APPLICATION PORTABILITY FOR PUBLIC AND PRIVATE CLOUDS

Manohar Jonnalagedda, Michael C. Jaeger, Uwe Hohenstein and Gerald Kaefer
Corporate Research and Technologies, Siemens AG, D-80200 Munich, Germany

Keywords: Cloud computing, Portability, Application symmetry, Application architecture, Hybrid cloud.

Abstract: With cloud computing, the general idea is to deploy applications and services in the cloud, at some cloud provider's facilities. But as with traditional software applications, business demands still exist including legal, privacy, cost and technical issues. These demands can prohibit the deployment of the entire software in a cloud provider space. Thus, some cases demand for a hybrid deployment where the application is split into one part that resides on premises and into another part that is deployed to the cloud provider facilities.
Nevertheless, individual components could be suitable for a deployment in the cloud. Thus, an important characteristic for cloud computing is portability of components: software should be ready for being deployed on-premises, in a provider cloud or in a hybrid (mixed) setup. The goal is to provide flexibility to this regard for leveraging the advantages of cloud computing. This paper introduces design considerations for developing a hybrid application, in terms of software architecture, communication and security between modules. We give recent trends and recommendations on how to solve these issues so as to achieve portability of the components.

1 INTRODUCTION

The cloud is used as a metaphor for the Internet, due to the depiction of the Internet in cloud forms. The metaphor results from the location transparency (cf. Reference Model for Open Distributed Processing (ISO/IEC, 1996)) that is a characteristic of the Internet today: the physical location of a service or node in the Internet is hidden, software and user deal with logical locations posed by IPv4 or IPv6 addresses. The cloud represents a set of services that provide computing, networking, and software capabilities without a clear physical location. Cloud computing refers to the provision of these services in three provisioning categories:

1. Software-as-a-Service (SaaS): giving a user access to software which runs on the cloud. All computation taking place on the cloud.
2. Platform-as-a-Service (PaaS): giving developers access to platforms and frameworks that allow them to leverage the computing power of the cloud and to develop own applications that run efficiently on the cloud.
3. Infrastructure-as-a-Service (IaaS): giving access to highly scalable and elastic computing, network or storage resources. This allows the developer to build applications that can scale (seamlessly) to many thousands of users, without having to buy the hardware for it.

While the first category is end-user oriented, it is IaaS and PaaS that have made cloud computing attractive to the software industry (Gartner Inc., 2010). Generally, the global pressure to reduce capital expenditures (CAPEX) and operational expenditures (OPEX) drives the industry to the adoption of rationalisation and automation. Cloud computing has gained a lot of popularity because of its promise to lower costs, which can be seen as the main driver here. Customers expect to pay less because of economy of scale: large providers invest highly in large and modern data centres which allow operations of servers at a very low cost. Customers expect to benefit from this cost reduction. Other important arguments in favour of cloud computing have been discussed by (Armbrust et al., 2009):

- **Flexible Contracts.** Contracts from existing offerings allow users to resign after a short period of time. Thus the customer does not have to plan a long term commitment to a provider.
- **Lower Administration.** Because infrastructure and platforms are basically part of the offering of the provider, their administration is also per-

formed by the provider.

- **Cloud Grade SLA.** At the time of this writing, large providers offer usually 99.9% (or 99.95%) availability of the services. Such availability is not given per se with individual deployments and one must carefully select and deploy a provisioning platform to actually offer 99.9%.
- **Towards Strategic Technology.** Cloud computing platforms target the market of classic application servers. As such, the classic application server technology has been developed further to serve as platform for cloud computing.
- **On Demand Self Service.** The cloud offerings have the advantage that they provide the customer with a self-service point in the Internet that is open 24h a day. The customer can initiate the product use at virtually any time of the day.

In summary, several drivers can exist for a business to use cloud services and it basically depends on the application case if either one or more drivers apply for the own business to move activities to the cloud. Companies are therefore looking to gain all the benefits by developing software for the cloud.

The portability of cloud applications has two main drivers: firstly, an organization plans to take benefit from the cloud computing offerings, and secondly, restrictions prevent to run the application entirely or partially in the provider cloud space. From a business point of view, an ideal goal is to decide *at deployment time*, depending on business demands, if the software shall run in the provider cloud, in a private cloud on premises or in a hybrid (mixed) setup, so as to have an optimal trade-off between privacy and economic use of cloud resources. In this paper, *application portability* refers to the ease with which modules of an application can be split and deployed on on-premises or cloud platforms, ending up in a mixed setup. Figure 1 depicts this idea.

We introduce the engineering issues for application portability, which refers to the flexibility at deployment regarding the location and the partitioning of the application. Furthermore we explain trends in the field and possible technical solutions. The paper continues with Section 2 which explains the business drivers and motivates this kind of portability. Section 3 describes the issues with portability while Section 4 outlines possible solutions. In Section 5 we look at related work, and Section 6 concludes this work.

2 MOTIVATION

Two basic cases for portability exist: one is a technical issue, as applications or parts of it can be moved from a private cloud which runs on-premises to a provider cloud to compensate for capacity shortage of own resources. This portability is required during runtime. Another kind of portability is a business issue: the deployment of an application shall take advantage from CAPEX and OPEX reduction promised by cloud computing offerings. Such a portability is required during deployment time.

The focus of this paper lies in the portability driven by business demands. There are naturally reasons why cloud computing may not be preferred. These may be privacy issues, legal or compliance, or operational-related, which could result from responsibilities and structures of a large organization. Moving business-critical information onto the public cloud is a matter of reticence to many big organizations as, if on the cloud, this information can physically lie in any continent, which may go against local laws and policies, or even make the information liable to laws in another country (see the Patriot Act (The Library of Congress / Thomas, 2001)).

Furthermore, there are legal obligations that companies must fulfill for particular information.

This means that the company or organization must actually store such information on-premises requested by governmental law. In a different case, some components can take benefit from a deployment in a provider cloud while other components are well located on existing infrastructure, not causing additional cost at the provider.

Finally, some applications may not be suited for the cloud at all, not for technical reasons, but the reason that an organization does not intend to reveal that such application runs in general (this category is naturally not the focus of the paper).

For companies with legal obligation on the provision of data but which do want to benefit from the cloud, technical and engineering questions present themselves. Examples for such companies or organizations are manifold. Consider an image management application which allows for uploading images taken with different devices to be uploaded onto a cloud computing application that performs image calculations. The images are stored somewhere in the world, but do not contain any metadata about users or persons depicted on these images. Imagine a vendor providing such software to different clients:

1. *The local police department* with high-definition cameras to take pictures of various crime scenes and other surveillance pictures, it is interested in

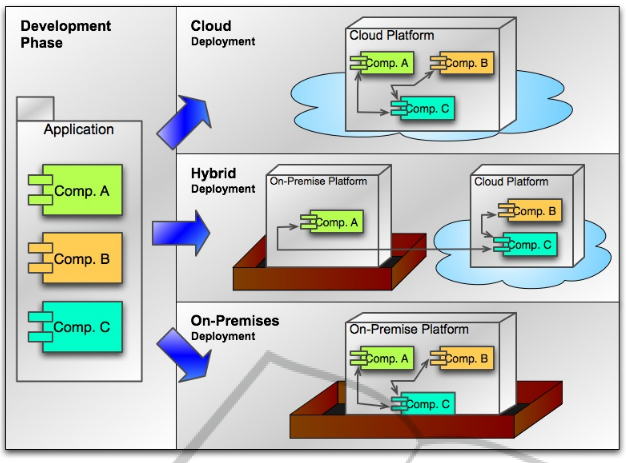

Figure 1: Development and Deployment Scheme.

keeping high-quality pictures in a private setup. It goes without saying that the police would like the portal to run on-premises.

2. *A medical diagnostics application* that has some high-tech equipment which takes pictures related to different illnesses inside the human body. These images are large in size, and plenty in number. The institute might therefore like to store and process the images themselves on the cloud, although metadata concerning each image, being confidential, would be kept on-premises.
3. *A group of college friends* would like their images of a trip to the Bahamas to be uploaded directly onto a public platform, so that their other friends can also view the pictures. They do not mind where the photos are stored.

So, in the individual cases, different demands for storing the data apply. However, as a software developer and vendor of the image management application, the application should not be developed three times. The business goal is reuse of the application software for all the three cases. An ideal solution for this situation would be to let the business model decide whether the application will run locally, on the cloud or in a hybrid setup. Dividing the application in such a manner brings forward some key engineering issues, which are discussed in the next section. In this case, the location is set at deployment time of the application. This degree of flexibility is not required at run-time.

3 ISSUES OF APPLICATION PORTABILITY

Portability in general, is a well know term. Some of the first notable publications define portability as *A program is portable to the extent that it can be easily moved to a new computing environment with much less effort that it would be required to write it afresh* (Johnson and Ritchie, 1978). Mooney adds the notion of narrowing the set of targets for the program and states accordingly that a program exhibits portability referring to a set of environments, if the cost is lower that a rewrite (Mooney, 1997). For portability with cloud computing we can build upon this definition and distinguish two major cases:

- **Vertical Portability** is the capability of an application intended for on-premises infrastructure to be portable to a provider cloud environment. The restriction is that the application runs on the same technology stack on both platforms (Java or .NET and respective cloud offerings, for example).
- **Horizontal Portability** is the ability to port an application from one technology stack to another, staying at the same level of abstraction but changing the technology provider.

As described in the above example of image management, the emergence of cloud computing brings the need to develop software that runs both on-premises and on the cloud. The business case demands for vertical portability, as it allows for the decision to stay within one technology stack. This presents the following scenarios: if a vendor is catering exclusively to the first or third type of client (police department or the private initiative), the problem is solved by making use of PaaS offerings (Windows Azure (Chappell, 2009), Google App Engine (GAE) (Google Inc, 2008), Heroku (Heroku Inc., 2010) are some examples) which provide frameworks and APIs for developing software. On the other hand, developing on-premises software is a tried and tested technique.

Engineering issues really arise when one wants to develop in a hybrid setup: ideally, we would like to develop the image sharing software only once, and deploy different modules of the application to different platforms, as dictated by the business requirements. In the following, the particular issues are described.

3.1 Application Architecture

Design principles are an important consideration when developing any software. The goal is to build applications which are highly scalable, maintainable and reliable. These requirements are naturally applicable when it comes to developing software for the cloud, but are not mandatory part for traditional application development.

From the above example and other general cases, an important issue is the division of an application into separate, self-contained and loosely coupled modules, which, ideally, should be deployable to different platforms. The age-old recommendation of *component-based* programming is, as we can see, very much applicable in the given scenario. The caveat, however, is that the platforms on which the modules will be hosted can be very different in terms of computer architecture, database management, and access to finer aspects of the underlying OS (cloud applications typically run on virtual hardware). The big challenge, in terms of software design and architecture is, therefore, that of creating modules that are conscious of the environment in which they run (so as to take advantage of the underlying structure), and at the same time independent enough so that their deployment to other platform is effortless.

3.2 The Data Model and Persistence

Cloud computing has brought everybody's eyes a shift in thinking about data that has been taking place in the IT world for the past decade. Traditionally, industrial applications have considered relational database management systems (RDBMS) as the de facto standard for persisting data. These database systems use SQL as the main query language, and are therefore also referred to as SQL databases. They are highly prevalent in the industry: they have the advantage of offering transactional services, a general schema for storing structured data, and indexing mechanisms which optimize queries. They concentrate mainly on consistency.

In recent years, however, many companies have been challenged by the sheer quantity of information stored and accessed over the Internet: the necessity to make their services highly available has been the main driving force. This has seen the development of many No-SQL type of databases, light-weight databases whose main selling point is distributed services and high availability. CouchDB (Apache Software Foundation, 2008), MongoDB (10gen Inc., 2009) and Cassandra (Apache Software Foundation, 2010) are some popular No-SQL databases.

Cloud computing, with its promise of elasticity and availability, is a natural fit for such No-SQL databases. This explains why the first storage solutions have been BLOB (binary large objects) storages emulating a file system to some extent, and table storages. The latter keep a table view without requesting a fixed pre-defined table schema to be defined in advance. Hence, heterogeneous structures can be stored in one table. The main focus of table storages is to offer the 80% of database functionality that typical Web applications require. They sacrifice consistency for the sake of availability and partition tolerance in the sense of Brewer's CAP theorem (Gilbert and Lynch, 2002).

Prominent cloud providers have started their offering with No-SQL DBs. They focussed on table and blob storage products which are advantageous w.r.t. availability and tolerance to partitions. However, in the recent past, providers have also extended their offerings with additional SQL-based RDBMS technology. For instance, Microsoft has announced Azure SQL in last year and Google has announced similar support recently.

3.3 Communication in a Hybrid Deployment

Another important aspect for portability is to ensure security with respect to communication between modules; the security of the cloud platform itself does not involve application symmetry, and generally, business decisions (based on cloud platform security concerns) decide which modules can be deployed on the cloud anyway. Modules can communicate with each other via different layers of the network stack. When developing Web applications, the main grounds for communication is to access and consume application-level data. Communication hence takes place at the application layer. This type of communication privileges data consumption and serving stateless requests.

Secondly, there is a need to communicate at a lower level: we need to be able to access and manage resources at a different level from the applications, and also ensure that incoming and outgoing packets to a certain module are secure (authenticated and encrypted).

4 TECHNIQUES AND RECOMMENDATIONS

The previous section outlined several issues that arise for engineering applications in a mixed cloud setup. The general rationale is that the finished application should be portable enough so that the entire application, or parts of it, can be deployed on a provider cloud with as little effort as possible. Thus, the proposed business goal of Section 2 is to decide at deployment time for a given setup at runtime. [1]

4.1 Software Architecture

For Platform as a Service (PaaS) offerings there are several paradigmatic characteristics for cloud use. One of the standard paradigms is *divide and conquer*, which has also been adopted in Google's MapReduce framework for efficiently processing large problem sets (Dean and Ghemawat, 2008). Cloud computing offers horizontal scalability: adding more working units to increase capacity, the ideal goal being to achieve almost infinite scalability. The common consensus seems to be that good cloud applications should be designed with awareness of horizontal scalability. If the problem can be partitioned, then solving the problem can be infinitely spread among a set of working units.

Secondly, with the proliferation of development platforms for the cloud, each of them privileges certain building blocks for writing software optimally for the cloud. The Windows Azure platform, for example, provides the concept of roles: self-contained entities implementing an elementary unit of processing capability. In general, it is a good practice to implement elementary units as opposed to classic large blocks of functionality: the latter would lead to large units in the software system, thereby not allowing for partitioning of the problem and not taking advantage of the horizontal scalability of the cloud. The partition of the application in many elementary units is a necessary condition for the split deployment of the application. For application portability, this design shows the advantage that the application has a high degree of partitioning opportunities.

In conjunction with elementary application units, the second larger design issue for cloud computing applications is decoupling of application components. For horizontal scalability, statelessness is preferred, as state at a certain key point of the application can block the command flow, hence losing all the advantages of elasticity and availability. Furthermore, strong coupling would prevent the application from the ability to scale dynamically. In order to elastically add working entities at runtime, the application architecture must show decoupled units of stages where problem parts can be processed in an independent manner. For splitting the application into two deployment locations, the decoupling allows for the use of service buses or other communication mechanisms in order to overcome the distance between the two parts (described in section 4.3).

Additionally, an aspect of software development, applying specifically to cloud computing, is the reduction of service calls and transactions. Cloud computing being pay-as-you-go, money can be saved by minimizing calls to services, database servers, other storage offerings, number of CPU cycles, etc. Although this effect is not strong when using only a few servers, savings can be significant when the application scales to hundreds or thousands of servers. For Internet programming, caching is a concept used to increase performance by decreasing the number of direct queries of the database: The idea is to cache the result of common queries in memory, and, for example, responding with this result in case such queries are made. With respect to cloud computing, such techniques also help save money, as they reduce storage access calls. Memcache of the Google App Engine environment (Google Inc, 2003) helps in achieving caching for saving calls to the storage system or for preventing redundant data transfer to external application.

To summarize, horizontal scaling of elementary working units in the staging of an application is an elementary architectural consideration in order to leverage the cloud resources in an optimal way. This paradigm also enables the slip deployment of the application in a hybrid cloud setup.

[1]Note: as outlined in Section 2 as a technical motivation, the freedom to migrate services into or off the provider during run-time might be also required but has different motivations. For example, an organisation could migrate services to the provider cloud in the case that the on-premises infrastructure does not have any further capacity to host applications (known as cloud-bursting). However, this degree of dynamic migration is not the scope of this work. Therefore, the following trends and recommendations *do not* cover dynamic migration but consider flexibility of application deployment.

4.2 Data Models

Data representation is a key part of any application. In this section, we look at a few techniques and trends as to how different models can be used so as to achieve application portability. In this section we look at schemes for storing structured data.

As explained in section 3.2, Web application developers and cloud providers privilege distributed key-value stores for persisting structured data. The advantages of such storages are described in the given section. However, there are, at the moment, no standardized database or storage products that run both on the cloud, and can be installed on-premises. For example, the table storage service as part of the Microsoft Azure offering is not compatible with Amazon SimpleDB, Moreover, the industry still largely prefers using the tried and tested technique of traditional RDBMSs (which explains why main cloud providers are also starting to provide SQL solutions), so there is a need to design data models so that they can run on many different platforms.

With this in perspective, Object-Relational Mappers (ORMs) offer an abstraction for representing the data of the database at the programming level. Such mappers were initially developed in order to connect relational databases to object-oriented programming languages (Ambler, 2002). The development of Web application stacks and tools has turned this abstraction into a good practice. These persistence abstractions present a uniform access interface to many different database types. When developing hybrid applications, structured data can be represented using these mappers: in the next subsection we explain how JDO and DataNucleus (DataNucleus, 2008) can be used to develop modules that could run either on Google App Engine or locally.

Google App Engine's Java runtime uses a Jetty servlet container and the Java Servlet Standard for Web applications. Its datastore, based on BigTable, supports standard Java interfaces such as JDO and JPA, which are themselves implemented using the DataNucleus Access Platform. DataNucleus supports many RDBMS systems, along with Web-based storage systems such as Amazon S3, Hadoop HBase and Google App Engine.

To make use of the JDO mappings, a developer needs to import the relevant packages and add annotations (`@Persistent`) to classes and attributes that he would like to be persisted. There are some additional configuration properties he needs to define in a `datanucleus.properties` file, which include the database which will be used for the application.

For an application that is to deployed on Google AppEngine, the configuration file will contain AppEngine specific information. In order to make it possible to deploy the same application on a MySQL database, the idea is to create another configuration file, which this time contains properties related to this database.

At the code level, the properties file is passed on as an argument to the `JDOHelper.getPersistenceManagerFactory` method. The value can be changed depending on which platform the

Abstractions on top of data storage schemes present an interesting solution to achieving portability of structured data. This follows the old adage: *"All problems in computer science can be solved by another level of indirection"*. More than being useful for accessing multiple types of databases, they have the already known advantage of helping in programming applications. ORM implementations such as Hibernate or EclipseLink are popular frameworks for facilitating the object/relational mapping with different database servers. As a third benefit, they can also be useful for transferring data from one platform to the other: one just needs to extract the data from one store and store it into an other, without having to bother about the database specifications at a high-level.

There are specific cases in which specific database operations are required, and for which it is wiser to directly address the data store without using the mapper framework. JDO allows bypassing the framework to query the underlying database directly, and even to execute stored procedures. These features are however very specific to the type of RDBMS used, and therefore not subject to abstraction. It is hence important to identify such requirements during the design phase of the application; in case specific operations are required, the application might not be suitable for running on multiple databases. In order to run such software in a hybrid setup, it is recommended to use identical underlying database technology.

4.3 Communication and Security

4.3.1 Application layer

Communication at the application layer refers to the availability of Web services and other API technology which allow data to be absorbed from or inserted into a service. Communication via Web services is one of the simplest forms of communication, and useful when developing highly available services, as it makes the software easier to use for clients and customers. As more and more data becomes available on the Internet, there are ever more ways to combine different pieces of information and present them in novel ways: many popular Web applications therefore provide REST APIs for clients, customers or users to access data, process and use it in new, imaginative ways. The Programmable Web (Musser, 2010) provides a list of applications offering such APIs.

This is relevant to application portability as we can imagine different modules of an application providing access to data through Web services. Moreover:

- Cloud services are accessible on the Internet, and there is a need for appropriate security measures at this level. Web services' security standards and identity federation are required building blocks for design.
- Providing a data access API generally requires little overhead on creating the application itself, and provides an easy-to-use, generalized interface to many potential clients without having to set up specific protocols with each of them.
- Two sites communicating uniquely via calls to Web services abstract the network layer beneath them, and alleviate the need for developing complex communication protocols. Any module of the application becomes portable: wherever it resides, it can be accessed as long as it provides an interface to absorb data.

One of the main security issues in terms of communication via Web services is authorization; how one gives an authorized user access to the API, and prevent non-authorized users from accessing it. Design of secure Web services is a difficult topic and an active subject of research and discussion (Fernandez, 2004; OAuth Core Workgroup, 2009; OASIS Consortium, 2002; Fielding, 2000). Many Web applications allow third-party login: it is possible to get access to an API/service by using one's Google credentials, for example. This is a security hazard, as applications could get access to unrelated login information. A better idea is to hand out tokens to the third-party website in question. This token, received from a trusted source, is accepted by the website, and access to data is thereby granted. This technique is named OAuth (an open standard) (OAuth Core Workgroup, 2009). Microsoft AppFabric's ACS (Brown, 2009) is an implementation of this standard and allows developers to write secure APIs that accept ACS tokens.

4.3.2 Network Layer

While the use of Web services for data access may be simple and elegant, there are certain types of applications (chat applications and collaboration tools, for example) for which such interfaces have too much overhead in communication. Transferring offline data from one platform to another can also be a bottleneck in terms of performance.

Large corporations having centers worldwide study ways of connecting them in secure ways. Enterprise Service Bus is a prevalent concept among them. Microsoft AppFabric's Service Bus aims to take the concept onto the cloud, enabling applications residing on the cloud to work with those on other platforms.

The service bus concept is one way of looking at the communication issue between different platforms. Another way is to attempt to bring both the cloud and on-premises platforms into a single virtual network. Amazon's VPC (Amazon.com Inc., 2009) is a product which goes in this direction: the on-premises network is connected to the Amazon Cloud resources via a secure VPN connection over the internet. This gives the illusion of a giant (seemingly infinite) enterprise network, with all the advantages of cloud computing (scalability, availability, computation). VPN connections as links between cloud and on-premises software have also been studied elsewhere. Wolinsky et al. (Wolinsky et al., 2009) studied network architectures in which network clients can be seamlessly added and removed from a virtual network. They conclude in their study that virtual networks provide excellent isolation, and good performance over Wide Area Networks.

5 RELATED WORK

Relevant cloud providers continuously work to provide seamless integration of cloud computing with on-premises technology. Amazon, Microsoft, and Google have all released, or are in the process of releasing, SQL related technologies to bridge the gap (Amazon RDS, SQL Azure and Google App Engine for Business, respectively).

On the other hand, people have come to realize the advantages on No-SQL databases and storage systems, given their success with Web 2.0 applications and social networks such as Facebook and Flickr. Many of these storage solutions themselves offer integration with the cloud. CouchDB, for instance is used by the Ubuntu One cloud provider as their distributed document storage software. Ubuntu users can use CouchDB to store data locally, and using CouchDB's replication capability, synchronize their documents over different computers or over the secure storage on the Ubuntu One cloud. The replication functionality comes into play for intensive cloud computing applications when dealing with several server nodes: efficient replication between server nodes allows for vertical scalability, meaning adding extra CouchDB nodes for improving overall capability. ThruDB is another relevant concept, especially for application portability. ThruDB offers a key value store provided by own implementation. In addition, it allows for setting Amazon S3 as storage implementation replacing the own functionality.

There are also many open-source solutions which attempt to grant access to No-SQL databases as well as cloud offerings. The DataNucleus Access Platform, as mentioned in the above section, is a solution that caters to various types of database servers or services, including Google App Engine and Amazon S3. Recent projects also include access to the MongoDB datastore. AppScale allows users to deploy Google App Engine applications on-premises, as well as on Amazon EC2. It allows mappings to many different datastores, such as Hypertable, HDFS, Cassandra, MongoDB.

Another general direction of research has been to bring the cloud paradigm to on-premises environments. Eucalyptus and Ubuntu Private Cloud are open-source frontrunners in this field. The advantage of such technology is that it makes it easier for distributed algorithms and patterns to be portable. A Hadoop-based Mapreduce application can be therefore run on-premises or on Amazon's Elastic Mapreduce.

Section 3 has already mentioned two references that discuss portability of the programming language C, its compiler and the UNIX system (Johnson and Ritchie, 1978), which was a fundamental effort, and the issues of portability from a software development perspective (Mooney, 1997). The general recommendation that can be drawn from these writings are:

- Reduce the amount of platform-related code, which is not focus of this work as the problem posed by the business case actually allowed to stay within one platform.
- Isolate dependencies which is generally a well established practice of programming. For the context of the problem of this work, this would translate to dependencies to the actual environment which has been discussed with the data model and storage provider issues.
- Externalize interfaces, which means that access to any given component inside or outside would be made explicit by using a contract rationale. This issue has also been covered by modern programming languages offering separate constructs for defining interfaces.

These recommendations have also led to design patterns: Most notable example is the Wrapper Facade (Schmidt et al., 2000) which proposes to isolate system- or platform calls to a single point of entry. Thus, porting the software to another system or platform will require changes to only a single location in the software. This applies also to the UNIX design featuring a hardware specific kernel, as described by Johnson and Ritchie (Johnson and Ritchie, 1978). Transferred to the vertical portability in the cloud case, this work has focussed on the data provider as specific platform calls.

Referring to cloud computing, the issue of portability is recent and not much covered in literature – as current offerings are still in the phase of orientation and sharpening of the focus. The main issue for ensuring this kind of portability is the fear of vendor lock in. Vambenepee states that many issues besides API compatibility are currently not solved, such as migrating the data from one storage to another, compatibility to billing and metering, error handling and logging or just administration(Vambenepee, 2009). However, these issues are not the focus of this work, because:

- The concept of vertical portability allows for using the same technology stack when deploying the application into the provider cloud.
- The business scenario decides on the deployment at the deployment time. Thus, once an application is deployed, it can keep its location. Thus, the migration of data from existing systems is not an issue in this scenario.

6 CONCLUSIONS

In this paper, we have discussed some of the trends and recommendations when it comes to the portability of application between the provider cloud and infrastructure on premises. It is important to note that we focussed on portability (flexibility in platform choice) at deployment time, as opposed to dynamic portability. We identified software architecture, data model, communication and security as key issues when it comes to developing such applications.

From the previously discussed points, we can summarize recommendations for each category of issues. In terms of software architecture, a developer should concentrate on:

- Build APIs and Web services which allow access to application data. This standardizes access to data and moves the application into a SaaS world (which is the overwhelming trend of software applications nowadays).
- Consider using patterns like MapReduce to distribute computation over multiple nodes and to diminish the overall time taken for batch-oriented

tasks.

- Consider deploying high availability requiring modules, such as Web applications, entirely on the cloud. Use Web application PaaS offers such as Google App Engine, Heroku or CloudBees for such modules, as they scale (semi-)automatically.

With respect to data models, it is important to:

- Note that with simple No-SQL storage service offerings, *join* operators in queries are sometimes not supported. Thus, the data model should support a convenient handling of join-like queries in program code.
- Consider using No-SQL databases on-premises for achieving similar scalability on local infrastructure as well.
- Design the logic of the application to be agnostic about a particular database software. Avoid writing queries which are specific to a certain database product or software in the source code. Use ORMs or abstraction layers to achieve this task.

Finally, for secure and authentication communication, it is important to:

- Restrict access to data by using open standards and claims-based authentication. Use open standards such as OpenID and OAuth to achieve this.
- Consider using service bus technology for connecting cloud and on-premises infrastructure.

Cloud computing, after the initial hype, has reached the first steps with many big industrial players actively looking for cloud solutions. Portability of applications between provider and private clouds is a very important issue for them, and we expect to see many new solutions and ideas in this field in the coming months. The presented trends and recommendations show that this desired flexibility can be achieved for some cases with patterns and programming paradigms. Also, the available technology offered by the major vendors in the cloud computing business confirms this possibility and provides API and frameworks for a distributed application setup.

REFERENCES

10gen Inc. (2009). Mongodb - scalable, high-performance, open source, document-oriented database. Available from http://www.mongodb.org/.

Amazon.com Inc. (2009). Amazon virtual private cloud (amazon vpc). http://aws.amazon.com/vpc/.

Ambler, S. W. (2002). O/r mapping in detail. http://www.agiledata.org/essays/mappingObjects.html.

Apache Software Foundation (2008). Couchdb project. http://couchdb.apache.org/.

Apache Software Foundation (2010). Cassandra project. http://cassandra.apache.org/.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley.

Brown, K. (2009). A developer's guide to access control in windows azure platform appfabric. White paper, Microsoft and Pluralsight. Available online (35 pages).

Chappell, D. (2009). Introducing windows azure. White paper, Microsoft and Chappell Associates. Available online (25 pages).

DataNucleus (2008). Datanucleus project. http://www.datanucleus.org.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Fernandez, E. B. (2004). Two patterns for web services security. In *Proceedings of the Int. Symposium on Web Services and Applications, Las Vegas*.

Fielding, R. T. (2000). *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.

Gartner Inc. (2010). Gartner top 10 strategic technologies 2010. http://www.gartner.com/it/page.jsp?id=1210613.

Gilbert, S. and Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent available partition-tolerant web services. In *In ACM SIGACT News*.

Google Inc (2003). Memcache. http://memcached.org/.

Google Inc (2008). Google app engine - run your web apps on google's infrastructure. http://code.google.com/appengine/.

Heroku Inc. (2010). Heroku : Ruby cloud platform as a service. http://heroku.com/.

ISO/IEC (1996). ITU.TS Recommendation X.902 — ISO/IEC 10746-1: Open Distributed Processing Reference Model - Part 1: Overview.

Johnson, S. C. and Ritchie, D. M. (1978). Portability of c programs and the unix system. *Bell System Tech J*, 57:2021–2048.

Mooney, J. D. (1997). Bringing portability to the software process.

Musser, J. (2010). Programmable web: Keeping you up to date with apis, mashups and the web as platform. http://www.programmableweb.com/.

OASIS Consortium (2002). Saml — security assertion markup language. http://saml.xml.org/.

OAuth Core Workgroup (2009). Oauth core 1.0. http://oauth.net/core/1.0/.

Schmidt, D. C., Stal, M., Rohnert, H., and Buschmann, F. (2000). *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Wiley, Chichester, UK.

The Library of Congress / Thomas (2001). Uniting and strengthening america by providing appropriate tools required to intercept and obstruct terrorism (usa patriot act). http://thomas.loc.gov/cgi-bin/bdquery/z?d107:h.r.03162:.

Vambenepee, W. (2009). The reality on cloud portability. Available from http://www.sdtimes.com/link/33502.

Wolinsky, D. I., Liu, Y., Juste, P. S., Venkatasubramanian, G., and Figueiredo, R. J. O. (2009). On the design of scalable, self-configuring virtual networks. In *SC*. ACM.