



A Comparison of Public Cloud Platforms

Microsoft Azure and Google App Engine

James W Smith

StACC (pronounced like 'stack') is a research collaboration launched in April 2009 focusing on research in the important new area of cloud computing. Unique in the UK, StACC aims to become an international centre of excellence for research and teaching in cloud computing and will provide advice and information to businesses interested in using cloud-based services.

More information available at: www.cs.st-andrews.ac.uk/stacc

A large, abstract graphic composed of overlapping, semi-transparent blue and grey geometric shapes, resembling a stylized landscape or a series of connected planes. The text "SICSA DemoFEST '09" is overlaid on the right side of this graphic.

SICSA DemoFEST '09

Executive Summary

This document discusses two Cloud Computing platforms from leading industry corporations Google and Microsoft. An overview of each platform, its technologies, limitations and procedures is presented. This is followed by a comparison of the platforms. This comparison is based around StACC's experiences in writing applications for each platform and comparing the real world development experience.

1. Introduction

Cloud Computing has been widely tipped to be the new forefront of innovation in Computer Science. Cloud based systems have proven themselves to have numerous advantages when compared with traditional systems, yet with enough challenges to overcome for Computer Science researchers [7]. Cloud Computing based systems are ones which are said to be accessible from any internet enabled device anywhere in the world, without preference of operating systems or browser [5]. To run a Cloud Computing system the requirements of that system must be considered and compared to the available resources of the organisation.

One can build and customise a private cloud, which first requires purchasing hardware and then installing Cloud platform solutions such as the open source package Eucalyptus (which we at St Andrews have used and recommend) [6], and then tailoring it to the specification. This is, of course, a luxury in financial terms so the alternative may be to turn to a Public Cloud provider.

Information Technology corporations such as Amazon.com, Microsoft and Google have released their own Cloud Computing platforms in recent years which can run applications in the remote Cloud environment. Virtualisation technology allows these platforms to appear to users as a single, dedicated machine in which they can run their applications. In reality these platforms are massive systems with multi-tenancy, built in data redundancy and always on connectivity, allowing applications to be accessed from any web-enable device, anywhere in the world. There may also be economic benefits for end users. The subscription or "utility" model of billing for cloud computing allows user to pay for only what they actually use, rather than an initial capital expenditure for equipment which may not be fully utilised [1].

There are however a number of criticisms which have been levelled at cloud computing and in particular public clouds. These systems remove the responsibility of the physical hardware from the user's hands. If users do not own the systems on which their data is stored, then possible security and integrity issues are raised. Geopolitical problems can surface if the cloud being used is not within national borders of a user's country of residence, for example storing sensitive data for research on a public cloud which is not hosted within the UK borders may be in violation of the United Kingdom Data Protection Act. It is therefore important that when agreeing to use a public cloud the following issues are investigated and concluded to the user's satisfaction:

- Data redundancy
- Data location
- Fault Tolerance

- Fault recovery time
- Long term viability

Resolving these issues with the vendor and obtaining Service Level Agreements (SLA) agreeing standards of service will go a long way to ensuring that the user has a positive experience in the Cloud. A discussion of two of the largest cloud computing platforms follows; Google App Engine and the current developer preview of Microsoft Azure.

2. Microsoft Azure

Microsoft Azure is a platform as a service (PaaS) solution for cloud computing. The platform provides a means to run applications on Microsoft Datacenters, and provides a SDK to develop these applications. Azure is a foundation upon which applications can be provided as Software as a Service (SaaS), for flexible, scalable, always-on systems. Windows Azure is the operating system which manages the Microsoft datacenters, controlling the hardware, the virtualisation techniques and dealing with the running and management of applications. There are three components which make up the Azure OS; the Fabric Control, storage mechanisms and the Compute Service [2].

2.1 Compute

The Compute service is the component which allows Azure to host user applications and cater to large numbers of simultaneous users. To do this, Azure can run many instances of the same application on different Virtual Machines (VM) within its data centre. Each user is provided its own VM and a copy of an instance of the application by the Microsoft cloud hypervisor. Using this procedure, the developer does not need to upload their own VM image, but only their application and the required configuration. The Azure platform will create the necessary number of instances required by user demand and maintain its own copy of Windows Azure OS to run in each VM.

There are two types of Windows Azure OS VM instances available to developers; Web role and Worker role. A Web role instance allows connections from the outside world via HTTP (*Hyper Text Transfer Protocol*) and HTTPS (*HTTP Secure*) requests. Using multiple instances of this type is the technique used by Azure to deal with simultaneous client requests. These instances are stateless and loosely coupled, so that any client-specific state must be written to the provided storage mechanisms so that they may be read by any different instances for future client requests. Clients are not guaranteed to have the same instance of Web role service multiple requests, so this architecture will support cross-instance state for multiple unique clients.

Worker roles cannot be accessed by outside requests like Web roles, as they do not have the functionality to accept any incoming requests. Instead, worker roles are tasked by application request in a similar manner to batch job processing. Any combination of Web and Worker roles can be used to create Windows Azure applications, particularly as both types can be spawned on demand to deal with additional workload. All instances are controlled using the Azure API to speak to the OS component Fabric Controller.

2.2 Fabric Controller

The Fabric Controller is the part of the Azure Cloud Operating System which is used to regulate the hosting environment by pooling individual computer hardware into a usable and stable network. The Fabric Controller is able to automatically manage load balancing, data replication and other tasks. This ensures that application developers can concentrate on the features of their programs rather than on the configuration and settings of the platform or hardware. Alongside monitoring running applications, the Fabric Controller can also patch the version of Windows OS running in the Azure VM's.

To do its job, the fabric controller requires a configuration file for each application which provides an XML-based description of what the application needs in order to operate. This file contains information detailing the number of instances of each type of role, Web or Worker that the fabric control should instantiate. From then on these instances will be monitored and repaired if necessary. The Fabric controller will then decide on which physical machine each individual VM should run on in order to balance and utilise resources.

The fabric also attempts to intelligently place application instances on physical hardware so that an entire set of instances cannot be taken down by a single failure to a switch or similar piece of network infrastructure. The data centre which hosts the fabric controller is divided into fault domains, which allows the fabric to decide how to place application instances. A fault domain is defined by Microsoft as a set of hardware systems which could be affected by failure to another single piece of hardware. As the storage mechanisms provide by Azure are considered by the Fabric controller to be just another application, this fault domain protection mechanism also applies to storage units and their redundant replicas.

2.3 Storage

Azure provides three different types of storage that can be used by applications. The first and simplest of those are called "Blobs" (*Binary Large Objects*), which are items of binary data, such as a media file. They have a simple storage hierarchy; each storage account on the Azure system has one or more *containers* which hold one or more blobs. Each blob can be up quite large in size, up to 50GBs and alongside the binary data can contain meta-data to describe the contents of the blob. This allows applications to store and access data in a very familiar way, by first creating containers (traditionally *folders*), then individual blobs to store in them (traditionally *files*). Each storage account, container and blob are given their own unique identifiers, so creating Uniform Resource Identifiers (*URIs*) for accessing each item is a simple process.

Each blob can be further subdivided into sub-blocks sections so that if a failure during transmission of a blob occurs then re-transmission can resume with the most recent uploaded sub-block section rather than from the beginning. This also means that blobs can be updated in a block-by-block basis, reducing bandwidth costs. In the event of simultaneous block write operations being conducted by more than one client Azure will default to a first-come basis. This means that the first operation to complete will be committed and any subsequent write operations which were initialised before the first write was finished will be discarded. This is a prime example of an optimistic concurrency model.

The next, more structured unit are “Tables”. Not to be confused with relational tables, Azure storage tables are made up of entities which have properties. An entity contains zero or more properties, each with a name, type and value. They are not accessed using SQL, but instead by using the conventions defined by the ADO.NET data services. The advantages of this approach is that data can be stored across many machines, allowing *scale-out storage* which is something that is difficult to achieve effectively using traditional relational databases. These tables enforce no schema and their properties can be different at different times depending upon what is stored in it. There are no additional restrictions for using this system; each table can contain billions of entities and terabytes of data. Traditional relational databases would require larger and more powerful machines to handle this level of data. For large numbers of simultaneous users scaling *out* is usually more advantageous than scaling *up* in terms of computer hardware, as more systems can handle more simultaneous requests than one large (abit, powerful) system.

Using tables in this way frees up developers from worrying about controlling a Database Management System (*DBMS*), allowing them to focus on their application rather than the mechanisms by which large amounts of data are stored and transferred. Reading a single entity from a table returns all of its properties, and writing to an entity is an atomic operation which updates all of the currently held properties of that entity. However, this does not answer the question of what happens during simultaneous write operations by more than one client on stored tables. In this case, Azure will follow the same optimistic concurrency model used for blob sub-block updates.

The final approach to data storage in Azure is the use of *Queues*. Primarily intended as a form of *Inter-Process Communication (IPC)*, Queues in Azure systems are used as a way for Web and Worker roles to pass requests and data. For example, should a client facing Web role receive a request for a computationally intensive task, it can use a Queue to publish a request for that task to be completed. A worker role can read that request, do the work and return a response using the same mechanism. When a Worker role reads a request for task, it does not delete that request but instead renders it invisible to other Worker roles for a set period of time (usually 30 seconds). Using this procedure ensures that should the Worker role crash or be unable to complete its actions in some other way, then the request will reappear after the set time-out and be available to be read by another Worker role. This is a perfect example of at-least-once requesting of operations, to ensure that tasks are carried out.

2.3 Miscellaneous

When creating a storage account the Azure system gives a developer a secret key, and requires that each request an application makes to this account uses a signature created with this key. This means that for the Azure platform authorisation takes place at the account level, not through the use of access control lists or any other more common mechanisms of authentication. Developers do however have the ability to implement any authorisation scheme they want for their applications users. The account level authentication applies only to communication between the application and the storage account.

All data stored using the Azure storage mechanisms are replicated three times to aid fault tolerance. This means that the loss of one set of data would not be fatal to the applications, thanks to redundant back-ups. Every piece of data in the Azure storage system uses Representational State Transfer (*REST*) standards to identify and expose data, meaning that every blob, table and queue

have known URIs and can be accessed over HTTP. Azure allows developers to use any HTTP-based authentication mechanism they like for their clients, meaning that they can implement their own custom in-house service or use Microsoft's Live ID program. Once the clients have authenticated with the application it will communicate with the Azure OS and use its secret key to access the required resources.

Developing applications for Windows Azure is much like developing programs for local machine. They can be programmed in a similar way to standard Windows applications, therefore presenting a familiar environment for Software Engineers. Microsoft provides templates for Azure applications as part of the Azure SDK Visual Studio 2008 in order to guide new developers to the system. The SDK also includes the development fabric, a locally run version of the Windows Azure environment to emulate the functionality of the cloud on the local machine. Complete with Web role, Worker roles and all of the storage mechanisms, the development fabric is a way for users to completely test and develop their software locally before uploading to the cloud. In this way, it's entirely possible to create a Windows Azure application without actually using the Microsoft datacenters. Of course, doing this would limit access to local users only.

Once a developer does decide to upload their application and run it on the cloud, they need to log in to the Azure portal and set up their program. Azure provides tools such as logging, resource monitoring and customisable alerts (mailed to the developer), to aid the process of testing, debugging and general software quality assurance. When a developer decides that the application is ready and wishes to make it "live" then the Azure portal will run the application and automatically change the DNS server entry to reflect the new live version. This means that application updating can be done with no noticeable downtime for users, and that the IP address of the physical machines upon which the application is hosted can be masked from end users. Azure also attempts to combat one of the issues effecting cloud computing; geopolitics. If, for example, a developer requires that an application's data must remain in the European Union (EU) for legal reasons, or that the primary customer base is in North America, then the developer can indicate when configuring their application which of the Microsoft datacenters, listed by rough location, should host the application and its data.

3. Google App Engine

Google's equivalent platform as a service infrastructure solution is called *App Engine*. Google App Engine (*GAE*) allows developers to create applications which run on Google's existing infrastructure. It supports applications written in several languages, most notably Java and Python. Others are supported through the use of interpreters (for example, Javascript), which allow these languages to run using the compilers of supported languages [4].

GAE follows the same billing model as Utility computing, where you pay for what you use with no initial expenditure costs. The different resources of storage, bandwidth and CPU time can all be allocated daily budgets to limit the cost of running the application so there are no unexpected bills. These hard limits cannot be breached without authorisation. Google also provides a generous amount of free resources, up to 500MB of storage and enough CPU and network bandwidth to serve

up to 5 million page requests a month for an application which is efficient in its processing and traffic. Paying quotas are on top of this free allowance, you only pay for what you use above the free amount. The platform is designed to be reliable and available to host applications which deal with large amounts of data and many simultaneous users. To cope with such requirements, App Engine supports:

- Dynamic web page serving, with full support for common web technologies
- Persistent storage, with queries, sorting and transactions
- Automatic scaling and load balancing for applications
- APIs for many features including user authentication and sending mail using Google Accounts
- Scheduled tasks for triggering background processes

3.1 Programs

In order to protect the physical hardware and operating systems of the machines GAE uses, a sandbox environment is used to execute applications. Using a secure, hardware independent environment ensures that the App Engine is able to distribute requests for instances of the application over multiple physical machines, scaling up and down to meet demand.

This imposes a number of limitations on the applications that run in the sandbox environment. These include no direct file system access and limited communication with the outside world. However, GAE does support a number of features to combat these limitations, including the use of the Datastore and memcache mechanisms to hold data which the application needs to persist between requests, and the URL fetch system to access external web resources. URL Fetch allows applications to access resources on the internet such as web services or other data, using the high-speed infrastructure that Google itself uses for its own web products.

Perhaps the biggest limitation that GAE imposes is the hard set time-out for processing of 30 seconds which applications must adhere to and return responses within. This limit is designed to ensure that response to the user is swift and that no application can hold all of the CPU time.

The Java runtime environment provided in App Engine allows the use of the Java servlet standard to communicate between applications and the sand box environment. It supports the use of common web application technologies such as Java Server Pages (*JSP*). An application can use any of the Java common features as long as that feature does not interfere with the sandbox limitations. Most GAE services, such as Datastore, are accessed using standard Java APIs, for example the Java Data Object (*JDO*) standard. Python is also supported in a similar manner to Java, with its own runtime environment, APIs and tools for managing and accessing data [3].

3.2 Datastore

The distributed data storage service that GAE provides, Datastore, includes a powerful query engine and support for transactions. A Datastore instance is not defined by a schema. The structure of entities is provided and enforced by applications code.

In a similar manner to Microsoft Azure, the App Engine Datastore also utilises optimistic concurrency control, but has a re-try functionality built-in. This means that if two processes are attempting to update the same entity simultaneously, the first to finish will succeed and the second will fail. However, it will be able to immediately re-try up to a fixed number of times.

In addition to this, Datastore updates can be conducted as transactions with atomicity so that they are either committed in full or not at all. This can be done for multiple operations at once, so that either all will succeed or none will. Transactions are implemented across a distributed network using “entity groups”. Entities which are decreed to be in the same group by the developer are stored together for efficient processing and execution of transactions. Groups can be assigned in objects by the developer when annotating for persistent storage. To perform queries, App Engine uses *GQL*, a query language with SQL like syntax. GQL is able to retrieve entire entities from the Datastore and includes the ability to filter on properties, specifically sorting order and limiting the number of results returned, much like one would expect from a traditional relational database query language.

Memcache is the name given to the high performance memory service included in GAE. It is useful for data that does not need the features of the Datastore, such as persistence and transactions, so may be used for temporary memory that the application requires, for example to store the return of a Datastore query, thereby saving performance costs and making the application more efficient.

3.3 Miscallenous

User authentication is achieved in App Engine by using Google Accounts, therefore providing the fine grained security that may be expected with the added bonus of zero initialisation time should the user already posses a Google account. APIs are provided to give the application information about the current user, allowing specific users to be made administrators or given separate privilege groups, therefore giving easy support for group specific parts of the application.

Google provides App Engine Software Development Kits (SDKs) for the languages Java and Python which emulate all of the facilities of the App Engine on a local machine. This software can be used to fully test applications before they are uploaded to the cloud and include a web server, secure sandbox environment for the application and all of the APIs and libraries used by GAE. This SDK is available as a plugin for Eclipse, the most common Integrated Development Environment (*IDE*) for Java, further simplifying the migration process for developers. When developing a new version of your application, the App Engine also allows developers to upload the new version while the current version is running, then specify which version of the application is *live* through the web interface, avoiding any down time during the upgrade process. There is actually no need to use the GAE Web UI during App Engine development after the initial setup as future versions of the program can be uploaded and set to live by simply clicking the “deploy” button provided by the GAE Eclipse Plugin. This will package, upload and run the program automatically with no visible downtime.

4. Comparison

4.1 Methodology

To give an idea of how the two platforms compared, for the purposes of this paper the author wrote a similar application for each platform. The application chosen was a type of online Guestbook, which would allow users to leave a message. This was chosen for a few reasons, the first being that both Microsoft and Google provided tutorials for creating this type of application and the second because the type of application uses a number of features that the platforms while remaining relatively simple. The application presents a web page so needs output mechanisms, and also requires client independent state, or persistent storage. As these features on cloud platforms are more complicated than on traditional applications, this would be a perfect way to experiment with the cloud mechanisms.

4.2 Getting Started

4.2.1 Google App Engine

The author began by creating a Google App Engine application as the author already had experience with the technologies and languages used to create App Engine programs. As GAE supports common Java technologies it is easy for an experienced Java programmer to begin coding programs to be used on this platform. Google makes it easy also to get started straight away, with a plugin for Eclipse Integrated Development Environment (IDE) which provides all of the App Engine tools required to begin developing programs. The plugin adds a few select new features to Eclipse such as the ability to create a "New Web Application Project" which defines the structure required by programs running on App Engine and the ability to locally test programs written for App Engine through a testing environments that emulates all of the properties and restrictions of Google's infrastructure.

4.2.2 Microsoft Azure

Perhaps the toughest part of creating an application for the Microsoft Azure platform is configuring our local machine so that you can begin developing software for the cloud. While App Engine requires the installation of the Eclipse IDE and then a plugin, Azure requires that you install the Visual Studio 2008 IDE, patch it to Service Pack 1, install .NET Framework 3.5 SP1, then download and install the Azure SDK and Azure Tools for Visual Studio before configuring the ASP.NET and WCF HTTP properties in your own operating system. However, don't forget to install SQL Server 2008 or the local version of the Microsoft data store will not be able to operate. While this is a lot of steps, the result is that you will be able to develop Azure applications on your local machine with an Azure development fabric running in the background to emulate the functionality and restrictions of Azure to allow you to locally test your code.

4.3 API

4.3.1 Google App Engine

The APIs provided by Google are functional and simple to use. As GAE supports existing web standards such as Java Server Pages (JSP) then it is simple to get started writing basic programs. The APIs add to this, by allowing complex interaction with the Google infrastructure in just a few lines of code. For example, adding user authentication to an application using Google Accounts is a simple process of adding a few project import declarations:

```
import appengine.api.users.User;
import appengine.api.users.UserService;
import appengine.api.users.UserServiceFactory;
```

And then extending the program in order to use instances of the User and UserService objects:

```
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
if (user != null) {
    ... // Throw Error
} else {
    ... // Use user details
}
```

4.3.2 Microsoft Azure

Once the Azure development environment is configured, starting a new project is done by selecting a "New Cloud Service" from the new project menu. This will create a project which has two sub projects: one which configures the entire application and one which defines the functionality of the Web Role required by the program. The Web Role functionality is based around the *Default.aspx* page, an HTML page embedded with ASP (*Active Server Pages*) script. The embedded script below defines what is done when interface buttons are clicked and the templates for formatting objects retrieved from the Storage mechanisms.

```
<asp:ObjectDataSource
    ID="ObjectDataSource1"
    runat="server"
    DataObjectTypeName="GB_WebRole.GBEntry"
    InsertMethod="AddGBEntry"
    SelectMethod="Select"
    TypeName="GB_WebRole.GBEntryDataSource">
</asp:ObjectDataSource>
```

This code shows how the method used to input data into the Storage mechanisms is selected and which classes are used to define how the object is created (*GBEntry*).

```
<ItemTemplate>
At
<%# Eval("Timestamp") %>
```

```
a guest called
<%# Eval("GuestName") %>
said
<%# Eval("Message") %>
<br />
</ItemTemplate>
```

4.4 Storage

4.4.1 Google App Engine

Accessing the App Engine data store is also a simple process as it is built upon Java Data Objects. In order to store application objects in the Datastore then you simply need to use Java annotations to specify which fields of the object should be persistent, which should be used as a primary identifier etc. *PersistentManager* class is used to allow the GAE application to communicate with the Datastore, the application needs only to load the required data into a properly annotated object and then ask the PersistentManager instance to store it. Likewise, when viewing a page, getting a list of stored objects from the PersistentManager is a simple method call taking a parameter of a JDOQL string. JDOQL is an SQL like query language for Java Data Objects used by GAE, which allows conditions on the fields of those objects.

GAE also provides functionality to queue data for background processing. This queue is similar in manner and structure to the queuing facility provided by Azure, but the entire queue of data is subject to the GAE standard 30 second limit on processing. In the JDO objects data can be stored as Blobs as attributes of the object. There is however a limit of 1MB on each of the properties in the Datastore table of entities, so this is the maximum size that the data Blob can be. Compare this to the 50GB limit for Azure, which also provides the ability to sub-block Blobs for easier updating and efficient retransmission.

4.2.2 Microsoft Azure

In Azure when accessing the actual data entries an SQL like syntax is used. In a similar manner to JDOQL it allows you to use properties of the object to condition the Query. So for example:

```
public IEnumerable<GuestBookEntry> Select() {
    var results = from g in _context.GBEntry where
        g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
    select g;
    var r = results.ToArray<GuestBookEntry>();
    return r;
}
```

This code is used to select from a data storage table GBEntries which have a particular DateTime string and return the results in a collection to the caller. Azure also provides SQL Azure for relational database tables, so that if the programmer wishes to use a more traditional model of data storage then that option also exists.

4.5 Publishing

4.5.1 Google App Engine

Uploading an application from Eclipse to the Google App Engine is a simple process the user is required to click the button marked "Deploy" and then type their password once prompted. It is an uncomplicated procedure, much like how one would execute a traditional application and changes in version on the cloud take place immediately. The first time you click deploy however, is slightly more complicated; the user will have to visit the Google App Engine website, login with a Google Account (or create one) then proceed to set up the project through the App Engine web interface.

4.5.2 Microsoft Azure

Publishing an application from Visual Studio to the Azure cloud is a more complicated process than using GAE and the Eclipse plugin. It is achieved by first changing the settings in the project configuration file to reflect the new remote environment for the application. Then uploading the Visual Studio build file to Azure through the Azure portal web interface. At this point is important to note down the location of the build file you require as the location is not automatically communicated to the Azure portal and the file has to be selected manually. Once uploaded, the user must then select which version of the application is to be used as "live" and wait for Azure to create a virtual machine to host application before it can be used. At this point the author also had issues with using browsers other than Internet Explorer, so would suggest using the Microsoft browser to avoid any in compatibilities with the Azure web interface.

5. Conclusion

Infrastructure as a service has massive potential and could eventually change the way in which computer systems are viewed. Why develop applications for more than one platform when a cloud application with a web front end can be viewed and interacted with on any internet-enabled device? Why invest capital expenditure in server infrastructure which may never be utilized 100% of the time, when you can use cloud computing with its utility billing to have pay-for-what-you-use costs and always available service? Both of these platforms run on the existing infrastructure that Google and Microsoft use to host their own services, so robustness and availability are built-in properties, which could also be said for long term viability.

A number of criticisms which have been levelled at cloud computing can now also be overcome; both of these platforms are taking steps to comply with Geopolitical issues regarding data location and they provide increased redundancy, scalability, and fault tolerance. The APIs and Storage mechanisms on both platforms are extensive and should cover all foreseeable needs for developers. When one gets used to how these systems operate and how they differentiate from traditional systems, development becomes simply a different experience, not necessarily a challenging one. Both platforms provide excellent tutorials and samples in order to aid developers. For existing Java developers and programming newcomers, App Engine provides an easier learning curve than Azure, especially in setting up development environments. However, for existing C# or ASP.NET developers who are looking to extend applications to the cloud, then Azure is a worthy choice.

GAE also provides an easier deployment process, allowing the user to simply update their program from within the IDE and see results immediately, something which is far more familiar to traditional programmers. Azure would do well to adopt a similar “one-click” approach in future versions, and they do so as Azure is not yet considered to be “feature complete”.

References

- [1] Nicholas Carr. *The Big Switch: Rewiring the World, from Edison to Google*. W.W. Norton & Co., 2009.
- [2] David Chappell. Introducing windows azure. Technical report, Microsoft Corporation, March 2009.
- [3] Google S. Engineers. Getting started: Java - google app engine, August 2009.
- [4] Google S. Engineers. What is google app engine?, August 2009.
- [5] George Reese. *Cloud Application Architectures*. O'Reilly, 2009.
- [6] Jonathan S. Ward. The design and implementation of the St Andrews cloud. Technical report, University of St Andrews, 2009.
- [7] Ronald Yanosky. *From Users to Choosers: The Cloud and the Changing Shape of Enterprise Authority*, chapter 4.3. Educause, 2008.