



香港城市大學
City University of Hong Kong

Lecture 5

Google App Engine

EE4221 Cloud Computing Systems
Nov. 5., 2015

Google App Engine

Datastore

Big query

Tasks queues

Google MapReduce

Dr. Anna Ruokonen
City University of Hong Kong

Agenda

- 1 Google App Engine
- 2 Datastore
- 3 Big query
- 4 Tasks queues
- 5 Google MapReduce



Google App Engine

Datastore

Big query

Tasks queues

Google MapReduce



- Web application development for Google cloud
- Platform (PaaS) and development tools
- Google Web Toolkit (GWT) for building graphical UIs
- Developer server for local debugging and development
- Admin Web console for management
- Deployment with Google account and application ID
- Basic usage is free of charge. For advanced features and large data, billing must be enabled.



香港城市大學
City University of Hong Kong

Google App Engine

Datastore

Big query

Tasks queues

Google MapReduce

- noSQL database Datastore
- Blobstore for storing large files
- Map reduce
- Big Query
- Task queues



- A fully managed NoSQL data storage service
- Data objects in the Datastore are known as entities
- An entity has one or more named properties
- Entities of the same kind need not have the same properties
- Also an entity's values for a given property need not all be of the same data type
- Each entity in the Datastore has a key that uniquely identifies it
- An application has access only to entities it has created itself
- Individual entities can be fetched using the entity key
- Or one or more entities can be fetched by a query based on the entity key or property values
- The Google Cloud Datastore API provides methods for creating, modifying and retrieving entities from the Datastore



Datastore example

```
class Guestbook(webapp2.RequestHandler):
    def post(self):

        guestbook_name = self.request.get('guestbook_name',
                                           DEFAULT_GUESTBOOK_NAME)
        greeting = Greeting(parent=guestbook_key(guestbook_name))

        if users.get_current_user():
            greeting.author = Author(
                identity=users.get_current_user().user_id(),
                email=users.get_current_user().email())

        greeting.content = self.request.get('content')
        greeting.put()

        query_params = {'guestbook_name': guestbook_name}
        self.redirect('/?' + urllib.urlencode(query_params))
```



- Allows storing large files
- The data amount is limited only by the data size that can be uploaded or downloaded over one single HTTP connection
- Applications do not create data blobs directly. They are created by uploading files via web forms or via or other HTTP POST requests
- These data objects are called Blobstore values or blobs
- Blobstore values can be provided to the user or accessed by the application using the Blobstore API



Blob example: File uploader

```
class UploadHandler(blobstore_handlers.BlobstoreUploadHandler):  
    """Handler to upload data to blobstore."""  
  
    def post(self):  
        source = "uploaded by user"  
        upload_files = self.get_uploads("file")  
        blob_key = upload_files[0].key()  
        name = self.request.get("name")  
  
        user = users.get_current_user()  
  
        username = user.nickname()  
        date = datetime.datetime.now()  
        str_blob_key = str(blob_key)  
        key = FileMetadata.getKeyName(username, date, str_blob_key)  
  
        m = FileMetadata(key_name=key)  
        m.owner = user  
        m.filename = name  
        m.uploadedOn = date  
        m.source = source  
        m.blobkey = str_blob_key  
        m.put()  
  
        self.redirect("/")
```

Blob example: File downloader

```
class DownloadHandler(blobstore_handlers.BlobstoreDownloadHandler):  
    def get(self, key):  
        key = str(urllib.unquote(key)).strip()  
  
        self.send_blob(key)
```



- SQL-like query language for large data sets
- BigQuery data is contained in Tables
- It comes along with a corresponding table schema, which defines, e.g., data fields and types
- API for managing data sets and tables, handling queries, etc.



BigQuery tables are created by

- loading data into a new table,
- running a query, or
- copying a table Datasets.



```
SELECT expr1 [[AS] alias1] [, expr2 [[AS] alias2], ...]  
  [agg_function(expr3) WITHIN expr4]  
  [FROM [(FLATTEN(table_name1|(subselect1)) [, table_name2|(subselect2), ...)]  
  [[INNER|LEFT OUTER|CROSS] JOIN [EACH] table_2|(subselect2) [[AS] tablealias2]  
    ON join_condition_1 [... AND join_condition_N ...]]+  
  [WHERE condition]  
  [GROUP [EACH] BY field1|alias1 [, field2|alias2, ...]]  
  [HAVING condition]  
  [ORDER BY field1|alias1 [DESC|ASC] [, field2|alias2 [DESC|ASC], ...]]  
  [LIMIT n]  
;
```

```
SELECT AVG(mother_age) AS avg_age, year  
FROM [publicdata:samples.natality]  
GROUP BY year  
ORDER BY year
```

```
select year, SUM(mother_age * record_weight) / SUM(record_weight)  
as age from publicdata:samples.natality  
where ever_born = 1 group by year order by year;
```



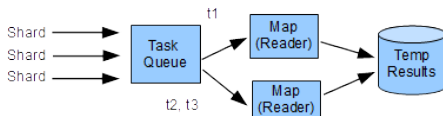
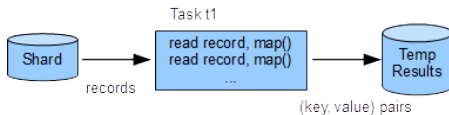
- An app can define task queues using a file named `queue.xml`
- The file specifies an element named *queue*
- It can contain any number of individual queues and define their processing rates



Tasks queue example: a task queue with an execute rate of 50 operation per second

```
queue:  
- name: default  
  rate: 50/s  
  bucket_size: 100  
  max_concurrent_requests: 100
```

- MapReduceJobs are based on Pipeline Jobs and Task Queues
- *Worker* is a base class for Mapper and Reducer
- Processes data for zero or more shards
- A job is characterized by the signature of its run method. A $\langle T, T_1, \dots, T_n \rangle$ -job is a job whose final output is of type T and that takes as input n arguments of types T_1 through T_n .



MapReduceJob specification

```
MapreducePipeline.run  
(  
    job_name,  
    mapper_spec,  
    reducer_spec,  
    input_reader_spec,  
    output_writer_spec=None,  
    mapper_params=None,  
    reducer_params=None,  
    shards=None  
)
```

WordCount example

```
def run(self, filekey, blobkey):

    bucket_name = app_identity.get_default_gcs_bucket_name()
    output = yield mapreduce_pipeline.MapreducePipeline(
        "word_count",
        "main.word_count_map",
        "main.word_count_reduce",
        "mapreduce.input_readers.BlobstoreZipInputReader",
        "mapreduce.output_writers.GoogleCloudStorageOutputWriter",
        mapper_params={
            "blob_key": blobkey,
        },
        reducer_params={
            "output_writer": {
                "bucket_name": bucket_name,
                "content_type": "text/plain",
            }
        },
        shards=4)
    yield StoreOutput("WordCount", filekey, output)
```



WordCount example cont.

```
def word_count_map(text):  
    for w in split_into_words(text.lower()):  
        yield (w, 1)
```

```
def word_count_reduce(key, values):  
    yield (key, len(values))
```



D. Sanderson

O'Reilly Media

Programming Google App Engine, 2nd Edition



Google App Engine: Python runtime environment URL:

`https:`

`//cloud.google.com/appengine/docs/python/`



Title: Google BigQuery URL:

`https://developers.google.com/bigquery/`