# AppScale: Open-Source Platform-As-A-Service

UCSB Technical Report #2011-01, January, 2011

Chris Bunch Navraj Chohan Chandra Krintz
Computer Science Department
University of California, Santa Barbara

## 1 Introduction

AppScale is a scalable, distributed, and fault tolerant cloud runtime system that we have developed at the University of California, Santa Barbara as part of our research into the next generation of programming systems [5, 3]. In particular, AppScale is a cloud platform, i.e. a platform-as-a-service (PaaS) cloud fabric, that executes over cluster resources. The cluster resources underlying AppScale can be managed with or without virtualization (e.g. Xen, KVM) or via popular cloud infrastructures including Amazon EC2 [6] and Eucalyptus [8].

The AppScale platform virtualizes, abstracts, and multiplexes cloud and system services across multiple applications, enabling *write-one, run-anywhere (WORA)* program development for the cloud. In addition to simplifying application development and deployment using cloud systems and modern distributed computing technologies, AppScale brings popular public cloud fabrics to "on-premise", or private, clusters. To enable this, we emulate key cloud layers from the commercial sector – (i) to engender a user community, (ii) to gain access to and to study real applications, and (iii) to investigate the potential implementations of, and extensions to, public cloud systems using open-source technologies.

The first API we have chosen to emulate is that of Google App Engine [11]. Google App Engine is a cloud platform that provides scalable implementations of technologies important for web services (messaging, key-value data storage, multi-tasking, web server support, elasticity, and resource management, among others). Using Google App Engine developers debug and test their programs using an open-source software development kit (SDK) provided by Google that implements non-scalable versions of the APIs. Developers then upload their code and data to Google clusters, and can then use Google resources and services "pay-per-use", on a free or low-cost rental basis.

AppScale implements the (open) Google App Engine APIs such that applications that execute on Google App Engine, also execute on AppScale without modification using private cluster resources or public cloud infrastructures. Our API and service implementations target scale, distribution, fault tolerance, high-performance, and high availability. We leverage mature open-source technologies to the greatest degree possible to enable this. AppScale implements multiple language runtimes (Java, Python, Ruby) as application frontends and a wide range of open source database technologies (key-value or relational) as options for its internal system-wide datastore. AppScale is not a replacement for Google App Engine or any other public cloud technology. Instead, AppScale is a robust and extensive distributed system that provisions the resources it is allocated scalably across multiple applications.

The AppScale platform exports services and APIs in addition to those provided by Google App Engine. These technologies are important for application domains beyond those of web services, e.g. data analytics and data and computationally intensive applications. AppScale exports these technologies as services, i.e. AppScale "service-izes" libraries, tools, and software packages, including MapReduce, X10, R, and MPI, and, as it does for the Google App Engine APIs, AppScale provides automatic configuration, deployment, and distribution for them, as well as support for their elasticity, load balancing, and fault tolerance.

Since AppScale provides a software layer between applications and the implementations of cloud APIs and services, we are also able to employ existing cloud services for the implementations. As such, AppScale provides a *hybrid* cloud platform – a programming system through which an application can access services from different clouds (a mix of public and private) at different times or concurrently. Such support can be used by developers to move data between clouds, e.g., for disaster recovery, fault tolerance, data backups, to reduce public cloud costs (to use lower-cost alternatives), and to "fall out" from a private cloud with limited resources to a public cloud on-demand.

## 1.1 The AppScale Internals

Figure 1 shows the layout of AppScale. At the highest level there is a load balancer, which takes incoming requests from users and routes them to an application server, copies of which may be on a number of remote hosts. The application layer is supported by a wide range of services that eases application development by precluding the need for reimplementation of common tasks. At the lowest level there is a system-wide database service which provides persistent storage onto the disk.

AppScale automatically provisions services and does so in a fault tolerant and scalable manner. A key automated process is the setting up of a datastore. AppScale supports many databases that are implemented in many different languages and have different designs. These databases are: `Cassandra` [4], `HBase` [12], `Hypertable` [13], `MongoDB` [15], `MemcacheDB` [14], `Scalaris` [17], `SimpleDB` [18], `MySQL Cluster` [16], and `Voldemort` [21].

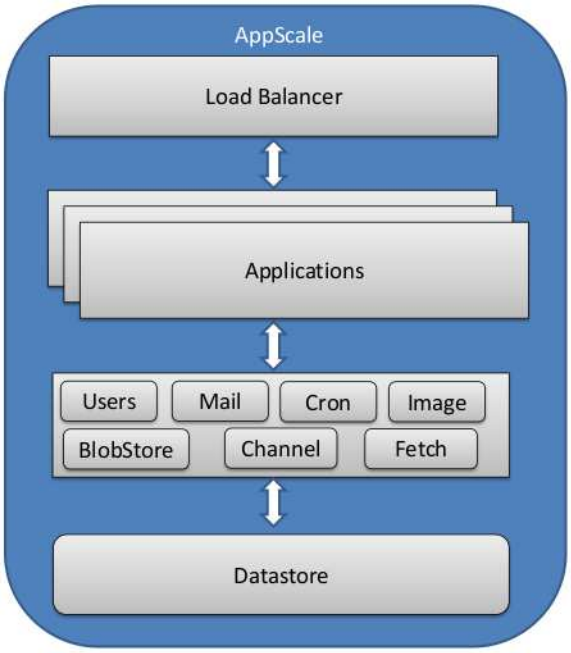


Figure 1: The multi-tiered approach within AppScale consists of a load balancer, multiple application servers, several services to support the different APIs, and a datastore for persistent data and state (not all APIs and services are shown here).

## 1.2 Google App Engine APIs

Foremost, AppScale provides implementations for the Google App Engine APIs. These APIs provide several scalable services which can be leveraged by a Google App Engine application. The Google App Engine APIs are `Blobstore`, `Channel`, `Datastore`, `Images`, `Memcache`, `Namespaces`, `TaskQueue`, `Users`, `URL Fetch`, and `XMPP`. Google App Engine provides an overview of the APIs and their functionality at `http://code.google.com/appengine/docs/`. We emulate this functionality within AppScale using open source software systems, tools, and services, as well as new components that we have written. We implement all of the APIs assuming distributed execution so that we are able to provide isolation, scalability, elasticity, and fault-tolerance for cloud platform applications. This is in sharp contrast to the Google App Engine SDKs that are provided for testing and debugging using a single machine [10]. We describe the features of the API that AppScale currently does not support in Section 4.

**Blobstore API**

The Blobstore API enables users to store large entities of text or binary data. The upload is performed by an

application using a form post with a file name. This large file is not constrained by the 1MB limitation of Blobs stored using the Datastore API.

**Channel API**

The Channel API allows applications to push messages from the application server to a client's browser. The program must register an application token with the service, which in turn is used by a JavaScript library to connect to a messaging service. The Channel API is useful for creating applications such as online chat or a real time multi-player game. The implementation in AppScale has the ability to send messages to multiple receivers who are registered using the same application token. Google App Engine imposes the restriction that there may be only one receiver per sending channel. AppScale uses Ejabberd and Strophejs in its implementation.

**Datastore API**

The Datastore API allows for the persistence of data. The API has many useful function calls that all ultimately map to either a PUT, GET, DELETE, QUERY. The Google Query Language (GQL) is similar to and is a subset of SQL; fundamentally, it lacks relational operations such as JOIN and MERGE. AppScale employs in-memory filters for GQL statements while Google App Engine maps them to range queries on the datastore.

AppScale implements transactional semantics (multi-row/key atomic updates) using the same semantics as Google App Engine. Transactions can only be performed within an entity group [9]. Entity groups are programmatic structures that describe relationships between datastore elements. Similarly, transactions are expressed within a program via application functions passed to the Google App Engine "run_as_transaction" function. All AppScale datastore operations within a transaction are ACID-compliant with READ-COMMIT isolation.

AppScale's implementation of transactions are database agnostic and rely on ZooKeeper [23] for locking entity groups. ZooKeeper stores information about which entities are currently locked and our transactional middleware employs a garbage collector for locks to make sure that locks are not held for a long periods of time. We store journal information within the database and implement rollback of failed transactions.

Developers use entity groups and transactions for applications that require atomic updates across multiple keys (in an entity group). Yet, transaction semantics introduce overhead, stress the underlying database implementation, and limit scalability (relative to non-transactional operations). To reduce this overhead, we encourage program developers to avoid creating large entity groups and to shard (and spread out) global state across many entity groups.

**Images API**

The Images API facilitates programmatic manipulation of images. Popular functions in this API include the ability to generate thumbnails, rotation, composition, convert formats, and cropping images. The API in AppScale uses the Google App Engine SDK implementation.

**Memcache API**

The Memcache API permits applications to store their frequently used data in a distributed memory grid. This can serve as a cache to prevent relatively expensive re-computations or database accesses. Developers must be aware that it is possible that entries may be evacuated to create space for new updates.

**Namespace API**

The Namespace API implements the ability to segregate data into different namespaces. For example, developers can test their application in production without tampering with live production data. The Namespace API can also be used with the Memcache and Task Queue APIs.

**Task Queue API**

The Google App Engine platform lacks the ability to do threading or computations within a request greater than 30 seconds. The TaskQueue API facilitates computation (tasks) in the background by enqueuing tasks via a web request. Task durations are restricted in Google App Engine to 30 seconds (as of GAE 1.4.0). AppScale implements the same restriction but this value can be increased with minor code modification. A potential workaround is to chain multiple task queues, where each task does up to 30 seconds of work and then enqueues another task to pick up where the current one left off.

**Users API**

The Users API provides authentication for web applications through the use of cookies. While GAE provides access to users who have accounts with Google, AppScale requires users to sign up an account through the AppScale portal URL. In addition, the API distinguishes between regular and administrator users.

**URL Fetch API**

The URL Fetch API enables an application the ability to do POST and GET requests on remote resources. REST APIs from third parties can be accessed using this API. The implementation in AppScale is identical to what is provided by the Google App Engine SDK.

**XMPP API**

The XMPP API presents the ability to receive and send messages to users with a valid XMPP account. Google App Engine leverages the Google Talk infrastructure while AppScale is able to use a scalable implementation in Ejabberd [7].

## 1.3 Other AppScale APIs

In addition to the Google App Engine APIs, AppScale implements other APIs which are not supported by Google App Engine. These APIs are of use to cloud platform applications developers for emerging application domains such as data analytics and high-performance computing.

### MapReduce Streaming API

. Within Google App Engine itself, there currently is no method by which long running, arbitrary computation can be performed by applications. AppScale supports such computation via Hadoop Streaming (`http://wiki.apache.org/hadoop/HadoopStreaming`). Through Hadoop Streaming, users can specify a Mapper and Reducer program that can run under the MapReduce programming paradigm. Fault-tolerance and data replication is automatically handled by the Hadoop Streaming framework, and AppScale exposes a simple API that interfaces to Hadoop Streaming. This API is:

- *putMRInput(data, inputLoc)*: Given a string ”data” and a Hadoop file system location “inputLoc”, creates a file on the Hadoop file system named “inputLoc”
- *runMRJob(mapper, reducer, inputLoc, outputLoc, config)*: Given the relative paths to a mapper and reducer file (relative to the main Python file being run), run a Hadoop MapReduce Streaming job. Input is fed to the program via the HDFS file named “inputLoc”, and output is fed to the HDFS file named “outputLoc”. If a hash is passed as ”config”, the key / value pairs are passed as configuration options to Hadoop Streaming.
- *getMROutput(outputLoc)*: Given a Hadoop file system location “outputLoc”, returns a string with the contents of the named file.
- *writeTempFile(suffix, data)*: Writes a file to /tmp on the local machine with the contents data. Is useful for passing a file with nodes to exclude from MapReduce jobs.
- *getAllIPs()*: Returns an array of all the IPs in the AppScale cloud. Is useful for excluding or including nodes based on some user-defined application logic.
- *getNumOfNodes()*: Returns an integer with the number of nodes in the AppScale cloud. Is useful for determining at MapReduce run time, how many Map tasks and Reduce tasks should be run for optimal performance (recommended value is 1 Map task per node).
- *getMRLogs(outputLoc)*: Returns a string with the MapReduce job log for the job whose output is located at outputLoc. Data is returned as a combination of XML and key / value pairs, in the standard Hadoop format.

Currently, Mappers and Reducers can be written by developers in Python, Ruby, or Perl. In addition, we and others can easily extend AppScale with support for other programming languages.

To use this API, the cloud administrator must select HBase or Hypertable as the platform datastore so that the Hadoop Distributed File System (HDFS) is automatically deployed (both of these datastores use HDFS for their implementations). A complete sample application that uses the MapReduce Streaming API is bundled with the AppScale Tools and is called *mapreduce*.

### EC2 API

Google App Engine does not provide any mechanisms by which users can interact with Amazon Web Services natively, so to fill this void, AppScale provides an Amazon EC2 API. Users can use this API to spawn virtual machines and manage them entirely through an AppScale web service. The main functions this API provides mirror those of the EC2 command line tools:

- *ec2_run_instances(options)*: Spawns virtual machines with the specified options (interpreted as command-line arguments).
- *ec2_describe_instances()*: Returns the status of all machines owned by the current user.
- *ec2_terminate_instances(options)*: Terminates virtual machines with the specified options (interpreted as command-line arguments).
- *ec2_add_keypair(options)*: Creates a new SSH key-pair that can be used to log in to virtual machines that are spawned with this key-pair's name.
- *ec2_delete_keypair(options)*: Deletes the named SSH key-pair from the underlying cloud infrastructure.
- *ec2_describe_availability_zones(options)*: In Eucalyptus, this returns information relating to the number of virtual machines available to be spawned by users.
- *ec2_describe_images()*: Returns information about the virtual machine images, ramdisks, and kernels that are available for use in the system.
- *ec2_reboot_instances(options)*: Reboots virtual machines with the specified options (interpreted as command-line arguments).

Other functions are available to handle the storage and retrieval of EC2 credentials: their usage can be found in the *ec2demo* application that is bundled with the AppScale Tools. Alternatively, the KOALA Cloud Manager project provides a Google App Engine application that implements several AWS APIs (as of writing, EC2, EBS, S3, and ELB are supported), and can run over Google App Engine as well as over AppScale. It can be found at: `http://code.google.com/p/koalacloud/`.

## 2 Installation and Deployment

The typical use of AppScale and the one we recommend is to install AppScale within a virtual machine image. The use of virtual machines for AppScale installation facilitates elasticity (growing and shrinking of services and applications, and load balancing). We currently support the use of Xen and KVM as the virtualization infrastructure, however, any technology should suffice. Deploying AppScale over a virtualized private cluster entails copying the virtual machine image and instantiating it manually. Thus, a cloud administrator is able to use any virtualization system with which she is able to instantiate guest virtual machines (VMs). We recommend the use of vmbuilder [20] for creating Xen guest VMs and virt-install [19] for creating KVM guest VMs. We currently support only the Ubuntu 10.04 (Lucid Lynx) Linux server system and distribution and so guest VMs should implement these technologies. Once a guest VM image is running Ubuntu 10.04 Server Edition ensure that the instance is able to obtain an accessible IP address (e.g., the networking operates correctly).

An alternative AppScale deployment is over a cloud infrastructure. We currently support both the Amazon Web Services (AWS) Elastic Compute Cloud (EC2) public cloud offering and the open source implementation of AWS,

Eucalyptus, which runs on virtualized private clusters. We have made an Amazon AMI image available to users that is installed with AppScale. The cloud administrator must obtains account credentials and should test that a single VM can be deployed without any issues. The instance should be able to access an IP address. Troubleshoot these issues with the cloud provider if there is failure either start a VM or obtain an IP address. Terminate the instance upon success and proceed to the next section.

## 2.1 Deploying AppScale

As described above, AppScale can be deployed by users over a virtualized private cluster or over a cloud infrastructure. AppScale automates the deployment process via a set of tools called the AppScale tools that implement a cloud platform administration interface [1].

The tools configure the platform components and start each in the correct order. In addition, the tools synchronize SSH keys and credentials across the AppScale VM instances. These tools can be run by the cloud administrator on any machine with network access to the instances that make up the platform. Note that the machine on which the tools are executed must be able to resolve the IP addresses of the AppScale virtual machine instances. Likewise, the VM instances that make up the platform must be able to resolve the IP addresses of and communicate with the other instances in the AppScale platform.

The size of the AppScale cloud is specified by the cloud platform administrator. The size can be one or more nodes; a node is an instance of the AppScale VM.

There is one key difference between the deployment of AppScale over a virtualized private cluster versus a cloud infrastructure. For private virtualized cluster deployment, the user (cloud administrator) must manually start (instantiate) the nodes. To do this, the user must make X copies of the AppScale image for a deployment of X nodes. These nodes (AppScale VMs) can be instantiated on any number of physical machines, depending upon the resources available. We recommend at least 1GB of memory and 2 virtual CPUs per AppScale instance. The administrator should log into each of the AppScale instances once started and collect each of the instance IP addresses.

AppScale automates VM instantiation when it is deployed over a cloud infrastructure, e.g. via AWS or Eucalyptus. That is, AppScale instantiates the VM instances automatically for the administrator as part of the cloud deployment process. In either case, the cloud administrator specifies the size (and optionally the roles) of the cloud platform using a configuration file located on the same machine from which the AppScale tools will be run.

The configuration file is written in YAML [22], a popular, easy-to-use, human-readable data serialization standard format. This file specifies the number of nodes in the cloud platform and their roles. For deployment over a private virtualized cluster resources, the administrator specifies the IP addresses of the AppScale VM instances that were started:

```
---
:controller: 192.168.1.2
:servers:
- 192.168.1.3
- 192.168.1.4
- 192.168.1.5
```

Note again that the IP addresses specified must be resolvable from the machine from which the AppScale tools are run.

For deployment over a cloud infrastructure, the administrator specifies a unique identifier of the form `node-X` for increasing integers starting with 1, for each of the VMs that will be part of the AppScale cloud:

```
---
:controller: node-1
:servers:
- node-2
- node-3
- node-4
```

The cloud configuration file identifies the first machine (192.168.1.2) as the master node (controller) in the system: on this node, AppScale will automatically deploy a Database Master service and load balancer. AppScale will deploy Database Slave services and Application servers on the other nodes (servers) specified in the configuration.

Application servers host the App Engine applications. AppScale can be run using a single VM instance by specifying only the controller (see `http://code.google.com/p/appscale/wiki/Single_Node_AppScale_Deployment`: `:controller: 192.168.1.2` or `:controller: node-1`, following the `---` first line. In this case, all of the components are invoked by the AppScale deployment process within the single VM specified.

There is one additional step that must be performed next for users that deploy AppScale over a virtualized private cluster. Note that for cloud infrastructure deployment, the tools perform this step automatically so this step is not necessary. This step sets up an SSH keypair for use across the AppScale VMs instances that the user has started. The AppScale tool 'appscale-add-keypair –ips ips.yaml" performs this function. `ips.yaml` is the name of the platform configuration file which contains lines similar to those specified above (with IP addresses to denote the use of private cluster). This tool prompts the user for the root password on each VMs listed in the configuration file. The output resembles the following:

```
user@mylaptop:~/appscale-tools/ips$ appscale-add-keypair --ips ips.yaml
root@192.168.1.3's password:
Now try logging into the machine, with "ssh 'root@192.168.1.3'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

The authenticity of host '192.168.1.4 (192.168.1.4)' can't be established.
RSA key fingerprint is e8:f5:6f:83:3a:06:84:38:b8:94:6d:59:d0:7d:fc:d0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.4' (RSA) to the list of known hosts.
root@192.168.1.4's password:
Now try logging into the machine, with "ssh 'root@192.168.1.4'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

The authenticity of host '192.168.1.5 (192.168.1.5)' can't be established.
RSA key fingerprint is e8:f5:6f:83:3a:06:84:38:b8:94:6d:59:d0:7d:fc:d0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.5' (RSA) to the list of known hosts.
root@192.168.1.5's password:
Now try logging into the machine, with "ssh 'root@192.168.1.5'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

root@192.168.1.2's password:
Now try logging into the machine, with "ssh 'root@192.168.1.2'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

A new ssh key has been generated for you and placed at
/home/user/.appscale/appscale.
You can now use this key to log into any of the machines you specified without
providing a password via the following command:
        ssh root@XXX.XXX.XXX.XXX -i /home/user/.appscale/appscale
```

From this point forward, the steps are the same for both a virtualized private cluster deployment and a cloud infrastructure deployment of AppScale. The next step is to initiate the automatic configuration and creation of the different AppScale components that make up the platform. This step is achieved using the AppScale tool "appscale-run-instances". The cloud administrator specifies that database to use within the platform and (optionally) the first application that the administrator wishes to deploy over the cloud platform. The administrator can use the tool "appscale-upload-app" later to upload one or more applications to a running AppScale cloud platform. The output from this tool is similar to the following. In this example, we are deploying AppScale over a virtualized private cluster; the ips.yaml file specifies the roles and IP addresses in this case as described above. To deploy over a cloud infrastructure, the user adds the flag `--infrastructure euca` for Eucalyptus or `--infrastructure ec2` for Amazon EC2, and replaces the IP addresses with node identifiers in the ips.yaml file, as described above.

```
user@mylaptop:~/appscale-tools/ips$ appscale-run-instances --ips ips.yaml
--file ../sample_apps/guestbook/ --table cassandra

About to start AppScale over a non-cloud environment.
Head node successfully created at 192.168.1.2. It is now starting up cassandra
via the command line arguments given.
Generating certificate and private key
Starting server at 192.168.1.2
Please wait for the controller to finish pre-processing tasks.

This AppScale instance is linked to an e-mail address giving it administrator
privileges.
Enter your desired administrator e-mail address: user@example.com

The new administrator password must be at least six characters long and
can include non-alphanumeric characters.
Enter your new password:
Enter again to verify:
Please wait for AppScale to prepare your machines for use.
Copying over needed files and starting the AppController on the other VMs
Starting up Cassandra on the head node

Your user account has been created successfully.
Uploading guestbook...
We have reserved the name guestbook for your application.
guestbook was uploaded successfully.
Please wait for your app to start up.

Your app can be reached at the following URL:
http://192.168.1.2/apps/guestbook
The status of your AppScale instance is at the following URL:
http://192.168.1.2/status
```

AppScale begins by initializing the first node (master node) in the system. It then asynchronously starts up the other nodes in the system. While this is happening, the administrator enters an e-mail address and password for platform administrator access. The platform administrator is a special user that is able to administrator and control all applications that execute in this AppScale deployment as well as the cloud platform itself. Next in the process, the system starts Cassandra, the datastore selected by the cloud administrator as the platform storage system. The system then creates the administrator account, uploads the guestbook application, and return a URL to the user that implements the AppScale cloud status interface. AppScale implements load-balancer in front of this URL to direct traffic across the available server instances in the platform. The status page lists the services running on each node in the platform, the CPU and memory used by each, and the applications that are currently running.

In a cloud infrastructure deployment, the output is similar. However, the process begins with the spawning (instantiating) of virtual machine instances using the infrastructure. Depending on the infrastructure, VM spawning can take

a significant amount of time (minutes to hours) depending on the number of nodes and the size of the VM images.

The administrator can use the tool “appscale-describe-instances” to see the status of a cloud platform deployment. The output of this tool is similar to that which AppScale displays on its platform status page:

```
user@mylaptop:~/appscale-tools/ips$ appscale-describe-instances
Status of node at 192.168.1.2:
    Currently using 1.0 Percent CPU and 96.94 Percent Memory
    Hard disk is 99 Percent full
    Is currently: load_balancer, shadow, db_master, zookeeper, login
    Database is at 192.168.1.2
    Current State: Preparing to run AppEngine apps if needed
Status of node at 192.168.1.3:
    Currently using 0.3 Percent CPU and 95.00 Percent Memory
    Hard disk is 98 Percent full
    Is currently: load_balancer, db_slave, appengine
    Database is at 192.168.1.3
    Current State: Preparing to run AppEngine apps if needed
    Hosting the following apps: guestbook
Status of node at 192.168.1.4:
    Currently using 0.3 Percent CPU and 96.45 Percent Memory
    Hard disk is 98 Percent full
    Is currently: load_balancer, db_slave, appengine
    Database is at 192.168.1.4
    Current State: Preparing to run AppEngine apps if needed
    Hosting the following apps: guestbook
Status of node at 192.168.1.5:
    Currently using 0.3 Percent CPU and 95.65 Percent Memory
    Hard disk is 98 Percent full
    Is currently: load_balancer, db_slave, appengine
    Database is at 192.168.1.5
    Current State: Preparing to run AppEngine apps if needed
    Hosting the following apps: guestbook
```

The tools“appscale-upload-app” and “appscale-remove-app” are used by cloud platform users to add and remove applications, respectively, from a running AppScale deployment. In this example, we first add an application called “shell”:

```
user@mylaptop:~/appscale-tools/ips$ appscale-upload-app --file \
../sample_apps/shell/

This AppScale instance is linked to an e-mail address giving it administrator
privileges.
Enter your desired administrator e-mail address: user@example.com
Preparing to run AppEngine apps if needed

Uploading shell...
We have reserved the name shell for your application.
shell was uploaded successfully.
Please wait for your app to start up.

Your app can be reached at the following URL:
http://192.168.1.2/apps/shell
```

As part of this process, we specify the path to the directory containing our application as well as the e-mail address of the administrator for this application (note this might not necessarily be the cloud administrator). We are not

prompted for a password in this example because the user we specify already exists. In this example, we remove the "shell" application:

```
user@mylaptop:~/appscale-tools/ips$ appscale-remove-app \
--appname shell

We are about to attempt to remove your application, shell.
Are you sure you want to remove this application (Y/N)? Y
Your application was successfully removed.
```

The output here is fairly straightforward: we provide "appscale-remove-app" with the name of the application to remove, and after confirming our selection, it contacts the master node in the system and stops the "shell" application. Note that if the application actually isn't running in the system (e.g., because we misspelled the name of the application to remove or we already stopped it earlier), then the output informs us that this is the case:

```
user@mylaptop:~/appscale-tools/ips$ appscale-remove-app --appname shell2
We are about to attempt to remove your application, shell2.
Are you sure you want to remove this application (Y/N)? Y
We could not stop your application because it was not running.
```

Cloud administrators can terminate an AppScale deployment using the tool "appscale-terminate-instances". For virtualized private clusters, this step shuts down all the AppScale services across the running VMs without shutting down the VM instances. cloud deployments, this step does both. For example, the shutdown of an AppScale cloud that we have deployed over virtualized private cluster, looks similar to the following:

```
user@mylaptop:~/appscale-tools/ips$ appscale-terminate-instances --ips ips.yaml
Not backing up data.

About to terminate instances spawned via Xen/KVM with keyname 'appscale'...
Shutting down AppScale components at 192.168.1.2.................
Shutting down AppScale components at 192.168.1.3.....
Shutting down AppScale components at 192.168.1.4
Shutting down AppScale components at 192.168.1.5
Terminated AppScale across 4 boxes.
```

Terminating in cloud deployments is similar. However, we specify the cloud infrastructure type over which we are deploying.

```
user@mylaptop:~/appscale-tools/ips$ appscale-terminate-instances \
--infrastructure euca
Not backing up data.

About to terminate instances spawned via euca with keyname 'appscale'...
Terminated AppScale in cloud deployment.
```

## 2.2 Advanced Deployment Strategies

AppScale is also capable of performing more advanced placement of components and services. This is controlled through the YAML cloud platform configuration file that we described in the previous subsection. In that previous example, the "ips.yaml" file identifies one node as the "controller". This role consists of a load balancer and Database Master service. The roles of the other nodes, called "servers", consist of Database Slaves and application servers. This is the default AppScale placement strategy and we illustrate it in Figure 2.

Users can define their own placement strategies instead of using the default. The possible roles are:

- Load balancer: The service that routes users to their Google App Engine applications. It also hosts a status page that reports on the state of all machines in the currently running AppScale deployment.

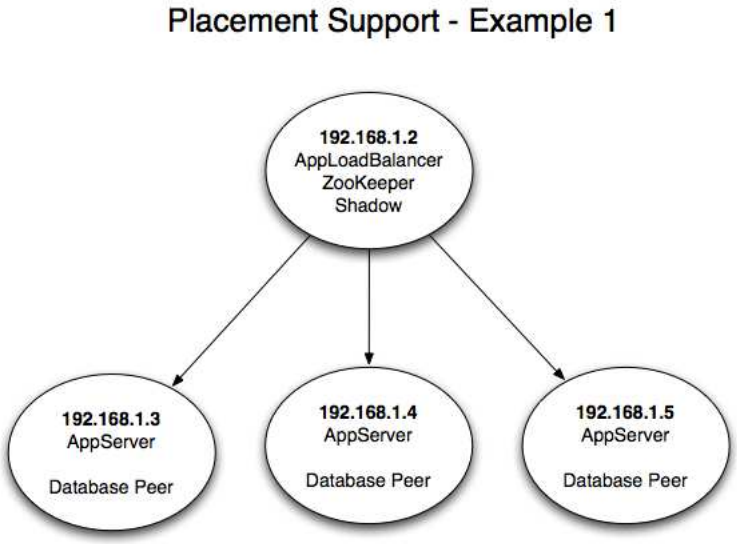


Figure 2: The default placement strategy within AppScale. Here, one node (the controller) handles load balancing and a single database server, while the other nodes (the servers) deploy application servers as well as database servers.

- App Engine: Our modified version of the non-scalable Google App Engine SDKs that adds in the ability to store data to and retrieve data from databases that support the Google Datastore API, as well as our implementations of the various other APIs that App Engine offers.
- Database: Runs all the services needed to host the chosen database.
- Login: The primary machine that is used to route users to their Google App Engine applications. Note that this differs from the load balancer - many machines can run load balancers and route users to their applications, but only one machine is reported to the cloud administrator when running "appscale-run-instances" and "appscale-upload-app".
- ZooKeeper: Hosts metadata needed to implement database-agnostic transaction support.
- Shadow: Queries the other nodes in the system to record their state and ensure that they are still running all the services they are supposed to be running.
- Open: The machine runs nothing by default, but the Shadow machine can dynamically take over this node to use it as needed.

We show how to implement a placement strategy via example. In this example, we wish to have a single machine for load balancing, for execution of the database metadata server via ZooKeeper, and for routing users to their applications. These functions are commonly used together, so the role "master" can be used to signify all of them in a single line of configuration. On two other nodes, we wish to host applications, while on the final node, we wish to host our database. The configuration file for this deployment strategy using a virtualized private cluster (because we specify IP addresses) looks like the following:

```
:master: 192.168.1.2
:appengine:
- 192.168.1.3
- 192.168.1.4
:database:
- 192.168.1.5
```

In another example deployment, we use a cloud infrastructure with one node hosting the "master" role, two nodes hosting web applications and database instances, and a final node routing users to their applications. The configuration file for this looks like the following:

```
---
:master: node-1
:appengine:
- node-2
- node-3
:database:
- node-2
- node-3
:login:
- node-4
```

Figure 3 illustrates this placement strategy, where the machines have been assigned IP addresses beginning with 192.168.1.2.

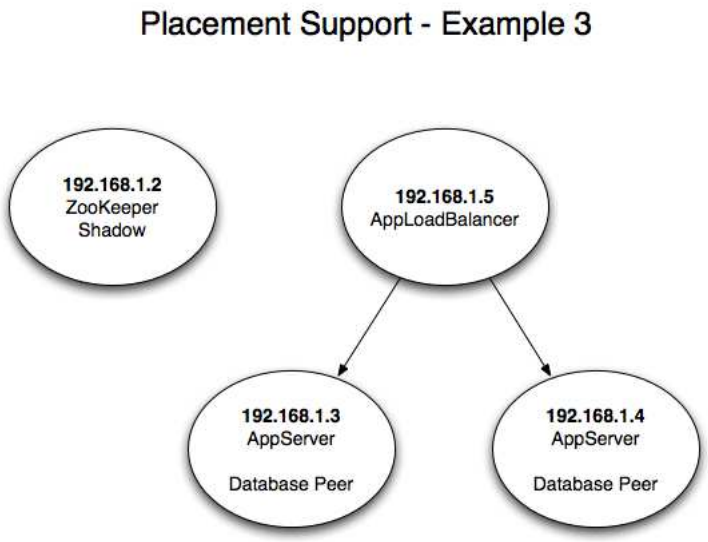


Figure 3: A more complicated placement strategy, in which a single node handles load balancing, two nodes handle application servers and database nodes, and a final node handles transaction management and system monitoring.

For databases that employ a master / slave relationship, the first machine specified in the “database” section is the master, while the others are slaves. For peer-to-peer databases, all nodes are peers, but the machine specified first in the “database” section is the first peer to be started.

## 2.3 Common Errors

While AppScale aims to simplify application deployment, there are many layers of abstraction on which it relies to do so. If any one of these layers fail, AppScale can fail as well. To make troubleshooting easier, the “appscale-run-instances” tool can be with the “-v” flag to produce verbose output, which oftentimes is verbose enough to capture information about errors in the system. When this flag is used, a log file is placed on the “master” node (or “controller” node in default deployments) in the /tmp directory, named ¡ip of master¿.log. When reporting errors or troubleshooting problems with AppScale, this file is often the key to a speedy resolution of problems. Similarly, the master node places log files on each other machine at the same location in case those machines encounter errors.

It is important to note that not all error messages within the verbose log are fatal: SSL-related exception messages within this log can be safely ignored, as well as nginx related messages. Other exceptions, however, may be indicative of actual problems occurring with an AppScale deployment.

One common way that AppScale can fail is if, in the virtualized deployment strategy, AppScale is installed on one virtual machine but not the others. As the components needed are not on all machines, AppScale does not have the necessary components to run and will fail. Users can easily verify if this is the problem they are running into by

checking the logs AppScale produces - it will contain an error message saying this is the problem and how to remedy it. Alternatively, users can check to see if each virtual machine has AppScale installed on it.

A similar type of problem can occur if users install support for a subset of supported databases in AppScale and then try to run a non-installed database. For example, if users only install Cassandra support for AppScale and then try to run HBase, AppScale will fail to start, as it does not have the necessary files to start HBase. Once again, the verbose logs contain an error message that indicate that this is the problem and how to remedy it.

Typically, more difficult problems tend to arise when non-standard networking setups are employed. If an "/etc/hosts" file is used that is not the default provided with the AppScale image, it is extremely likely to break components within AppScale. This is because various components in AppScale rely on network lookup operations occurring in a certain fashion, and if these assumptions are violated, the underlying software may not work as effectively or at all. Diagnosing these problems tends to be much more difficult: the verbose logs should contain error messages indicating that network-related problems are occurring, and if these are seen, reverting to the standard "/etc/hosts" file may be beneficial.

Another difficult class of problems arise when other types of software that AppScale relies on are not installed or configured correctly. For example, if deploying AppScale over Eucalyptus, it is critical to ensure that Eucalyptus can deploy other images without error before trying to deploy the AppScale image to it. Here, it is vital to ensure that images can acquire both public and private IP addresses correctly, and that these IP addresses resolve in a sensible manner (e.g., that the public IP addresses actually resolve to the machine running the AppScale Tools). We have encountered many problems where users have had DHCP misconfigured for Eucalyptus or a failure to allocate enough disk space for Eucalyptus images: it is thus crucial to ensure that Eucalyptus is working correctly before attempting to run AppScale over it.

### 2.4 Community Support

To this point, we have described only the surface of AppScale: there is much more beneath the surface. We maintain a Google Code site at `http://code.google.com/p/appscale` that contains a comprehensive and up-to-date listing of each of the commands that users can employ to deploy and manage their AppScale deployments as well as the state of supported APIs. This site also contains information about published papers that use AppScale for research as well as add-ons to the AppScale platform.

On this Google Code site, users can also find a link to the most recent pre-built AppScale Xen image. Due to the large image size, we are not able to host the image on Google Code itself, but the link does provide a BitTorrent [2] file that allows for a quick download of the image itself as well as other necessary files.

We also support an active mailing list hosted by Google Groups for the AppScale Community, which can be found at `http://groups.google.com/group/appscale\_community`. Users that have problems deploying AppScale are advised to search this mailing list for answers to their problems and ask their own if others have not reported the problem thus far. Users wishing to do so are advised to attach the output of the problematic command as well as the verbose log, found on their master node.

Links to all these sites, as well as others, can be found on our main site, `http://appscale.cs.ucsb.edu`. This site also hosts our "apt" repository, where the most recent version of the AppScale installation packages is available.

## 3 Using AppScale

In this section, we discuss the critical services that make up an AppScale deployment and how they serve the applications that execute over the platform and employ the AppScale APIs.

### 3.1 Database Services

An AppScale deployment ensures that a database is started automatically and done so for a variable number of machines. Each database can be started and stopped by the AppController in a repeatable fashion, and can do so for a variety of database technologies. Each database has an implementation of the AppScale DB API of PUT, GET, DELETE, and QUERY. There are three services which use this interface.

First is a Protocol Buffer Server (PBServer), which receives Protocol Buffers from an App Engine application's database calls (Protocol Buffers are Google's internal data serialization method). The PBServer handles requests from different applications and implements the transactional middleware used to provide database agnostic transactions.

Second, is the User/App Server (UAServer). The UAServer uses the AppScale DB API to access a users and applications table and provides SOAP functions for simple RPC calls. All persistent data for the Users API is stored here. Data about an application, its ips, ports, admins, etc are stored here. Moreover, XMPP accounts and Channel API session information is both retrieved and committed to the UAServer (all considered application data).

Lastly, blobs which are uploaded using the blobstore service are stored in the database in 1MB chunks. A Blobstore Server is running on all nodes which are hosting applications. POST of blob files are forwarded to this service and once the file has completed uploading, the service redirects the user back to the application's blobstore upload success handler.

### 3.2 Monitoring Services

AppScale employs a piece of monitoring software called Monitr. It uses collectd to get standard metric information such as CPU, disk utilization, memory usage, and network access for all nodes in the deployment. A link to Monitr is present on the status page hosted by the load balancer. Furthermore, AppScale employs heartbeat checks to verify that critical components are functional. If a component is not responsive, or the process has died, AppScale will revive the component.

### 3.3 Neptune

Neptune is a domain specific language supported by AppScale. With Neptune, users can dictate workflows for certain languages, platforms, and scientific applications. These include but are not limited to X10, R, MPI, and Hadoop MapReduce. Users of Neptune also have the capability to control execution of jobs between different clouds (hybrid cloud execution).

## 4 Limitations and Future Work

There are several current limitations in AppScale that developers should be aware of before deploying AppScale and their applications. The list of limitations is as follows; for each, we provide possible work-arounds and/or our plan for addressing them.

- Persistence: If AppScale is on a virtualized platform, terminating instances can cause a loss of data. To address this issue we are implementing a bulk uploader and downloader similar to the one included with Google App Engine.
- Blobstore Max File Size: 100MB. This value is configurable within the code and we will allow for the setting of this value in a configuration file.
- Datastore: AppScale does not index data but rather does filtering of queries in-memory. As the size of the database gets larger, users may see a decrease in performance for queries. We are working on an indexing system which can create indexes yet stay compliant with the ACID semantics needed for transactions.
- Task Queue: Tasks which are enqueued are not fault tolerant, nor does AppScale handle delayed workers. The Task Queue will be reimplemented using an open source distributed queue technology.
- Mail: Only the administrator is allowed to send mail, and reception of mail is not implemented. Extensions to this API are on our road map.
- OAuth API is not implemented in AppScale. We have this API on our road map.
- AppServers currently follow a "deploy on all nodes" method and do not scale up or down as needed. We will allow for pluggable scheduling modules which will dictate dynamic placement in the future.
- Ubuntu is the only distribution supported. This is to limit testing allowing for faster release cycles. AppScale is portable to other distributions and in the past has successfully run on Red Hat.

- The Java AppServer does not have support for the Blobstore or Channel APIs. Google has not released the source code of the Java App Engine server, making it difficult to add new services. For each new feature we must decompile, add the feature, and recompile. We are working with Google to have this source code released.
- Some datastores do no have a method of retrieving the entire table to run a query. They must use a set of special keys which track all keys in the table. These datastores therefore have the added overhead of always accessing the special keys. These datastores are: MemcacheDB, Voldemort, SimpleDB, and Scalaris. We recommend using either Cassandra, HBase, Hypertable, or MySQL Cluster because they do not suffer from this limitation.

### 4.1 Future Directions

We began investigating AppScale as a web platform for executing Google App Engine applications without modification, in a scalable, efficient manner. We are continuously working to improve the design and implementation of AppScale to address the limitations above and to increase transparency, performance and scale. In addition, we have been focused on the automation and control of, as well as more coordinated interaction (for scheduling, automatic service-level agreement negotiation, elasticity, etc.) with cloud infrastructures (e.g. Amazon EC2 and Eucalyptus).

As part of future work, we are extending AppScale with new services for large-scale data analytics as well as data and computation intensive tasks. In addition, we are investigating a wide range of hybrid cloud technologies to facilitate application development and deployment that is cloud-agnostic. This entails new services, APIs, scheduling, placement, and optimization techniques. Finally, we are investigating new language support, performance profiling, and debugging support for cloud applications as well as the integration of mobile device use within the platform.

In summary, we have presented AppScale, its design, implementation, and use. We have detailed the APIs that AppScale supports and how they relate to Google App Engine. Moreover, we have described extensions which are specific to AppScale that expand the applicability of the platform, while retaining the simplicity and automation tht clouds offer. We encourage readers to try AppScale either using the readily available AMI image or the Xen Image on our hosting site `http://code.google.com/p/appscale/`. We appreciate all feedback and suggestions that users can provide through our community mailing list `http://groups.google.com/group/appscale_community`.

## 5 Acknowledgements

This work was funded in part by Google, IBM, and the National Science Foundation (CNS/CAREER-0546737, CSR-0905273, and CNS-0627183).

## References

[1] Appscale tools wiki page. `http://http://code.google.com/p/appscale/wiki/AppScale_Tools_Usage`.

[2] Bittorrent web site. `http://www.bittorrent.com`.

[3] C. Bunch, N. Chohan, C. Krintz, J. Chohan, J. Kupferman, P. Lakhina, Y. Li, and Y. Nomura. An Evaluation of Distributed Datastores Using the AppScale Cloud Platform. In *IEEE International Conference on Cloud Computing*, Jul. 2010.

[4] Cassandra. `http://incubator.apache.org/cassandra/`.

[5] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. AppScale: Scalable and Open AppEngine Application Development and Deployment. In *ICST International Conference on Cloud Computing*, Oct. 2009.

[6] Amazon elastic compute cloud – `http://aws.amazon.com/ec2/`.

[7] ejabberd. `http://www.ejabberd.im/`.

[8] Eucalyptus home page. `http://eucalyptus.cs.ucsb.edu/`.

[9] Transactions. http://code.google.com/appengine/docs/python/datastore/transactions.html.

[10] GAE Downloads. http://code.google.com/appengine/downloads.html.

[11] Google AppEngine. http://code.google.com/appengine/.

[12] HBase. http://hadoop.apache.org/hbase/.

[13] Hypertable. http://hypertable.org.

[14] MemcacheDB. http://memcachedb.org/.

[15] MongoDB. http://mongodb.org/.

[16] MySQL. http://www.mysql.com.

[17] Scalaris. http://code.google.com/p/scalaris/.

[18] SimpleDB. http://http://aws.amazon.com/simpledb/.

[19] Virt-Install. http://manpages.ubuntu.com/manpages/lucid/man1/virt-install.1.html.

[20] VMBuilder. http://manpages.ubuntu.com/manpages/lucid/en/man1/vmbuilder.1.html.

[21] Voldemort. http://project-voldemort.com/.

[22] YAML. http://www.yaml.org.

[23] Zookeeper. http://http://hadoop.apache.org/zookeeper/.