

# appscale

open source Google App Engine

## Analyse und Installation des Plattformdienstes AppScale

Studienarbeit  
von

**Christian Kumpe**

an der Fakultät für Informatik,  
Institut für Telematik,  
Lehrstuhl für Verteilte und Parallele Hochleistungssysteme,  
Karlsruher Institut für Technologie

Verantwortlicher Betreuer:	Prof Dr. Achim Streit
Betreuer:	Dr. habil. Marcel Kunze
Betreuer:	M. Sc. Christian Baun



---

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 12. September 2011



# Zusammenfassung

Diese Arbeit beschäftigt sich mit AppScale (AS), einer freien Reimplementierung des Cloud-Plattformdienstes Google App Engine (GAE). Schwerpunkt der Arbeit ist die Analyse von AS und die Installation in öffentlich zugänglichen und privaten Cloud-Infrastrukturdiensten. Es werden Problemstellungen analysiert und Lösungsmöglichkeiten evaluiert. Die Arbeit beschäftigt sich schwerpunktmäßig mit der Version 1.3 von AS, die zum Zeitpunkt der Arbeit aktuell war.



# Inhaltsverzeichnis

<b>Zusammenfassung</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Cloud Computing</b>	<b>3</b>
2.1 Wichtige Merkmale . . . . .	3
2.2 Funktionale Unterscheidung . . . . .	4
2.2.1 Infrastrukturdienst (IaaS) . . . . .	4
2.2.2 Plattformdienst (PaaS) . . . . .	4
2.2.3 Software-Dienst (SaaS) . . . . .	4
2.3 Organisatorische Unterscheidung . . . . .	4
2.3.1 Öffentlich verfügbare Dienste . . . . .	4
2.3.2 Private Dienste . . . . .	5
2.3.3 Hybride Dienste . . . . .	5
<b>3 Google App Engine</b>	<b>7</b>
3.1 Dienste der Google App Engine . . . . .	7
<b>4 AppScale (AS)</b>	<b>11</b>
4.1 Einschränkungen . . . . .	12
4.2 Unterstützung der APIs . . . . .	12
4.3 Schwierigkeiten bei der Installation . . . . .	13
4.4 Schritt für Schritt . . . . .	13
<b>5 Vergleichsmessungen</b>	<b>15</b>
<b>6 Zusammenfassung und Ausblick</b>	<b>19</b>
<b>A Der Benchmark im Quellcode</b>	<b>21</b>
<b>Literaturverzeichnis</b>	<b>25</b>





# Abbildungsverzeichnis

5.1	Der <i>noop</i> -Test . . . . .	16
5.2	Der <i>count</i> -Test . . . . .	17
5.3	Der <i>datastorePut</i> -Test . . . . .	17
5.4	Der <i>datastoreQueryAndDelete</i> -Test . . . . .	18
5.5	Der <i>memcacheIncr</i> -Test . . . . .	18



# Tabellenverzeichnis

3.1	Beschränkungen in der Google App Engine (GAE) . . . . .	7
4.1	Verfügbare Datenbanken für AS . . . . .	12
4.2	Von AS unterstützte APIs der GAE . . . . .	13



# Listings

4.1	Umgebungsvariablen	13
4.2	AppScale starten	14
4.3	Hochladen der Webanwendung	14
A.1	Quellcode des Benchmarks	21
A.2	Deskriptor	23



# 1. Einleitung

Einer der aktuellen Trends in der Informationstechnologie (IT) ist das *Cloud Computing*. Vorreiter auf diesem Gebiet sind Unternehmen wie z. B. Amazon mit seinem Infrastrukturdienst Amazon Elastic Compute Cloud (EC2) und Google mit seinem Plattformdienst Google App Engine (GAE).

Die Einschränkungen und Risiken beim Datenschutz und ungewollte Anbieterbindung bei öffentlich verfügbaren Cloud-Diensten führten zur Entwicklung von Lösungen zum Betrieb privater Cloud-Dienste. Deren Etablierung wird durch die Unterstützung verbreiteter Schnittstellen und damit des Ökosystems verfügbarer Werkzeuge begünstigt.

Erst im Nachhinein starteten Entwicklungen, unter anderem aus dem Forschungsumfeld, um freie Alternativen zu den bereits etablierten Cloud-Diensten zu schaffen. Auf der Ebene der Plattformdienste begann mit AppScale (AS) ab 2009 ein freier Nachbau der GAE. In dieser Arbeit wird dieses Projekt bzw. seine Ergebnisse untersucht. Zur Evaluierung der Software wurde in verschiedenen Infrastrukturdiensten und direkt auf Basis der Virtualisierungslösung VMware getestet. Ziel war der Betrieb des *KOALA Cloud Managers* in AS in den verschiedenen Umgebungen.

Dem frühen Entwicklungsstand waren viele Fallstricke geschuldet, deren Lösungen in dieser Arbeit Seiten aufgezeigt werden.





## 2. Cloud Computing

In diesem Kapitel werden die Merkmale des Cloud Computing und der angebotenen Dienste kurz vorgestellt und kategorisiert. Dabei gibt es zwei wesentliche Arten die Dienste in der Cloud zu unterscheiden:

- Funktionale Unterscheidung
- Organisatorische Unterscheidung

### 2.1 Wichtige Merkmale

Die folgenden Punkte sind zentrale, gemeinsame Merkmale des Cloud Computings [Köh10, S.4]:

- *On Demand*: Die bereitgestellten Ressourcen werden bei Bedarf abgerufen und können meist ohne großen Vorlauf gebucht werden.
- *Pay-as-you-go-Prinzip*: Dem Benutzer werden nur die verwendeten Ressourcen in Rechnung gestellt. Die Abrechnung erfolgt meist feingranular, etwa auf Stundenbasis.
- *Skalierbarkeit*: Der Benutzer kann immer genau so viele Ressourcen anmieten, wie er gerade benötigt. Nicht mehr benötigte Ressourcen kann er jederzeit freigeben.
- *Entkopplung der Rollen*: Der Nutzer von Ressourcen muss nicht mehr gleichzeitig der Besitzer sein. Der Anbieter regelt den Betrieb der Ressourcen und die Kunden nutzen diese.
- *Economies of Scale*: Weil viele Benutzer die Ressourcen des Anbieters gemeinsam verwenden, kann dieser die Ressourcen in großem Umfang und wirtschaftlicher anbieten, als dies einem einzelnen Nutzer möglich wäre.
- *Commodity Hardware*: In den meisten Fällen werden beim Cloud Computing die Dienste durch den Anbieter nicht durch teure spezialisierte Serverhardware zur Verfügung gestellt, sondern der Betrieb der Dienste erfolgt meist auf Standardhardware, die in er Anschaffung günstiger ist.

## 2.2 Funktionale Unterscheidung

Eine wichtige Möglichkeit Cloud-Dienste zu unterscheiden ist die Einordnung in funktionale Schichten. Dabei bauen in vielen Fällen die Dienste der höheren Schichten auf einem Dienst der darunterliegenden Schicht auf.

### 2.2.1 Infrastrukturdienst (IaaS)

Beim *Infrastrukturdienst* (engl. *Infrastructure as a Service, IaaS*) werden dem Benutzer die Dienste in Form virtueller Maschinen bereit gestellt. Er erhält administrativen Zugriff auf die angeforderten Instanzen. Dabei handelt es sich z.B. um Basisinstallationen des gewählten Betriebssystems, etwa Linux oder Windows-Server-Installationen. Es werden aber auch Installationen angeboten, die bereits höherwertige Dienste enthalten, etwa fertig installierte Datenbanken oder Webserver. Die Leistung der virtuellen Maschine, deren Plattenplatz und die Anzahl der CPUs kann je nach Bedarf ausgewählt werden und wird entsprechend abgerechnet. Ein bekanntes Beispiel für einen Infrastrukturdienst ist EC2[EC211].

### 2.2.2 Plattformdienst (PaaS)

Nimmt der Benutzer einen *Plattformdienst* (engl. *Platform as a Service, PaaS*) in Anspruch, wird ihm dabei direkt ein höherwertiger Dienst angeboten, ohne dass er sich um die Verwaltung der darunterliegenden Server-Instanzen kümmern muss. Diese Dienste werden meist über die APIs verschiedener Programmiersprachen angeboten. Um das ausfallsichere Bereitstellen und Skalieren des angebotenen Dienstes kümmert sich der Anbieter. Ein Beispiel ist die im nächsten Kapitel vorgestellte GAE[GAE11].

### 2.2.3 Software-Dienst (SaaS)

In der höchsten Ebene, dem *Software-Dienst* (engl. *Software as a Service, SaaS*) werden dem Benutzer fertige Anwendungen bereitgestellt. Diese werden meist über Browser-Oberflächen bedient und sind damit über das Internet erreichbar, ohne dass diese lokal beim Anwender installiert werden müssen. Die im Internet verfügbaren Office Werkzeuge von Google sind ein Beispiel für diese Schicht.

## 2.3 Organisatorische Unterscheidung

Eine andere Unterscheidungsmöglichkeit für Cloud-Dienste bietet die organisatorische Sichtweise. Dabei liegt der Fokus auf dem Betreiber des zur Verfügung gestellten Dienstes und dessen Verfügbarkeit.

### 2.3.1 Öffentlich verfügbare Dienste

Die öffentlich verfügbaren Dienste werden meist von speziellen Cloud-Anbietern zur Verfügung gestellt und können von jedermann über das Internet, meist gegen Bezahlung, genutzt werden. Der große Vorteil durch diese Trennung von Benutzer und Anbieter liegt in dem bereits erwähnten *Economies of Scale*.

### 2.3.2 Private Dienste

Hier werden die Cloud-Dienste organisationsintern zur Verfügung gestellt. Dabei werden meist die gleichen oder ähnliche APIs verwendet, wie dies bei den öffentlich verfügbaren Cloud-Diensten der Fall ist. Allerdings stehen die Dienste Benutzern außerhalb der Organisation nicht zur Verfügung. Durch diese Abschottung und den internen Betrieb behält die Organisation die Kontrolle über den Betrieb und die verarbeiteten Daten. Dies ist in vielen Ländern zur Einhaltung gesetzlicher Bestimmungen beim Datenschutz oder zur Wahrung von Firmengeheimnissen oft unabdingbar.

### 2.3.3 Hybride Dienste

Bei den hybriden Diensten werden die beiden zuvor genannten Formen vermischt. So können unkritische Daten problemlos und meist kostengünstiger in öffentlich verfügbaren Cloud-Diensten verarbeitet werden, während die kritischen Daten in der internen Cloud verbleiben.

Ein anderer Anwendungsfall ist das so genannte Cloud Bursting. Dabei geschieht die Verarbeitung der Daten weitestgehend in der internen Cloud. Werden dort die verfügbaren Ressourcen knapp, wird zusätzlich ein Teil der Verarbeitung in öffentlich verfügbare Dienste ausgelagert. Dieser Übergang kann nahezu nahtlos erfolgen, wenn sich die APIs der privaten und öffentlichen Dienste nicht unterscheiden.



## 3. Google App Engine

Mit der GAE stellt die Firma *Google* den Entwicklern von Webanwendungen einen Plattformdienst bereit. Bei der Nutzung steht ein gewisses Kontingent an Ressourcen (*Quota*) pro Tag kostenlos zur Verfügung. Einige Zahlen zu den Beschränkungen der GAE sind in Tabelle 3.1 zu finden. Gezählt werden dabei etwa Zugriffe auf die Datenbank, die übertragene Datenmenge oder die Anzahl der Anfragen an den Server. Durch weitere Vereinbarungen mit Google können auch die bei Bezahlung angegebenen Obergrenzen noch erweitert werden.

Ressource	freies Kontingent	maximales Kontingent
Serveranfragen	43.200.000	43.200.000
gesendete Daten [GB]	1	1.046
empfangene Daten [GB]	1	1.046
CPU-Zeit [h]	6,5	1.729

Tabelle 3.1: Beschränkungen in der Google App Engine (GAE)

Der entscheidende Vorteil des kostenlosen Kontingents ist, dass bei kleineren Anwendungen und während der Entwicklung und Testphase noch keine Kosten anfallen. Auch muss die Bezahlung bei Bedarf explizit aktiviert werden, so dass der Entwickler in der Startphase kein finanzielles Risiko eingehen muss.

Zum Aufbau seiner Anwendung muss der Entwickler nur über ein Google-Benutzerkonto verfügen und kann bis zu 10 Webanwendungen unter der Domain [appspot.com](https://appspot.com) online stellen.

### 3.1 Dienste der Google App Engine

Grundlage für die GAE ist in Java ein eingeschränkter Servlet-Container, bei Python das Django-Framework. Um die durch die Anfrage (engl. Requests) eingehenden Daten weiter zu verarbeiten und zu speichern bzw. auf verschiedene Art und Weise mit dem Benutzer zu interagieren, bietet die GAE den Benutzern verschiedene Dienste.

- *Datastore*: Der Datastore ist eine sogenannte NoSQL- bzw. spaltenorientierte Datenbank nach dem Prinzip der BigTable[CDGH+08]. Ein Datensatz, also eine Zeile in der Tabelle, kann beliebige Attribute haben und muss keinem festen Schema genügen.

Um die Konsistenz der Daten muss sich die Webanwendung selbst kümmern. Allerdings ist so auch ein einfaches Erweitern des Datenmodells um fehlende Attribute möglich. Der Datastore bietet auch Transaktionen um Zugriffe gegenseitig abzuschirmen. Um dem Entwickler die Arbeit zu erleichtern existieren Mapper, die den Datastore als JPA- oder JDO-kompatiblen Dienst zur Verfügung stellen.

- *Blobstore* Hier können größere Mengen an Binärdaten abgelegt werden. Da es in der GAE nicht möglich ist, auf das lokale Dateisystem zuzugreifen. Der Speicher ist primär für Bilder gedacht. Der Blobstore ist so konzipiert, dass der Anwender über Webformulare die Daten direkt in den Blobstore überträgt und damit nicht die Server-Infrastruktur belastet.
- *Images*: Mit der Images-API kann einfache Bildverarbeitung betrieben werden. So können Bilder z.B. skaliert, rotiert und beschnitten werden.
- *Mail*: Mit der Mail-API bietet die GAE einen Dienst zum Empfangen und Versenden von Emails. Dabei werden eingehende Emails, ähnlich wie bei XMPP-Nachrichten, auf Zugriffe zu speziellen URLs weitergeleitet und können dort von der Webanwendung entgegengenommen werden. Das Versenden von Emails unterliegt starken Einschränkungen, um den Missbrauch zu verhindern. So muss entweder einer der eingetragenen Administratoren als Absender gesetzt sein, oder es können nur Emails an den gerade aktiven Benutzer geschickt werden.
- *Memcache*: Hier können über eine standardisierte API (JSR-170) Zwischenergebnisse abgelegt werden. Somit können stark frequentierte Webanwendungen, etwa generierte Seiteninhalte im Memcache zwischenspeichern, um sie nicht bei jedem Zugriff neu erstellen zu müssen. Der Zugriff auf den Memcache erfolgt deutlich schneller als auf den Datastore, weil er ausschließlich aus Hauptspeicherbausteinen besteht. Allerdings gibt es keine Garantien wie lange die Daten im Memcache verbleiben.
- *Urlfetch*: Der Urlfetch-Dienst bietet einem Programm in der App Engine die Möglichkeit, Daten von entfernten HTTP-Servern zu laden.
- *Users*: Über den User-Dienst können Benutzer authentifiziert werden. Dabei muss der Benutzer über ein Google-Benutzerkonto verfügen.
- *XMPP*: Mit dem XMPP-Dienst können „Jabber-Nachrichten“ empfangen und versendet werden. Ähnlich wie bei eingehenden Emails wird der Empfang einer Nachricht über einen Zugriff auf eine spezielle URL verarbeitet. Nach außen hin stellt jede Instanz in der GAE einen XMPP-Client dar, der ständig online ist und Authentifizierungsanfragen immer mit einer Zustimmung beantwortet. Über XMPP können etwa Administratoren einer Webanwendung über außergewöhnliche Ereignisse informiert werden.

- 
- *Taskqueue*: Genau wie eingehende Emails und XMPP-Nachrichten wird auch die Ausführung von Tasks in der Taskqueue über Zugriffe auf festgelegte URLs realisiert. Die Zugriffe werden über spezielle Regeln definiert, die etwa ein regelmäßiges Ausführen der Task in bestimmten Zeitintervallen zulassen.





## 4. AppScale (AS)

AppScale (AS) ist eine quelloffene Reimplementierung der GAE und siedelt sich damit auf der PaaS-Ebene an. AS wird am RACELab der University of California in Santa Barbara entwickelt. Die Ziel-Plattformen sind EC2 oder deren freie Reimplementierung Eucalyptus.

Damit kann AS vom Benutzer sowohl in der öffentlich verfügbaren Cloud bei Amazon oder in einer privaten Cloud, mit Eucalyptus betrieben werden. Das bietet dem Benutzer nicht nur die Möglichkeit einer Anpassung an seine Bedürfnisse und verhindert eine zwingende Bindung an einen festen Anbieter (engl. Vendor Lockin), sondern ermöglicht auch ein besseres Monitoring und Debugging der Webanwendung vor der Freigabe auf Googles Plattformdienst.

Um eine möglichst weitgehende Kompatibilität zu gewährleisten, baut AS direkt auf dem SDK von Google auf. Dabei wird zur Laufzeit jeweils eine Instanz des Development-Servers pro Sitzung gestartet. Bei diesem Server wurden die im Hintergrund laufenden Dienste durch alternative Komponenten ersetzt. So wird z.B. die eingebaute Datenbank durch eine Anbindungen an etablierte, für den Produktivbetrieb geeignete, Datenbanken ersetzt. Hierbei lässt einem AS die Wahl zwischen aktuell sieben verschiedenen Produkten: HBase, Hypertable, MySQL-Cluster, Cassandra, Voldemort, MongoDB und MemcacheDB.

Es folgt eine kurze Zusammenfassung und Charakterisierung der Datenbanken, wobei alle Datenbanken quelloffen verfügbar sind. Eine zusammenfassende Übersicht findet sich Tabelle 4.

- *HBase*[HBa11] ist eine nichtrelationale, verteilte, in Java geschriebene Datenbank, die auf der Hadoop-Plattform[Had11] aufsetzt und nach dem Vorbild der BigTable[CDGH+08] entstanden ist.
- Auch *Hypertable*[Hyp11] orientiert sich an BigTable und kann auf verschiedenen verteilten Dateisystemen eingesetzt werden. Sie wurde zur Geschwindigkeitssteigerung in C++ implementiert.
- *MySQL*[MyS11] ist eine weit verbreitete relationale Datenbank von Oracle.

- *Apache Cassandra* [Cas11] wurde 2008 von Facebook veröffentlicht und orientiert sich ebenso am Design der BigTable.
- *Voldemort* [Vol11] ist eine NoSQL-Datenbank. Bei dieser Datenbank brachen die Benchmarks nachvollziehbar ab, meist wegen zu langer Anfragezeiten.
- *MongoDB* [Mon11] ist eine in C++ geschriebene dokumentenorientierte Datenbank.
- *MemcacheDB* [Mem11] ist eine verteilte Schlüssel/Wert-Datenbank.

Datenbank	Typ	getestet	Bemerkung
HBase	spaltenorientiert	✓	
Hypertable	spaltenorientiert	✓	
MySQL	relational	✓	
Cassandra	Schlüssel/Wert	✓	Standarddatenbank
Voldemort	Schlüssel/Wert	✓	schlechte Leistung und Benchmarks brechen häufig ab
MongoDB	dokumentenorientiert	✓	
MemcacheDB	verteilte Schlüssel/Wert-Datenbank	✓	

Tabelle 4.1: Verfügbare Datenbanken für AS

## 4.1 Einschränkungen

Wichtige Einschränkungen gegenüber der GAE sind die fehlende dauerhafte Persistenz und die Benutzerauthentifizierung über Google-Benutzerkonten. Beim ersten Punkt hat AS nachgebessert und in den neuen Versionen wird es möglich sein, den aktuellen Inhalt der Datenbank in eine Sicherungsdatei zu schreiben und beim nächsten Start wieder einzuspielen.

Eine weitere Einschränkung ist das Benutzermanagement. In der GAE können Benutzer über ihre bereits existierenden Google-Benutzerkonten authentifiziert werden. In einer AS-Installation besteht diese Möglichkeit nicht. Hier müssen die Benutzer erst ein eigenes Benutzerkonto anlegen, um die Webanwendung mit Authentifizierung nutzen zu können.

Außerdem startet AS wie oben bereits erwähnt pro installierter Webanwendung eine eigene Instanz des Entwicklungs-Servers der GAE. Diese hören jeweils auf eigenen Ports, auf einem der Server, auf denen sich die AS-Installation verteilt. Dadurch können Probleme mit Firewalls auftreten, die nur Zugriffe auf Webseiten mit den Standard-Ports für HTTP (80 und 443) erlauben.

## 4.2 Unterstützung der APIs

Von den bei der GAE bereitgestellten Diensten unterstützt AS momentan noch nicht alle und einige noch nicht in vollem Umfang. In Tabelle 4.2 befindet sich eine Zusammenfassung.

GAE-API	Unterstützt von AS
Datastore	✓
Blobstore	?
Images	✓
Mail	?
Memcache	?
Urlfetch	✓
Users	✓
XMPP	✓
Taskqueue	?

Tabelle 4.2: Von AS unterstützte APIs der GAE

### 4.3 Schwierigkeiten bei der Installation

Viele Schwierigkeiten bei der Installation bzw. der Inbetriebnahme von AS wurden in der Version 1.4 beseitigt. So ist man jetzt nicht mehr auf das Erstellen eines kompletten Images für Eucalyptus oder EC2 angewiesen sondern man kann AS auch in eine Installation mit Ubuntu Server integrieren. Dieses wurde im Rahmen dieser Arbeit in einer virtuellen Maschine durchgeführt.

Zudem wurde das Testen deutlich einfacher und ressourcenschonender durch die Möglichkeit, fast alle Datenbanken in einem Single Node Setup betreiben zu können.

Leider traten bei den Versuchen verschiedene, teils nicht reproduzierbare Probleme auf, was der prototypischen Softwarequalität von AS geschuldet ist.

Durch die relativ großen Images stellt AS auch eine Belastung für die verwendete Infrastruktur dar. So mussten die Experimente mit einer Single-Node-Eucalyptus-Installation abgebrochen werden, weil beim Einspielen der Images regelmäßig Fehler auftraten, so dass ein sinnvolles Testen zu diesem Zeitpunkt nicht möglich war.

### 4.4 Schritt für Schritt

Zuerst müssen die nötigen Umgebungsvariablen gesetzt werden. Das meiste davon übernimmt das Script der *ec2-tools*. Eine entsprechende Anleitung findet sich in Listing 4.1.

```
# verhindert Probleme mit der Internationalisierung
user@host:~$ export -n LANG

# Gibt die Zone an
user@host:~$ export EC2_URL="https://ec2.amazonaws.com"

# Wo liegen das EC2-Zertifikat
user@host:~$ EC2_KEY_DIR=$HOME/.ec2/

# Zertifikat und dazugehöriger privater Schlüssel
user@host:~$ export EC2_CERT="{EC2_KEY_DIR}/cert-xyz.pem"
user@host:~$ export EC2_PRIVATE_KEY="{EC2_KEY_DIR}/pk-xyz.pem"

# Access-Key und zugehöriger privater Schlüssel
user@host:~$ export EC2_ACCESS_KEY="YOURAWSACCESSKEY"
```

```
user@host:~$ export EC2_SECRET_KEY="AwsSecretKey"
```

Listing 4.1: Umgebungsvariablen

Das Starten der AS-Instanzen in EC2 ist in Listing 4.2 zu sehen.

```
user@host:~$ appscale-run-instances --max 1 --machine ami-044fa56d
Run instances message sent successfully. Waiting for the image to start up.
[Fri Nov 12 13:43:11 +0100 2010] 3599.999979 seconds left until timeout...
[Fri Nov 12 13:45:17 +0100 2010] 3473.20229 seconds left until timeout...
[Fri Nov 12 13:47:23 +0100 2010] 3347.547578 seconds left until timeout...
Please wait for your instance to complete the bootup process.
Head node successfully created at ec2-184-72-177-45.compute-1.amazonaws.com. It
is now starting up memcachedb via the command line arguments given.
Killing and starting server at ec2-184-72-177-45.compute-1.amazonaws.com
Please wait for the controller to finish pre-processing tasks.
```

Listing 4.2: AppScale starten

Im nächsten Schritt kann eine Anwendung in AS hochgeladen werden. Den Vorgang zeigt Listing 4.3.

```
user@host:~$ appscale-upload-app --email mail@example.com --file gaevsas.tar.gz
...
```

Listing 4.3: Hochladen der Webanwendung

## 5. Vergleichsmessungen

Um die Leistungsfähigkeit der AS-Plattform im Vergleich zur GAE einschätzen zu können, wurden in dieser Arbeit verschiedene Vergleichsmessungen vorgenommen. Dabei kam ein einfaches Python-Skript zum Einsatz, mit dessen Hilfe verschiedene Aspekte des Plattformdienstes getestet wurden:

- Beim *noop*-Test (siehe Abbildung 5.1) wird immer eine feste Zeichenkette zurückgegeben. Ziel ist eine grobe Abschätzung der Laufzeiten im Netzwerk. Zudem wurden die weiteren Messergebnisse durch die hier ermittelten Werte bereinigt um die unterschiedlichen Laufzeiten des Internets im Vergleich zum lokalen Netz auszugleichen.
- Der *count*-Test (siehe Abbildung 5.2) zählt im Skript eine Variable von 0 bis zum angegebenen Wert. Ziel ist eine grober Maßstab für die Leistungsfähigkeit der darunterliegenden Hardware-Plattform bzw. des verwendeten Python-Interpreters.
- Im *datastorePut*-Test (siehe Abbildung 5.3) wird pro Anfrage an den Server eine angegebene Anzahl von einfachen Dokumenten in der Datenbank abgelegt. Dadurch soll ein Bild über die Leistungsfähigkeit der Datenbankimplementierung gewonnen werden.
- Im anschließenden *datastoreQueryAndDelete*-Test (siehe Abbildung 5.4) werden die zuvor erzeugten Einträge in der Datenbank einzeln über eine Abfrage in der Datenbank gesucht, geladen und anschließend gelöscht. Auch hier steht die Leistungsfähigkeit der Datenbankimplementierung im Blick.
- Beim *memcacheIncr*-Test (siehe Abbildung 5.5) wird ein Zähler über die Memcache-API hochgezählt, um die Leistungsfähigkeit von deren Implementierung zu evaluieren.

Die Tests wurden zum einen in der GAE über eine DSL 16.000 Leitung durchgeführt. Die Messungen mit AS fanden im lokalen Netzwerk statt. AS wurde dazu auf einem 2,5 GHz AMD Athlon Dual Core in einer virtuellen Maschine unter KVM betrieben.

Bei einigen Tests traten wiederholt Probleme auf. AS bricht Anfragen an den Server nach maximal 60 Sekunden ab. Da bei einigen der verwendeten Datenbanken diese Grenze bereits bei sehr wenigen Datenbankzugriffen pro Anfrage überschritten wird, konnten einige Tests nicht stabil zu Ende gebracht werden. An den entsprechenden Stellen bricht die Messkurve frühzeitig ab.

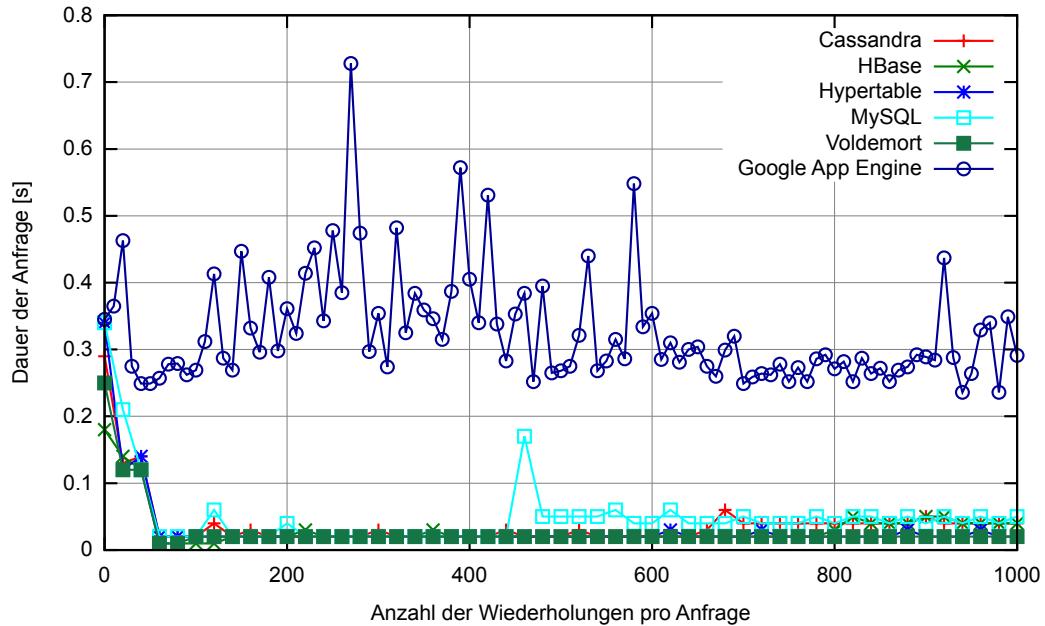
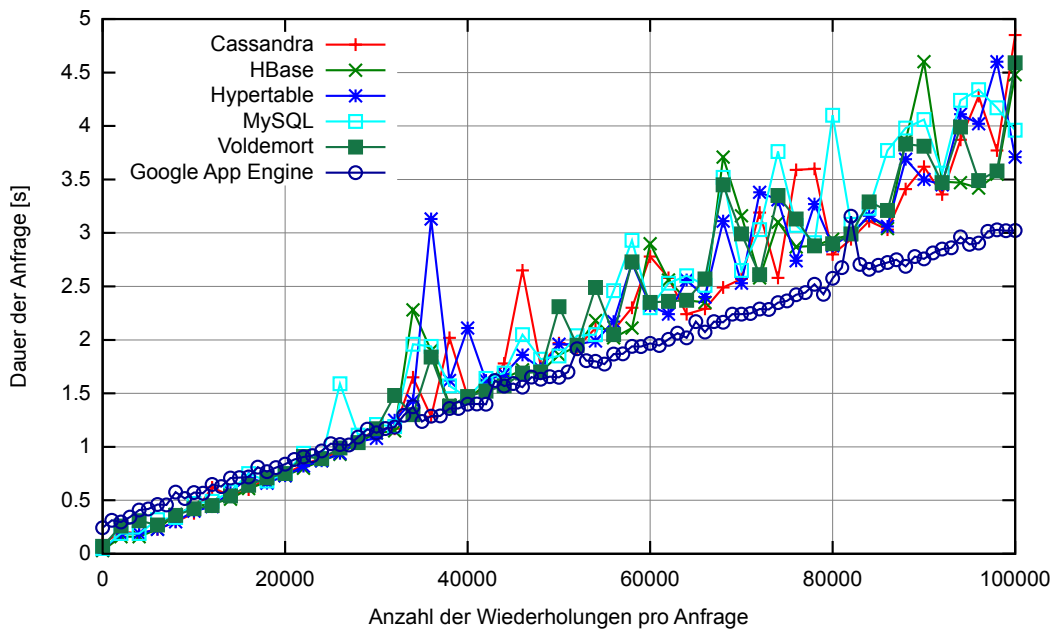
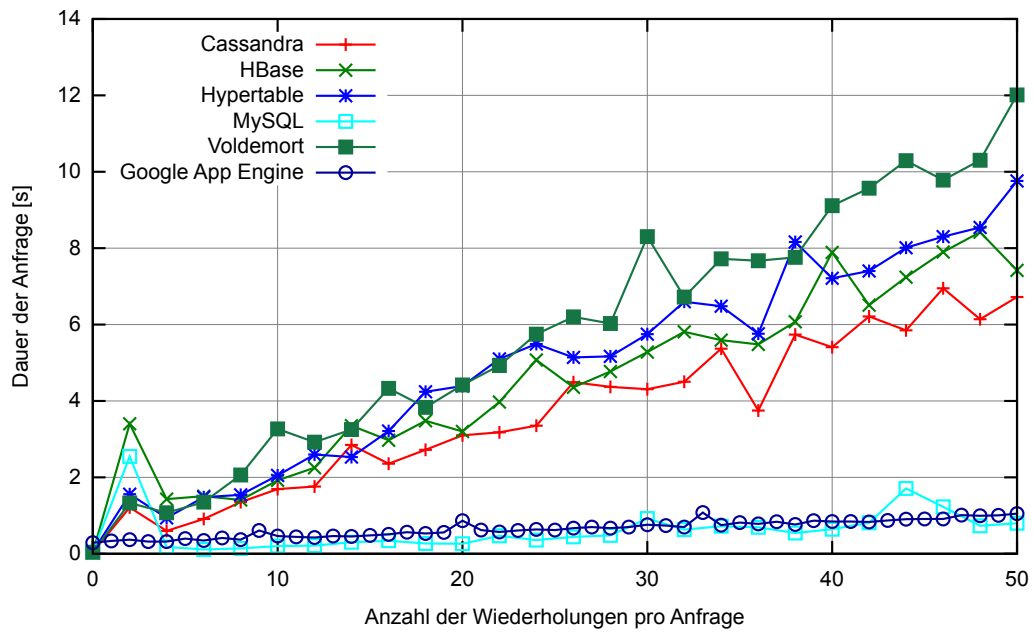
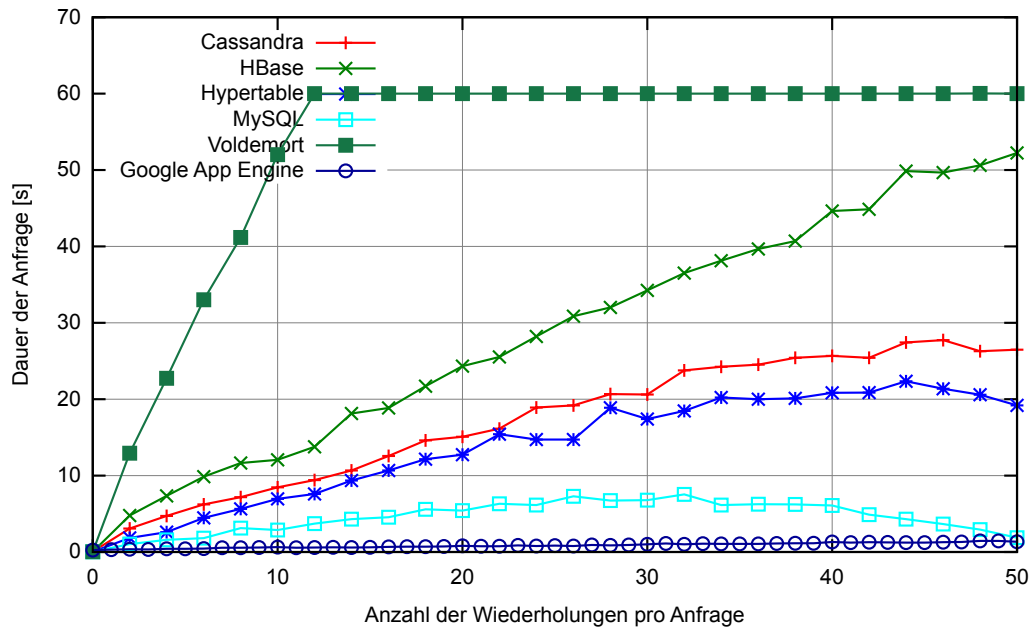
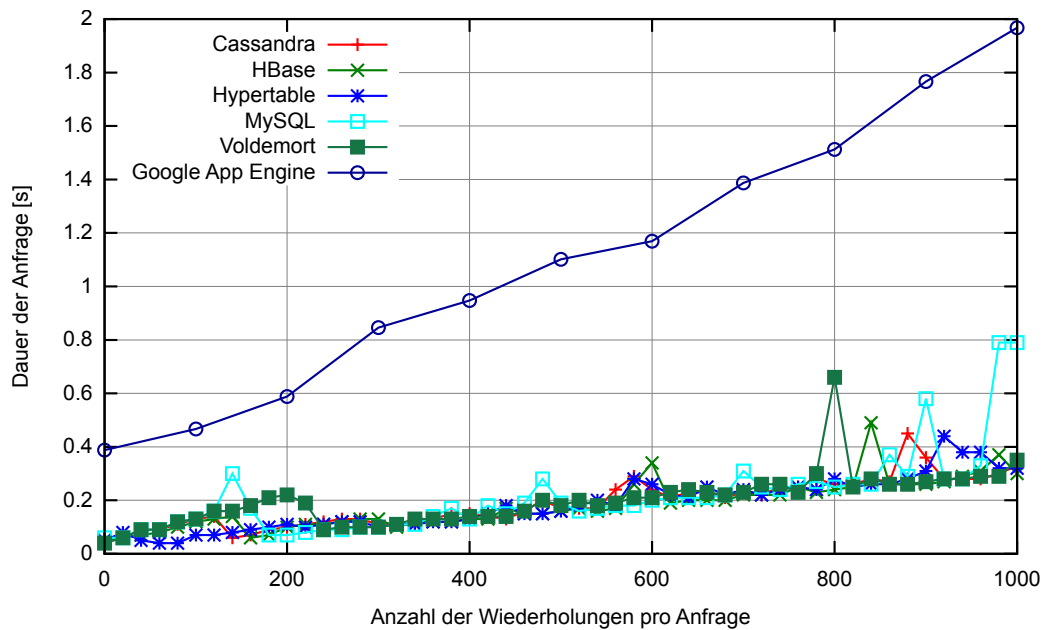


Abbildung 5.1: Der *noop*-Test

Abbildung 5.2: Der *count*-TestAbbildung 5.3: Der *datastorePut*-Test

Abbildung 5.4: Der *datastoreQueryAndDelete*-TestAbbildung 5.5: Der *memcacheIncr*-Test



## 6. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde AppScale (AS), einer alternativen quelloffenen Plattform zur Google App Engine (GAE), untersucht. Dabei wurde AS in verschiedenen Umgebungen gestartet und auf seine Funktionsfähigkeit überprüft. Am Ende wurden auf einer einfachen AS-Installation Leistungsmessungen durchgeführt um die Leistung der Plattform in ein grobes Verhältnis zur GAE stellen zu können. Dabei wurde erkannt, dass die Leistung einer AS-Installation maßgeblich von der verwendeten Datenbank abhängt und einzig mit MySQL eine Leistung erzielt werden kann, die auch bei mehr Datenbankabfragen pro Seitenabruf zufriedenstellend ist.

AS baut auf dem SDK der GAE auf, meist nicht auf der aktuellsten Version, so dass nicht immer alle APIs unterstützt werden. Allerdings sind die Entwickler von AS stets bemüht diese Lücken zu schließen. Vor allem die Mängel in der Benutzerverwaltung und das Weiterleiten der Anfragen auf verschiedene Netzwerk-Ports der laufenden Server ergeben nach außen hin noch kein so rundes Bild wie es bei der originalen GAE-Plattform der Fall ist. Daher ist AS zwar für den Betrieb bestimmter Intranet-Lösungen oder zum Testen von GAE-Anwendungen geeignet, der Betrieb einer öffentlich verfügbaren Plattform dürfte damit allerdings noch gewisse Schwierigkeiten bereiten.



# A. Der Benchmark im Quellcode

In diesem Anhang ist der Quellcode des verwendeten Benchmarks zu sehen. Die Implementierung erfolgte in Python.

```
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext import db
from google.appengine.api import memcache
import time

class Benchmark(webapp.RequestHandler):
    def get(self, *args):
        self.response.headers['Content-Type'] = 'text/plain;_charset=utf-8'
        startingTime= time.clock()
        functionName= args[0];
        try:
            function= getattr(self, functionName)
        except AttributeError:
            self.response.out.write( '404_NOT_FOUND')
            self.response.set_status(404)
            return;
        function( *args)
        endingTime= time.clock()
        duration= endingTime- startingTime
        self.response.out.write(
            'OK:_%s_seconds\n' % ( duration))
    def getCount(self):
        try:
            return int(self.request.get( 'count'))
        except:
            return 0
    def noop(self, *args):
        self.response.out.write( 'Nothing_to_be_done.\n')
    def count(self, *args):
        sum= 0
        i= 0
        while i< self.getCount():
            sum+= i
            i+= 1
        self.response.out.write( 'Sum_from_i=_0_to_%s_is_%s\n' % (i, sum))
```

```

def datastorePut(self, *args):
    class Entity(db.Model):
        data = db.StringProperty(required=True)
    i= 0
    while i< self.getCount():
        entity= Entity(
            data= "Entity_Nr._%s_from_%s" % (i, self.getCount()))
        entity.put()
        i+= 1
    self.response.out.write( 'Put_%s_entities_into_datastore.\n' % i)
def datastoreQueryAndDelete(self, *args):
    class Entity(db.Model):
        data = db.StringProperty(required=True)
    deletionCount= 0
    i= 0
    while i< self.getCount():
        query = db.GqlQuery(
            "SELECT_*_FROM_Entity_WHERE_data_=_%s:1",
            "Entity_Nr._%s_from_%s" % (i, self.getCount()))
        result = query.fetch( 1)
        for entity in result:
            entity.delete()
            deletionCount+= 1
        i+= 1
    self.response.out.write(
        'Queried_%s_times_and_deleted_%s_entities_from_datastore.\n'
        % (i, deletionCount))
def deleteAll(self, *args):
    class Entity(db.Model):
        data = db.StringProperty(required=True)
    deletionCount= 0
    query = db.GqlQuery("SELECT_*_FROM_Entity")
    result = query.fetch(1000)
    for entity in result:
        entity.delete()
        deletionCount+= 1
    self.response.out.write(
        'Deleted_%s_entities_from_datastore.\n' % (deletionCount))
def memcacheIncr(self, *args):
    memcache.set( "counter", 0)
    i= 0
    while i< self.getCount():
        memcache.incr( "counter")
        i+= 1
    self.response.out.write(
        'Incremented_memcache-counter_%s_times.\n'
        % memcache.get( "counter"))
application = webapp.WSGIApplication(['/(.*)', Benchmark], debug=True)
def main():
    run_wsgi_app(application)
if __name__ == "__main__":
    main()

```

Listing A.1: Quellcode des Benchmarks

---

Die nötige Konfigurationsdatei für die GAE:

```
application: gaevsas
version: 1
runtime: python
api_version: 1

handlers:
- url: .*
  script: main.py
```

Listing A.2: Deskriptor



# Literaturverzeichnis

- [Cas11] Apache Cassandra. Stand August 2011.  
<http://cassandra.apache.org>
- [CDGH<sup>+</sup>08] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes und R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* Band 26, June 2008, S. 4:1–4:26.
- [EC211] Amazon EC2. Stand August 2011.  
<http://aws.amazon.com/ec2>
- [GAE11] Google App Engine. Stand August 2011.  
<http://appengine.google.com>
- [Had11] Apache Hadoop! Stand August 2011.  
<http://hadoop.apache.org>
- [HBa11] Apache HBase. Stand August 2011.  
<http://hbase.apache.org>
- [Hyp11] Hypertable. Stand August 2011.  
<http://www.hypertable.org>
- [Köh10] J. Köhler. Analyse aktueller Forschungsschwerpunkte im Cloud-Computing anhand wissenschaftlicher Veröffentlichungen. Diplomarbeit, KIT, November 2010.
- [Mem11] MemcacheDB. Stand August 2011.  
<http://memcachedb.org>
- [Mon11] MongoDB. Stand August 2011.  
<http://www.mongodb.org>
- [MyS11] MySQL. Stand August 2011.  
<http://www.mysql.com>
- [Vol11] Voldemort. Stand August 2011.  
<http://project-voldemort.com>