— Under the Hood of the Google App Engine —

Cloud Computing PaaS Middleware

Christopher A. Wood
`www.christopher-wood.com`
`woodc1@uci.edu`

May 5, 2014

Agenda

Motivation

Google App Engine Middleware

Front-End

Back-End

- Data Storage
- Process and Application Communication
- Process and Task Management
- Misc Content

Wrapping Up

Introduction

A PaaS is, by definition, a set of middleware components between applications and the infracstructure on which they run to abstract away:

- Reliability issues
- Data redundancy guarantees
- Load balancing
- . . .

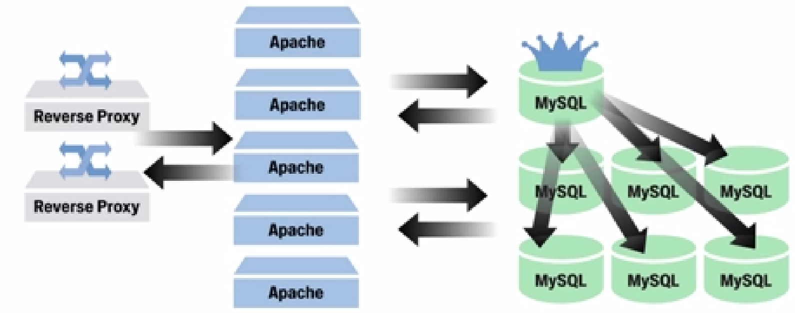
A New Application Infrastructure

The Google App Engine architecture is designed to provide application:

- Scalability: load balancing, asynchronous event handling, etc.
- Reliability: transparent fault tolerance
- Cost efficiency: application cost dynamic varies based on resource usage

How are traditional web applications built to achieve these three properties?...

Traditional Enterprise Application/Backend Designs

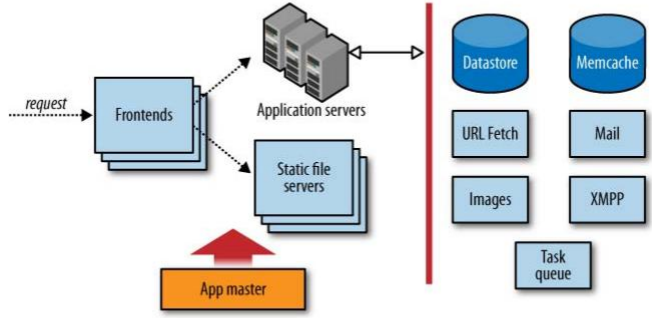


GAE Goal

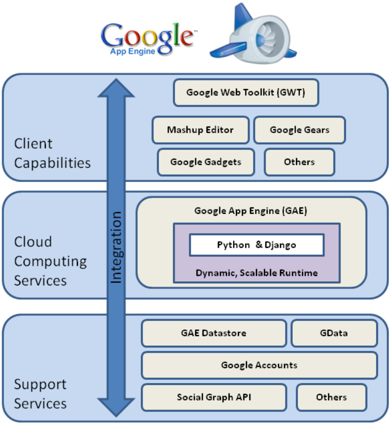
Question: How can we get the same *performance* with a significantly *less coupled* backend?

Answer: Google App Engine!

GAE Architecture



GAE Middleware Role



“Application-Side” Middleware Abstractions

The application design goals are abstracted by two primary components in the middleware:

- Front end
  - Enables “edge caching” at locations close to the user (load balancing)
  - Instaces dynamically scale based on the number of incoming requests
- App server
  - Runtime environment for application instances (basically, a VM to host your app)

Backend Middleware

There are many important components of the GAE backend:

- Data storage, retrieval, and search
- Communication mechanisms
- Process management
- Configuration and management

Data storage, retrieval, and search

- Google Cloud SQL
  - Relational database support for more “legacy” applicatons
- Datastore and blobstore vs memcache (and dedicated memcache)
- Search
  - Provides a model for indexing (datastore) documents that contain *structured* data, and an API for searching the index
- . . .

Datastore

A persistent storage for AppEngine

- BigData/NoSQL data model as opposed to traditional relational data model
- Queried using GQL (SQL-like query language developed by Google)
- Significantly better scalibility than relational data models
- Designed as a hierarchy of components: Datastore → MegaStore → BigTable

Datastore (cont'd)

Handles distribution, replication, and load balancing of data

- Based on the Paxos consensus algorithm
- **Scalability**: Automated sharding (i.e., striping) - BigTable
- **Reliability**: Replication via BigTable and transaction support (with ACID-guarantees) via MegaTable
- **Performance**: Enhcanced lock granularity and co-location of data for better concurrent access (e.g., tables, entities, properties)

Datastore (cont'd)

- Provides simple API to read/write entities backed by a powerful query engine and reliable transaction support
- Stores data as *entities*, which are uniquely identified by keys
  - Indidivual data elements are properties of entities (analogous to relation fields)
  - Entities can have parents and children
  - Enables a *hierarchical data model*, much like relational data models
  - Entities without parents are called *root entities*

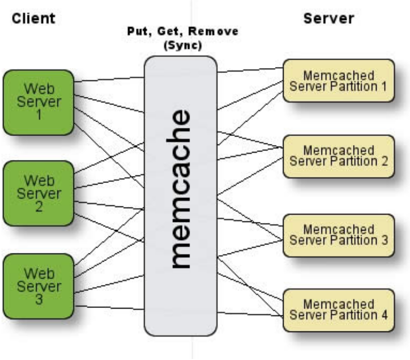
Paxos Consensus Algorithm

- Paxos is a consensus algorithm executed by a set of processes (replicas) to agree on a single value in the presence of failures.
- Composed of three stages:
  1. Elect a replica to be the coordinator (we've seen this - distributed elections).
  2. The coordinator selects a value and broadcasts it to all replicas in a message called the accept message. Other replicas either acknowledge this message or reject it (we've seen this too - load balanding).
  3. Once a majority of the replicas acknowledge the coordinator, consensus has been reached, and the coordinator broadcasts a commit message to notify replicas (yup, we've seen this as well - load balancing again).
- Google's implementation transformed a single page of pseudocode into several thousands of lines of C++ code.

Memcache

An in-memory, key-value data store

- Any serializable object/value can be inserted as a key or value


Client
Put, Get, Remove
(Sync)
Server
Web Server 1
Web Server 2
Web Server 3
memcache
Memcached Server Partition 1
Memcached Server Partition 2
Memcached Server Partition 3
Memcached Server Partition 4

Memcache (cont'd)

- Used for caching (obviously) datastore query results, user session information, etc.
- Also used for sharing data cross app instances
- APIs support general (atomic) read/write operations, as well as batch operations (e.g., writeAll(), readAll(), ...)
- Memcache **gets** be up to **10**x faster than the datastore **queries**
  - High memcache hit rates lead to *massive* application perforamnce
- Dedicated memcaches can be created (for a price) to store data specifically for your application
  - → no contention among other applications for cache space!

Memcache (cont'd)

Memcache sample usage:

```
def get_data():
    data = memcache.get('key')
    if data is not None:
        return data
    else:
        data = self.query_for_data()
        memcache.add('key', data, 60)
        return data
```

Communication mechanisms

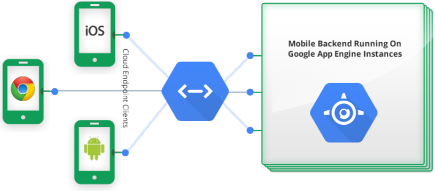
- Channels
- Google Cloud Endpoints
- Mail communication, URL fetch, XMPP (common instant messaging protocol)
  - Exposed via natural APIs for more “traditional” application or host-to-host (instance-to-instance) communication
  - Uses Google infrastructure for improved communication efficiency (e.g., for HTTP(s) GET requests issued to URLs)

Channel Communcication

- Creates a persistent connection between applications and Google servers
- Applications send messages to JavaScript clients in real time without “polling”
  - *Enables real-time updates to be pushed to clients*
- Used when asynchronous events/information needs to be pushed to clients
- Clients open channel connections to AppServer instances, and servers *push* data to all open channels when needed

Cloud Endpoint Communcication

- Enables automatic generation of APIs via a set of tools, libraries, and components
  - API code is *annotated* so that the GAE tools/libraries can automatically generate corresponding APIs for all platforms
- Used to create a *shared* web backend for web clients and mobile clients (e.g., Android or iOS)
  - Greatly reduces the amount of platform-specific work that needs to be done



Process management

Batch processing is handled by two primary components:

- Task queue
- Task scheduling

Task Queue

- Used to handle user tasks *outside* of a user request (e.g., execute the task asynchronously in the background)
- Tasks are configured according to a **Task** interface (e.g., Java task instances must implement the `Task` interface)
- Queues are onfigured on a per-pplication basis using YAML
  - Both *push* and *pull* queues can be defined
- Tasks can be enqueued as part of a datastore transaction
- Follows a “leaky bucket” approach to handle task bandwidth and burstiness submissions to AppServer instances
- RPC stubs (akin to Java `Future` objects) are returned when tasks are submitted so that task statuses can be queried

Task (Job) Scheduler

- Cron scheduler support for a variety of application languages
- Jobs can be scheduled to execute at periodic times (e.g., nightly batch processing tasks)
- GAE provides a tiered price plan for the number of allowable jobs that can be scheduled
- Jobs are configured with the `cron.xml` file
- Authorization rules can be built into the cron job

Configuration and Management

And of course, the GAE middleware provides the following additional features

- Application identity
- Remote application access
- User and application capabilities
- (custom domain) SSL certificate configuration
- Traffic splitting (e.g., *roll out* new features over time)
- User management (e.g., support login using Google accounts or OpenID)

GAE in One Sentence

GAE extends Google's scalable enterprise backend to your applications with a virtually transparent, flexible, and easy-to-use middleware.

References

All images taken from Google Developers documentation:

`https://developers.google.com/appengine/features/`