost savings by switching clouds or storage systems with possible tradeoffs and assist with storage system selection in private cloud deployments.

We evaluate two aspects of our approach: the extensibility of our approach, and the wall clock time spent processing each use case. Since cloud computing is an evolving field cloud providers will change existing storage systems and release new ones. We present the coding effort required for each data requirement that the user can check against the description of the storage systems; in our example *Durability* is coded in under 100 lines of C# code. For each of our use cases we present time measurements processing each cloud provider (Amazon, Azure and local) and the total wall clock time; all measurements fall below 70 ms.

The storage selection system presented in this paper is part of a broader effort to automate data management in cloud computing. Our final goal is to present the users with the following: a generic API for accessing data and a mechanism to add datasets (which includes both the data and additional metadata such as storage requirements). The underlying storage services and APIs will be hidden from the user; a data management layer can evaluate each available storage system while taking into account several factors (cost, latency, etc.) that affect all the user's datasets. By introducing a data access layer (above the cloud storage systems) changes in data requirements (different usage patterns, cold data) and storage systems (lower prices, different performance, new services) can be detected and we can run some of the use cases presented in this paper without user intervention and without modifying the applications. The storage service selection problem presented in this paper is the first step in our work.

In the next section we review related work. In Section 3 we discuss the target storage systems of this work. We introduce in Section 4 our XML schema; and in Section 5 example descriptions of current storage systems. Section 6 describes several example use cases for our system. These use cases, together with the schema, are evaluated in Section 7. We finally conclude in Section 8.

2. RELATED WORK

Recent work in the cloud computing area has focused on the storage APIs and underlying systems. We can find in the literature evaluations of current cloud storage systems such as Amazon S3 [16] and Windows Azure [10]. Additionally, new storage systems such as an elastic transactional data store have been suggested [6]; other papers focus on the storage stack [2] and moving from the file system interface to scalable cloud storage [17]. The focus of our proposed work is not introducing new storage systems or APIs, but rather developing a higher level process that will use these underlying systems. Thus, our aim is to express the characteristics and features of current storage systems (while being flexible enough to accommodate future ones) in a machine readable format. These descriptions allow us to automate processes such as storage selection and cost analysis; we could implement past manual work that have evaluated specific applications such as Montage [7] or platforms such as grids for volunteering computing [11].

Although we mainly focus on cloud computing platforms with different storage systems, our approach also relates to existing data grids [4], whose services provide abstractions for accessing and storing data. However, there is an important limitation for applying data grid APIs to the current environment: data grids usually use the file abstraction as the basic unit of storage while cloud computing has introduced new data storage interfaces. A file instance in grids corresponds to the blob instance in clouds, but the queue and table cloud abstractions have no corresponding counterpart in grids. Thus, we need to extend previous work on these storage abstractions such as the GLUE schema [3] in order to accommodate the new cloud services. One of the new characteristics is cost: every hour of computation, GB stored remotely or transferred over the network has an explicit price tag. Prior to the cloud, the price to pay for computing had been hidden from users; in cloud computing users are faced with the task of optimizing cost, but this task is usually daunting without an automated approach since there exists multiple cloud providers with different storage options and prices.

Since data is commonly replicated in a distributed system, applications need access to a replica catalog to locate the actual data [23] and implement a strategy for replica selection; Condor relies on a matchmaking algorithm to select a replica that fits the application requirements. The requirements are specified as expressions over attribute-value pairs, for example "reqdSpace = 5G". Matching replicas are ranked by a certain attribute, for example available space. The matchmaking framework [18] introduces the matchmaking algorithm and protocols (for advertising, matchmaking and claiming) to assign each user's request to a resource provider and process it. Our approach is similar although instead of selecting replicas we select storage services. We also present an extensible interface to execute arbitrary code instead of doing an attribute-value match: this way we can take into account the particularities of cloud computing and match higher level requirements against the different cloud storage services.

3. TARGETED STORAGE SYSTEMS

Until now, scientists usually have very limited options for data storage. A usual local cluster solution includes a small user directory (with backup), an NFS system that can hold several GB of data, and a big scratch temp folder. More advanced systems have also included a high performance parallel filesystem, such as GPFS [20]. In recent years cloud computing has burst onto the high performance computing scene and has established itself as a viable alternative to customized HPC clusters for many users who do not have the resources (either time or money) to build, configure, and maintain a cluster on their own. Many eScience developers are increasingly looking to create data-intensive applications [21, 8] that can take advantage of the pay-as-you-go cost model. Thus, in this paper we will focus mainly on cloud storage systems although we also discuss the traditional storage systems for local clusters.

The first cloud provider that we have examined is Amazon. Amazon's pioneer effort in the area [1], API compatibility with other projects (Eucalyptus [15], OpenNebula [14]), wide customer base and the widest and most mature offer of cloud services make Amazon EC2 our primary target for cloud providers. Within the Amazon cloud we consider the following storage services: S3, EBS, SimpleDB and Relational Database. S3 provides storage for objects of a wide range of sizes (up to 5 TB), which are organized into buckets. The most closely related interface to S3 is the traditional directory/file interface. EBS offers a device type interface, in which a virtual hard drive (formatted with a filesystem) can be attached to a Virtual Machine running on the Amazon cloud. SimpleDB is based on tables that store items composed by attribute/value pairs. Although SimpleDB does not support a rich SQL interface its potential for scalability is superior. Finally, the Relational Database Service is essentially a MySQL database running on the cloud. Each storage service is offered in different regions that include the United States, Europe and Asia. In summary, Amazon offers very different options for data storage; each option is best suited to certain tasks.

We have also considered Microsoft's Windows Azure [12] cloud which, although it provides the Platform as a Service (PaaS) abstraction, gives much flexibility in terms of data storage. The storage offer in Azure is very similar to Amazon: Blobs are similar to S3, Azure Drives to EBS, SQL Azure to Relational Database Service and Tables to SimpleDB. Even though the storage abstraction are essentially the same, the implementation details differ. For example, Amazon S3 offer both regular and reduced redundancy storage unlike Windows Azure Blob. Thus an application can make the explicit tradeoff of cost and durability in Amazon to save money; this tradeoff is not possible for Azure Blob. On the other hand, Azure Blob comes in two flavors: page and block. Streaming data are best kept in block blobs, while random accessible data in page blobs. On top of the differences between features there are performance and cost differences that, for some given application's requirements, can tip the balance in favor of one of these cloud providers.

In addition to the Amazon EC2 and Azure cloud platforms we have included several local storage systems: users directories and scratch folders mounted over NFS in a local cluster, and a local Hadoop deployment. We believe that the inclusion of storage systems from these three sources covers the vast majority of storage systems available (SQL, NoSQL -scalable tables-, block devices, files, etc.). We also take into account that storage systems are continuously evolving: different aspects related to extensibility are introduced in the next sections. In summary, we try to target a broad range of storage systems so that our system does not artificially limit the number of possibilities presented to the user.

```
<xsd:element name="CloudProvider"
 type="tns:CloudProviderType"/>
<xsd:complexType name="CloudProviderType">
 <xsd:element name="StorageServices">
   <xsd:element name="StorageService">
     <xsd:element name="Regions">
       <xsd:element name="Cost">
       <xsd:element name="Performance">
     <xsd:element name="StorageAbstraction">
       <xsd:element name="Container">
         <xsd:element name="Object">
<xsd:/complexType>
```

Figure 1: The hierarchical organization of the most important elements in the XML schema. The global element is the cloud provider, which offers several storage services, each one representing a storage abstraction, across multiple regions that vary on cost and performance.

4. XML SCHEMA

In this section we present the general structure of the XML schema used to describe the storage systems supported by the different cloud providers. We also introduce the types declared and focus our attention on a couple of examples. The complete schema is readily available on our website [19]: it currently features 54 complex types in over 500 lines. Figure 1 shows the most important elements; the global element is the CloudProvider and the second level is the StorageServices element. Each StorageService element represents a certain Storage Abstraction that is offered in several Regions. An example of a Storage Abstraction is the Windows Azure Table. In general, we find that each storage abstraction can be thought of as a set of containers which store objects. For example, a container may be a table (Azure Tables), a bucket (Amazon S3) or a directory structure (NFS). The respective contained objects are items with attribute/-value pairs, S3 objects and files. This part of the schema focuses on the functional description of the storage system: characteristics and features that appear on the service documentation.

The second child of the StorageService element is the Regions element. Inside this element we find the datacenters where this service is being offered, each one with its cost and its performance. Non-functional characteristics like performance vary from region to region; it is common to have different costs depending on the location of the datacenter because of the variation in electricity prices, labor costs, regulation and taxation, etc. We provide the user with multiple complex types to express the different cost models of clouds: StorageCost (GB per month), DataTransferCost (in and out), RequestCost (measured in number and type of request), QueryProcessingCost (measured in compute time to process a query), OffNetworkDataTransferCost (usually a

```
<xsd:element name="Object">
<xsd:complexType>
   <xsd:sequence>
      <xsd:element name="AcccessControl" type="tns:AccessControlType" minOccurs="0"/>
      <xsd:element name="Interface" type="tns:InterfaceType" maxOccurs="1"/>
      <xsd:element name="Metadata" type="tns:MetadataType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Data" minOccurs="0" maxOccurs="1">
        <xsd:complexType> <xsd:choice>
          <xsd:element name="AttributeValue" > <xsd:complexType>
               <xsd:attribute name="AttributeNameLength" type ="xsd:integer"/>
               <xsd:attribute name="AttributeValueLength" type ="xsd:integer"/>
          </xsd:complexType></xsd:element>
          <xsd:element name="Stream" type="xsd:string"/>
          <xsd:element name="RandomAccess" type="xsd:string"/> </xsd:choice>
      <xsd:attribute name="Formats" type="xsd:string"/>
      <xsd:attribute name="DaysToExpiration" type="xsd:float"/>
      <xsd:attribute name="ReadOnly" type="xsd:boolean"/>
   </xsd:complexType></xsd:element></xsd:sequence>
<xsd:attribute name="ID" type="xsd:string"/>
<xsd:attribute name="Name" type="xsd:string"/>
<xsd:attribute name="Description" type="xsd:string"/>
<xsd:attribute name="NamingRegularExpression" type="xsd:string"/>
<xsd:attribute name="CreationDate" type="xsd:boolean"/>
<xsd:attribute name="ModificationDate" type="xsd:boolean"/>
<xsd:attribute name="MaxSizeNumber" type="xsd:integer"/>
<xsd:attribute name="MaxSizeKB" type="xsd:float"/>
</xsd:complexType></xsd:element>
```

Figure 2: The type definition of the Object element. An object can represent different entities such as a blob, a file or a table item (set of attribute/value pairs).

flat fee for processing a hard drive), HourlyCost (usage cost for a certain resource, such a database server) and ReservationCost (usage prepayment for reservation of a resource during a set period of time).

The Performance element allows us to express performance characteristics for every datacenter. The two that we have used are latency and throughput. In general, we can declare a Measurement of a variable (latency in ms) for an operation (read) and specify the details in several ways: as a simple scalar number, as a polynomial approximation, as a histogram or as a set of sample measurements. Polynomials, histograms and sample sets are based on one or more variables; in our research we have found the number of concurrent clients as the most useful one because of the performance variability. As the number of concurrent clients increases many storage services' performance diminishes; users need to take this into account during the design of cloud applications. Other variables that we could use are, for example, size in KB of the request, size of the object and number of items in the container. The focus on this section of the schema is to provide enough information so our matching algorithm can make a good estimate of the performance of the user's application on this datacenter. The information in this section may originate from websites like Azure Scope [13], which provides up-to-date benchmark results for the Azure platform.

Figure 2 shows the type definition details for the Object element. The attributes of each object, aside from the common ones (ID, Name, Description), describe some features, such as support for creation and modification dates; and limitations of the service, such as the maximum number of sub-elements (for attribute/value pairs) or the maximum size. The possible child elements are: AccessControl, Interface, Metadata and Data. The AccessControl element can be used to describe the multiple systems supported by the storage service: from simple UNIX type permission bits to more elaborated systems such as access control lists and custom access policies languages (both supported in Amazon S3). The Interface element enables us to list every operation supported: from simple create/delete and upload/download to creating snapshots or acquiring a lease; additionally we can include the consistency options and transaction support. The Metadata element contains information about the supported metadata formats (if any), most commonly a set of attributes with string values. The Data element could be a simple stream of bytes with possibly random access support (blobs) or a set of attributes (SimpleDB or Azure Table items).

Finally, we acknowledge that cloud computing is a rapidly evolving field that can make our schema fall out of sync with the storage services. Even though we have included all the features and characteristics of the Amazon and Azure clouds anytime a new feature could be announced that can not be expressed with our schema. Our main response to this challenge is to plan for extensibility: features should be able to be easily and fully included in our system. Regarding the XML schema, this requirement translates in the addition of *<xsd:anyAttribute>* and *<xsd:any>* so that new elements and attributes can be included in XML files describing storage systems while conforming to our schema. More difficult

```
<Object ID="AZURE_BLOB_PAGE" Name="Windows Azure Page Blob" Description="The Blob ..."
    NamingRegularExpression="^(?![0-9]+$)(?!-)[a-zA-Z0-9-]{,63}(?&lt!-)$"
    ModificationDate="true" CreationDate="false" MaxSizeKB="1073741824">
 <Interface>
    <CustomInterface RandomAccess="true">
      <Delete>Delete Blob</Delete>
      <Download>Get Blob</Download>
      <Upload>Put Blob</Upload>
      <CreateSnapshot>Snapshot Blob</CreateSnapshot>
      <ListParts>Get Page Regions</ListParts>
      <UploadPart>Put Page</UploadPart>
      <Lease Duration="60" API="Lease Blob"/>
      <Copy>Copy Blob</Copy>
    </CustomInterface>
  </Interface>
  <Metadata>
    <MetadataInterface>
      <CustomInterface>
        <Download>GetBlobMetadata;GetBlobProperties</Download>
        <Upload>SetBlobMetadata;SetBlobProperties</Upload>
      </CustomInterface>
    </MetadataInterface>
    <MetadataSet type="SystemMetadata" abstraction="ValuePair"/>
      <MetadataSet type="UserMetadata" abstraction="ValuePair"/>
    </Metadata>
    <Data DaysToExpiration="0" Formats="binary;text" ReadOnly="false">
      <RandomAccess/>
    </Data>
</Object>
```

Figure 3: The Object element that describes the paged Blob storage service in the Windows Azure Platform.

is correctly processing these new elements; further sections in this paper will address this issue.

5. DESCRIPTIONS OF STORAGE SYSTEMS

In this section we discuss the actual XML representation of some of the most popular storage systems from the Amazon and Azure clouds. We also give a high level description of how we process these input files. Figure 3 is a snippet of the XML file describing the Windows Azure platform that belongs to the paged Blob service. (Figure 2 is the corresponding section of the schema). This piece of code lists all the valid operations on blobs, as well as the support for attribute/value metadata and the type of data supported (both binary and text that can be randomly accessed). The AccessControl element is absent in this case, since Azure does not allow access control lists at the blob level, but rather at the container level. The rest of the XML file for the Azure platform, as well as the Amazon platform, can be downloaded from our website [19].

We present a high level view of the files in Table 1, where we present some statistics about the different parts of our schema. The description of the StorageAbstraction is a manual and involved process: it requires reading the full documentation of each storage service. The output in terms of lines of code is between 72 and 94, which leads us to believe that any future modifications to this part would be easy to implement.The description for each region (or datacenter) is not larger than the descriptions of the abstraction. However, the number of regions in which a storage service is offered is a significant factor in the length of the file: we count 9 different region options for Windows Azure and 4 for Amazon. Here we are not concerned with the length of the file, because after all it is supposed to be part of program input. What we are trying to highlight is the expected effort to create and maintain these files with accurate and up-to-date information about the cloud providers.

The description of the storage abstraction of the schema is a one-time effort only, plus the corresponding updates whenever a cloud provider changes the storage service to modify some features or add new ones (the latter one being much more likely). More prone to change is the information regarding each region. Cost do change over time, albeit slowly. More commonly there are offers and deals that expire such as temporary free data input to a certain cloud for a couple of months. Cloud providers could publish this cost information in a machine accessible way or perhaps there is the possibility of automatically parsing the HTML documentation. The performance measurements are the most variable part of the schema and we believe that automation is very much possible. We have already mentioned Azure Scope [13], which provides up-to-date information of the performance of the storage services in the Azure cloud. Using this website we can provide more accurate information about the expected latency and throughput of the different services, under different conditions (size of the data, number of batch requests, number of objects already in the container, etc.). A similar website, such as CloudHarmony or CloudClimate, or a benchmark running on behalf of the

Table 1: Average number of lines of XML code needed to describe a storage system, divided by schema elements. Performance and Cost are child elements of Region. Region and StorageAbstraction are children of StorageService.

Cloud Provider	StorageAbstraction	Region	Cost	Performance	StorageService	Total all platform services
Windows Azure	80	70	6	62	731	3673
Amazon	94	36	17	13	246	1498
Local Cluster	72	23	4	15	100	315

user could provide importable performance information for other target cloud systems so there is no need for human intervention; in this paper we collect the performance information from the Azure Scope website for Windows Azure and from our own micro-benchmarks for the Amazon cloud and the local cluster.

6. USE CASES

In this section we present several use cases that, in our opinion, reflect common situations that scientists and other cloud users face. The selection of an storage system has many implications for cloud applications; we provide valuable information about performance and cost expectations evaluating different cloud providers and recommend for each dataset a certain storage service. Currently our storage selection application presents a standard Windows Forms interface to the user, where each of the following use cases is a separate section (tab). The tables included in this section represent what the user sees on the screen.

We would also like to note that cloud computing is an evolving field and therefore any decisions from our experiments, which are presented in the following sections, could change in the future if the cost and performance of cloud services change. Up-to-date information on cloud services is essential and we believe that our XML descriptions of the storage services should be automatically updated, via cloud benchmarks run by the user or via a service that provides this information (we have already mentioned Azure Scope and CloudHarmony).

6.1 Design of an application

The first of our use cases focuses on the choices that cloud users make during the application design. At this stage we assume that we have a good estimation of the data requirements of the application: for each dataset we have the size, the required access latency and throughput and the number of concurrent clients. Our application takes this information and together with our description of the storage capabilities of the different clouds produces an assignment of datasets to storage services. For example, let's consider the following list of datasets and characteristics for a climate simulator:

- Climate measurements come from a satellite feed. It is very important to not lose the data. The size of the data is in the order of several GBs, it is read-only and concurrent access is limited to up to 10 concurrent clients.

- The application follows a bag-of-tasks model where intermediate results, in the range of 10s of MBs, are continuously being produced and consumed for up to 100 concurrent clients. High access latency or low throughput could affect the overall application's performance.

Table 2: Storage services recommendations for the datasets in our first use case by cloud platform. An * indicates storage systems that do not meet at least one user requirement.

Dataset	Amazon	Azure	Local Cluster
Satellite Data	S3	Page Blob	Hadoop*, NFS*
Intermediate Results	S3 RRS*, SimpleDB*	Page Blob*, Table*	NFS*
Experimental Results	S3	Page Blob, Block Blob	NFS

- The final result for each run is a single file, in the order of MBs, which is read only and is globally accessible for any scientist around the globe to download.

The storage service choices made by our application can be summarized in Table 2. For the Azure cloud, the paged Blob service may store the first dataset, where the expected throughput is almost 14 MB/sec and the latency 225 ms. For the second dataset there is not a single storage service that would meet all the requirements. The paged Blob service does not offer very low access latency (205 ms expected). The Table service does have low latency (15 ms) but cannot store items greater than 1MB. Thus, the user would have to judge whether the application may tolerate higher access latency or whether the data could be partitioned. In addition to the user choosing from several storage services that do not meet all the requirements there could be cases where there is missing information. For example, latency benchmarks may not adequately cover the current use case: we have measurements with 10 concurrent clients but not with 100. In this case we choose to display a warning message; the final decision is the user's. Finally, both the paged Blob and the block Blob can meet the requirements for the third dataset.

Similar results are presented for the Amazon cloud. Even in this example with simple user requirements, we believe that our application can provide value to the cloud user by highlighting the differences between latency and bandwidth for each cloud, as well as estimating the storage cost in order to guide the application design process.

6.2 Cost savings analysis

In this use case we analyze an existing cloud application that is currently running in a cloud provider. As an example we will use the Amazon cloud. The developers and users of the application would like to find out if they could lower the monthly bill by switching storage services, within the same cloud or from another cloud provider. For this example, let's consider the following usage of the Amazon storage services:

- S3 holds a total of 2.5 TB of data, which are located

Table 3: Storage services recommendations for our second use case, including monthly savings and tradeoffs.

Current Amazon Service		Service Recommendation			Savings	Pros	Cons
Service	Region	Cloud	Service	Region			
S3	US CA	Azure	Page Blob	US	$11	2.09x better latency	
S3	US CA	Azure	Block Blob	US	$11	2.07x better latency	
S3	US CA	Amazon	S3	US	$36		
S3	US CA	Amazon	S3 RRS	US	$153.5		0.0099999% less durability
S3	US CA	Amazon	S3 RRS	US CA	$127.5		0.0099999% less durability
S3	US CA	Local	NFS	US	$407.5	117.6x better latency	0.499999% less durability
SimpleDB	US CA	Amazon	SimpleDB	US	$.2		
RDS	US CA	Amazon	RDS	US	$92		
RDS	US CA	Azure	SQL	US	$130	1.31x better latency	

in one bucket in the US Northern California Region. The number of blobs stored is 10000.

- SimpleDB hosts data one data domain which stores around 250,000 elements (8 GB).
- Relational Dabase Service stores a 4 GB database. The average number of requests is 5,000 per day.

Table 3 shows the results. For each storage service we present several alternatives that are cheaper. The storage service with more alternatives is S3 and some of them within Amazon. We can cut costs by changing to a cheaper region in the US or by changing to the Reduced Redundancy Storage. While in a cheaper region S3 continues to offer essentially the same level of service, RRS has a tradeoff because of the reduced durability. Also, the Azure cloud provides a service slightly cheaper and with better access latency. Storing data in a local cluster is "free" and has very good latency (local LAN), but is less durable. For the SimpleDB service there are essentially no cheaper options (although there exists equivalent alternatives such as Azure Table or SimpleDB in other regions with a similar cost). Finally, SQL Azure is a cheaper alternative to RDS with better latency. The user, given this information, can evaluate whether the tradeoffs are possible and the savings are worth it.

6.3 Cost and performance estimation

There are many applications in which a change of the backend storage services is difficult: cost of re-programming, no access to source code, scientists have credits for use with a certain cloud provider, etc. But even when the storage services will continue to be the same ones, we believe that there is value in giving the application user estimations for future costs and performance. Given the resource usage detailed in the previous Section 6.2, we shall consider the following trends, this time using the Windows Azure cloud:

- The storage and number of requests in both Blob, Table and SQL Azure will increase at three following rates: 2%, 5% and 15% each month.
- The cloud user would like to consider three different scenarios for concurrent access to the storage services: 10, 25 and 60 clients accessing the Blob and Table data.

In our application we present a table of monthly costs for a certain period of time -a year by default- and for every storage service. Users can see the evolution of the storage costs under different scenarios by changing the assumptions (data growth rates and number of clients). For some cloud applications growth could follow a predictable pattern, for example an application that is only used within a closed community (research lab, university). Other cloud applications, which are open to global users, could scale up suddenly if they become popular. In the first case, we can offer the user with the value of the future bill from the cloud provider. For the second case, our application can provide the user with cost estimations under several scenarios. Aside from cost, we can also generate estimations for the performance metrics included in our XML schema, if they are available. For example, using the Azure Table service we estimate that with 10 concurrent clients the per client throughput will be 31 items/sec. This number does not vary if we increase the number of concurrent clients up to 60. For the Blob service, on the other hand, expected throughput is 28 MB/sec with 10 clients, 25 MB/sec with 25 clients and 17 MB/sec with 60 clients. In summary, given the information collected from these different cloud storage services we can give the users good cost and performance estimates for several scenarios.

6.4 Amazon EC2 to Eucalyptus

Software such as Eucalyptus [15] and OpenNebula [22] can transform a local cluster into a private cloud. In this use case, we consider the following scenario: an application is being moved from the Amazon EC2 cloud to a local Eucalyptus deployment. What does the application user do with the current data? One option would be to continue using the Amazon cloud, but data transfer fees and increased latency can make this approach inviable. Another option is to deploy a local storage system: given the latency and throughput requirements we can choose a NFS installation, a Hadoop deployment or other local data storage systems. Since researchers have already developed private cloud systems that mirror the functionality of public clouds (Eucalyptus and Amazon; Hadoop and Google's Map Reduce and Big Table), it is not unreasonable to expect that, in the near future, an open implementation of storage services such as Blob and Table (SimpleDB) that would expand the options for local storage systems further. In this use case we continue with the usage example in Section 6.2 and recommend the best storage system to select in the cloud or deploy locally.

The output of our application is synthesized in Table 4. We compare each current storage service against local options and itself. In the latter case the client is moved from the Amazon cloud to the local cluster and our application

Table 4: Storage recommendations for our fourth use case. Each current storage service used is compared with local storage services and the current storage service; computation is moved to the local cluster from Amazon EC2. Column cost reflects one time costs (transfer data to new service) as well as monthly cost.

Current Storage S.	New Storage S.	Latency	Throughput	Comments	Cost
S3	NFS	>1ms	39.02 MB/sec	S3 offers 1% more durability (99.99999999%) NFS container capacity is 10 GB (2500 GB req.)	$250 one-time $0 monthly
S3	Hadoop DFS	N/A	N/A	S3 offers 0.00099999% more durability (99.99999999%) Hadoop container capacity is 1024 GB (2500 GB req.) Hadoop does not support random access	$250 one-time $0 monthly
S3	GPFS	N/A	N/A	S3 offers 0.00099999% more durability (99.99999999%)	$250 one-time $0 monthly
S3	S3 (no change)	205 ms	3.17 MB/sec	Data transfer fees incurred by each data access	$0 one-time $362.5 monthly
SimpleDB	MySQL	3.45 ms	288.8 items/sec	SimpleDB offers 1% more durability (99.99999999%) Interface differences: SQLInterface and AttributeValue	$0.8 one-time $0 monthly
SimpleDB	SimpleDB (no change)	35.46 ms	28 items/sec	Data transfer fees incurred by each data access	$0 one-time $5.88 monthly
RDS	MySQL	>1 ms	14359 items/sec	RDS offers .5% more durability (99.5%)	$0.7 one-time $0 monthly
RDS	RDS (no change)	13 ms	14172 items/sec	Data transfer fees incurred by each data access	$0 one-time $328.2 monthly

considers the data transfer costs and the performance of the remote accesses. Given this information the user can decide where to move the data. For example, data currently being stored in S3 can be moved to a local NFS if durability is not a requirement. The available space is also very limited (10 GB), so a local deployment needs to expand the current storage capacity. Other options, such as Hadoop and GPFS will be specially suitable if the number of concurrent clients increases (here is assumed to be at most 10 for each file/blob). Hadoop is closely related to the Map/Reduce application model; this special access interface differs from a randomly accessible file in UNIX and it must be taken into account. As our last option, part of the data could be archival data that may continue in S3 since the increase in access latency is less important than the durability guarantees. The user would have to continue paying storage fees and data transfer fees (if the data needs to be retrieved).

SimpleDB is a more complex storage service that currently has no local counterpart: the most similar storage system would be a traditional relational database.The migration from SimpleDB to a relational database would imply the modification of the application access to the tables (or domains in SimpleDB). Keeping the data in SimpleDB may be the best course of action, especially if the latency increase does not impact global application performance and the data transfer cost is bearable. Since there are good and free solutions to relational databases RDS can be easily substituted by a local MySQL deployment. In summary, in a migration from a public cloud to a private cloud deployment there are many possible choices for data movement; our application can take the information about the capabilities of each storage system to guide the migration.

In this use case we have assumed "free" local storage services. However, there is no such thing as a free lunch; someone has to pay for it. It is a common case that scientists get financial support from grants and companies to buy their computational resources; the university provides the electricity and real estate. In this situation the scientists' final bill is what we present in Table 4. This does not imply that we believe this to be true always: by modifying the storage cost values in our XML descriptions of the local storage we are able to express different cost models (the same ones from the public cloud providers) and accurately represent each local cluster.

7. EVALUATION

7.1 Extensibility of schema and algorithms

We have already mentioned the extensibility of the XML schema; in this section we present an overview of the cost of extending our algorithms so they are able to process new elements and attributes in the XML descriptions of the cloud provider. We will consider the following example: data durability. Durability is a common requirement and some cloud providers, such as Amazon, provides the actual number (99.99999999% for S3) that each storage service of the Amazon platform was designed to provide. In the simple case, in order for a cloud to meet the user's durability requirement we just have to find this attribute in the StorageAbstraction element and compare it to the user's input. However, some of the cloud providers, such as Windows Azure, do not provide this measurement and it is difficult to compare storage services or to evaluate them. Therefore, we have found the necessity of extending this simple durability requirement evaluation to a more complex one.

In order to better evaluate the durability of two storage services we have chosen to compare the number of replicas and its location (in the same datacenter or multi-region replication). If both storage services have equivalent levels of replication we consider them to have essentially the same durability (while this is not exactly true, we consider it to be a good estimate when we lack more information about the storage service implementation). Every data requirement in our prototype application inherits from the *Requirement* class, whose method *match* is called when evaluating a certain storage service and region combination from a cloud

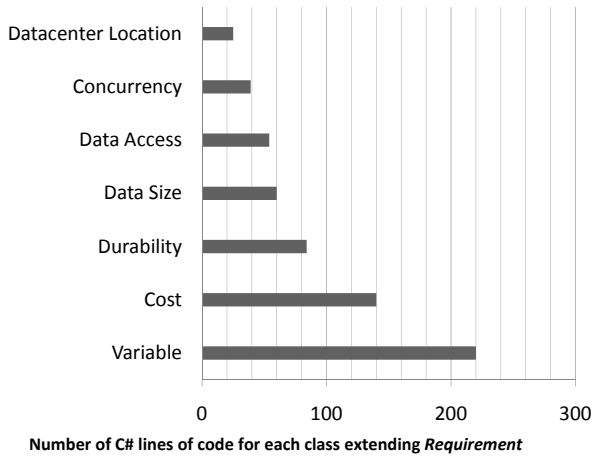

Figure 4: Number of C# lines of code for some of the data requirements' handling code in our application.

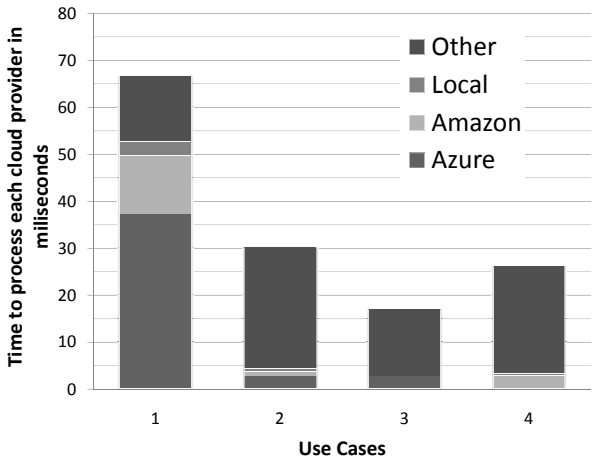

Figure 5: Wallclock application time for each of the use cases presented in Section 6 in miliseconds, divided into time spent processing each cloud provider and input/output processing (Other).

provider. Within this method the programmer has access to the XML document (*CloudProvider* type). The return of this method (*MatchResult* type) tells us if the storage service matches the given requirement and any warnings or errors.

```
class Durability : Requirement
...
   override public Matchresult match(
     CloudProvider c, Matchresult r,
     string storageServiceID, string regionID)
```

We give the numbers of C# lines of code in Figure 4 to give the user a general overview of the programming cost of extending our application. Requirements that involve checking for simple elements and attributes, such as support for concurrency or a certain type of access (random, stream, attribute/value), can be coded in under 50 lines. More complex ones, such as calculating the cost of a certain variable (latency, throughput) require between 140 and 220 lines of code. Our implementation of the *Durability* class was done in 84 lines; we believe that the cost of extensibility is quite low since the size of the code is manageable and it is a one-time effort.

7.2 Use cases

Here we present the performance results of our application for each of the use cases introduced in Section 6. We run each use case 10 times, and Figure 5 shows the average wall clock time. For every use case our application finishes in less than 70 ms. We detail the time it takes to process the XML descriptions of each cloud provider in the same graph. Windows Azure takes longer than Amazon and Local because we were able to include more detailed performance information, thanks to the Azure Scope website. In our second use case (cost savings) we first calculate the cost for each storage service: if it is greater than the current service we skip to the next service on the list. Thus, we do not need to process most of the services. Use case number 3 (performance and cost estimates) focuses exclusively on the Azure cloud, as use case number 4 (Amazon to Eucalyptus) does with the Amazon cloud and the local cluster. The XML files are located in the hard drive; the application time incurred to load these files is included under the *Other* label together with input and output processing.

In the future, running times could become larger due to the following factors: a more detailed description of each service, an increase in the number of requirements processed, and a greater number of cloud providers and regions. An increase of details for the performance section is certainly possible and advisable: we can estimate better variables such as latency and throughput with more data samples from benchmarks. For example, our performance description for Azure table includes latency measurements that vary based on the number of concurrent clients. Additional data can reflect the performance impact of other factors such as entity size, number of entities per table, etc. For our use cases we analyze requirements such as data size, durability, calculate cost and latency, etc. For any given community there will be additional data requirements to process; these requirements, however, would be essentially limited by the storage capabilities described and the number of requirements expressed by users. Finally, every region added will mean more processing for cost and performance calculations; every new cloud provider will have an impact similar to the ones detailed in Figure 5. Given the current performance results, we feel confident that our approach can accommodate these changes without impacting the user experience. For future work we are planning to include these software as a library that can be used by a higher level data management software to further automate the processing of data by cloud applications.

8. CONCLUSION

We have presented in this paper an automated approach to the selection of cloud storage services that can meet the user's requirements. First, we have defined an XML schema that guides the description of the different capabilities of cloud storage systems. This schema is based on the documentation of different storage services and our experiences with cloud computing. We use this XML schema to provide descriptions for the storage services of Amazon, Azure and

some software that is commonly deployed in local clusters (NFS, Hadoop, MySQL, etc.). Second, we have developed an application that processes these XML descriptions and attempts to match common data requirements from users to them. Our application is also able to provide cost and performance estimates. Both our XML schema and our application can be easily extended to describe new features and process new requirements. Finally, we have shown that, in less than 70 ms, we are able to recommend storage services for a cloud application, estimate possible cost savings and tradeoffs by switching storage services, estimate storage costs and performance under different growth scenarios, and provide information to assist in the migration of cloud applications to private cloud deployments.

9. REFERENCES

[1] Amazon. Amazon elastic compute cloud. http://aws.amazon.com/ec2/, 2008.

[2] R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, and R. Tewari. Cloud analytics: do we really need to reinvent the storage stack? In *Workshop on Hot topics in cloud computing*, page 15, San Diego, 2009. USENIX Association.

[3] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J. Schopf, M. Viljoen, and A. Wilson. GLUE Schema Specification-Version 1.2. *OGF GLUE-WG*, 2007.

[4] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tueckeb. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23(3):187–200, July 2000.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing, 2009.

[6] S. Das, D. Agrawal, and A. E. Abbadi. ElasTraS: an elastic transactional data store in the cloud. In *Workshop on Hot topics in cloud computing*, page 7, San Diego, 2009. USENIX Association.

[7] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The montage example. In *International Conference for High Performance Computing, Networking, Storage and Analysis. SC.*, pages 1 –12, Nov. 2008.

[8] Girish Subramanian and Y. Simmhan. Tools for Genome Haplotyping in the Windows Azure Cloud. In *Microsoft Research eScience Workshop*, Microsoft Research, 2009.

[9] A. J. G. Hey and A. E. Trefethen. *The Data Deluge: An e-Science Perspective*, pages 809–824. July 2003.

[10] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey. Early observations on the performance of Windows Azure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 367–376. ACM, 2010.

[11] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. Anderson. Cost-benefit analysis of cloud computing versus desktop grids. In *IEEE International Symposium on Parallel Distributed Processing*, pages 1 –12, May 2009.

[12] Microsoft. Microsoft Windows Azure Platform. http://www.microsoft.com/windowsazure/, 2008.

[13] Microsoft Extreme Computing Group. Azure Scope. http://azurescope.cloudapp.net/, 2010.

[14] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Elastic management of cluster-based services in the cloud. *In Proceedings of the Sixth IEEE International Conference on Autonomic Computing (ICAC'09)*, pages 19–24, 2009.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. *CCGRID*, pages 124–131, 2009.

[16] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for science grids: a viable solution? In *In Proceedings of the 2008 International ACM Symposium on High Performance Parallel and Distributed Computing*, pages 55–64, 2008.

[17] S. Patil, G. A. Gibson, G. R. Ganger, J. Lopez, M. Polte, W. Tantisiroj, and L. Xiao. In search of an API for scalable file systems: under the table or above it? In *Workshop on Hot topics in cloud computing*, page 13, San Diego, 2009. USENIX Association.

[18] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, pages 28–31, 1998.

[19] A. Ruiz-Alvarez and M. Humphrey. Xml descriptions of amazon, azure and a local cluster. http://www.cs.virginia.edu/˜ ar5je/SCPaper.html.

[20] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Conference on File and Storage Technologies (FAST)*, pages 231–244, May 2002.

[21] Y. Simmhan, C. V. Ingen, G. Subramanian, and J. Li. Bridging the Gap between the Cloud and an eScience Application Platform. In *In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, pages 474–481, 2010.

[22] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, Sept. 2009.

[23] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the Globus Data Grid. In *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'01)*, pages 106–113, Brisbane, Qld. , Australia, May 2001.