Case Study: PAAS (Face book, Google App Engine)

Theory:

Platform-as-a-Service (PaaS):

Cloud computing has evolved to include platforms for building and running custom web-based applications, a concept known as Platform-as-a- Service. PaaS is an outgrowth of the SaaS application delivery model. The PaaS model makes all of the facilities required to support the complete life cycle of building and delivering web applications and services entirely available from the Internet, all with no software downloads or installation for developers, IT managers, or end users. Unlike the IaaS model, where developers may create a specific operating system instance with homegrown applications running, PaaS developers are concerned only with webbased development and generally do not care what operating system is used. PaaS services allow users to focus on innovation rather than complex infrastructure. Organizations can redirect a significant portion of their budgets to creating applications that provide real business value instead of worrying about all the infrastructure issues in a roll-your-own delivery model. The PaaS model is thus driving a new era of mass innovation. Now, developers around the world can access unlimited computing power. Anyone with an Internet connection can build powerful applications and easily deploy them to users globally.

Google App Engine:

Architecture :

The Google App Engine (GAE) is Google`s answer to the ongoing trend of Cloud Computing offerings within the industry. In the traditional sense, GAE is a web application hosting service, allowing for development and deployment of web-based applications within a pre-defined runtime environment. Unlike other cloud-based hosting offerings such as Amazon Web Services that operate on an IaaS level, the GAE already provides an application infrastructure on the PaaS level. This means that the GAE

abstracts from the underlying hardware and operating system layers by providing the hosted application with a set of application-oriented services. While this approach is very convenient for

developers of such applications, the rationale behind the GAE is its focus on scalability and usage-based infrastructure as well as payment.

Costs :

Developing and deploying applications for the GAE is generally free of charge but restricted to a certain amount of traffic generated by the deployed application. Once this limit is reached within a certain time period, the application stops working. However, this limit can be waived when switching to a billable quota where the developer can enter a maximum budget that can be spent on an application per day. Depending on the traffic, once the free quota is reached the application will continue to work until the maximum budget for this day is reached. Table 1 summarizes some of the in our opinion most important quotas and corresponding amount per unit that is charged when free resources are depleted and additional, billable quota is desired.

Features :

With a Runtime Environment, the Data store and the App Engine services, the GAE can be divided into three parts.

Runtime Environment

The GAE runtime environment presents itself as the place where the actual application is executed. However, the application is only invoked once an HTTP request is processed to the GAE via a web browser or some other interface, meaning that the application is not constantly running if no invocation or processing has been done. In case of such an HTTP request, the request handler forwards the request and the GAE selects one out of many possible Google servers where the application is then instantly deployed and executed for a certain amount of time (8). The application may then do some computing and return the result back to the GAE request handler which forwards an HTTP response to the client. It is important to understand that the application runs completely embedded in this described sandbox environment but only as long as requests are still coming in or some processing is done within the application. The reason for this is simple: Applications should only run when they are actually computing, otherwise they would allocate precious computing power and memory without need. This paradigm shows already the GAE‘s potential in terms of scalability. Being able to run multiple instances of one application independently on different servers guarantees for a decent level of scalability. However, this highly flexible and stateless application execution paradigm has its limitations. Requests

are processed no longer than 30 seconds after which the response has to be returned to the client and the application is removed from the runtime environment again (8). Obviously this method

accepts that for deploying and starting an application each time a request is processed, an additional lead time is needed until the application is finally up and running. The GAE tries to encounter this problem by caching the application in the server memory as long as possible, optimizing for several subsequent requests to the same application. The type of runtime environment on the Google servers is dependent on the programming language used. For Java or other languages that have support for Java-based compilers (such as JRuby, Rhino and Groovy) a Java-based Java Virtual Machine (JVM) is provided. Also, GAE fully supports the Google Web Toolkit (GWT), a framework for rich web applications. For Python and related frameworks a Python-based environment is used.

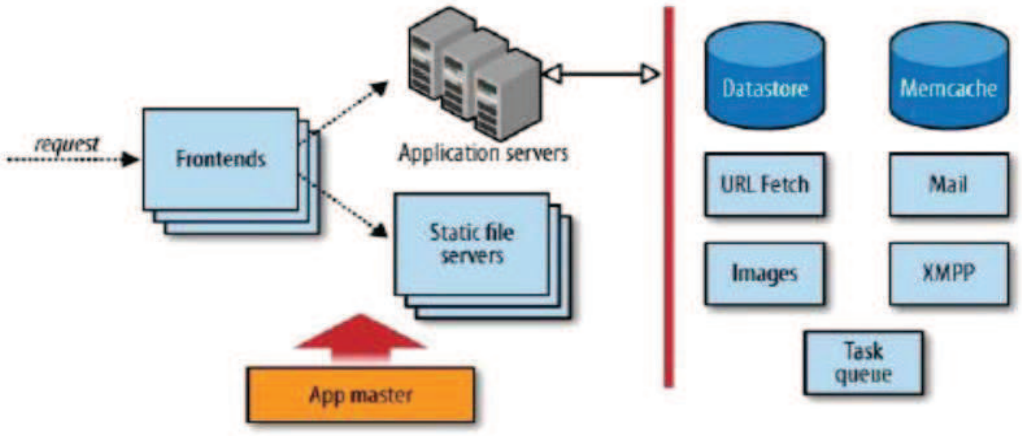


FIGURE 4: STRUCTURE OF GOOGLE APP ENGINE (13)

Persistence and the datastore

As previously discussed, the stateless execution of applications creates the need for a datastore that provides a proper way for persistence. Traditionally, the most popular way of persisting data in web applications has been the use of relational databases. However, setting the focus on high flexibility and scalability, the GAE uses a different approach for data persistence, called *Bigtable* (14). Instead of rows found in a relational database, in Google‘s *Bigtable* data is stored in *entities*. Entities are always associated with a certain *kind.* These entities have *properties*, resembling columns in relational database schemes. But in contrast to relational databases, entities are actually schemaless, as two entities of the same kind not necessarily have to have the same properties or even the same type of value for a certain property.

> The most important difference to relational databases is however the querying of entities withina Bigtable datastore. In relational databases queries are processed and executed against a database at application runtime. GAE uses a different approach here. Instead of processing a query at application runtime, queries are pre-processed during compilation time when a corresponding index is created. This index is later used at application runtime when the actual query is executed. Thanks to the index, each query is only a simple table scan where only the exact filter value is searched. This method makes queries very fast compared to relational databases while updating entities is a lot more expensive.

Transactions are similar to those in relational databases. Each transaction is atomic, meaning that it either fully succeeds or fails. As described above, one of the advantages of the GAE is its scalability through concurrent instances of the same application. But what happens when two instances try to start transactions trying to alter the same entity? The answer to this is quite simple: Only the first instance gets access to the entity and keeps it until the transaction is completed or eventually failed. In this case the second instance will receive a concurrency failure exception. The GAE uses a method of handling such parallel transactions called optimistic concurrency control. It simply denies more than one altering transaction on an entity and implicates that an application running within the GAE should have a mechanism trying to get write access to an entity multiple times before finally giving up.

Heavily relying on indexes and optimistic concurrency control, the GAE allows performing queries very fast even at higher scales while assuring data consistency.

Services

As mentioned earlier, the GAE serves as an abstraction of the underlying hardware and operating system layers. These abstractions are implemented as services that can be directly called from the actual application. In fact, the datastore itself is as well a service that is controlled by the runtime environment of the application.

MEM CACHE

The platform innate memory cache service serves as a short-term storage. As its name suggests, it stores data in a server‘s memory allowing for faster access compared to the datastore. Memcache is a non-persistent data store that should only be used to store temporary data within a series of computations. Probably the most common use case for Memcache is to store session specific data (15). Persisting session information in the datastore and executing queries on every page interaction is highly inefficient over the application lifetime, since session-owner instances are unique per session (16). Moreover, Memcache is well suited to speed up common datastore queries (8). To interact with the Memcache

GAE supports JCache, a proposed interface standard for memory caches (17).

URL FETCH

Because the GAE restrictions do not allow opening sockets (18), a URL Fetch service can be used to send HTTP or HTTPS requests to other servers on the Internet. This service works asynchronously, giving the remote server some time to respond while the request handler can do

other things in the meantime. After the server has answered, the URL Fetch service returns response code as well as header and body. Using the Google Secure Data Connector an application can even access servers behind a company‘s firewall (8).

MAIL

The GAE also offers a mail service that allows sending and receiving email messages. Mails can be sent out directly from the application either on behalf of the application‘s administrator or on behalf of userswith Google Accounts. Moreover, an application can receive emails in the form of HTTP requests initiated by the App Engine and posted to the app at multiple addresses. In contrast to incoming emails, outgoing messages may also have an attachment up to 1 MB (8).

XMPP

In analogy to the mail service a similar service exists for instant messaging, allowing an application to send and receive instant messages when deployed to the GAE. The service allows communication to and from any instant messaging service compatible to XMPP (8), a set of open technologies for instant messaging and related tasks (19).

IMAGES

Google also integrated a dedicated image manipulation service into the App Engine. Using this service images can be resized, rotated, flipped or cropped (18). Additionally it is able to combine several images into a single one, convert between several image formats and enhance photographs. Of course the API also provides information about format, dimensions and a histogram of color values (8).

USERS

User authentication with GAE comes in two flavors. Developers can roll their own authentication service using custom classes, tables and Memcache or simply plug into Google‘s Accounts service.

Since for most applications the time and effort of creating a sign-up page and store user passwords is

not worth the trouble (18), the User service is a very convenient functionality which gives an easy method for authenticating users within applications. As byproduct thousands of Google Accounts are leveraged. The User service detects if a user has signed in and otherwise redirect the user to a sign-in page. Furthermore, it can detect whether the current user is an administrator, which facilitates implementing admin-only areas within the application (8).

OAUTH

The general idea behind OAuth is to allow a user to grant a third party limited permission to access protected data without sharing username and password with the third party. The OAuth specification separates between a consumer, which is the application that seeks permission on accessing protected data, and the service provider who is storing protected data on his users' behalf (20). Using Google Accounts and the GAE API, applications can be an OAuth service provider (8).

SCHEDULED TASKS AND TASK QUEUES

Because background processing is restricted on the GAE platform, Google introduced task queues as another built-in functionality (18). When a client requests an application to do certain steps, the application might not be able to process them right away. This is where the task queues come into play. Requests that cannot be executed right away are saved in a task queue that controls the correct sequence of execution. This way, the client gets a response to its request right away, possibly with the indication that the request will be executed later (13). Similar to the concept of task queues are corn jobs. Borrowed from the UNIX world, a GAE cron job is a scheduled job that can invoke a request handler at a pre-specified time (8).

BLOBSTORE

The general idea behind the blobstore is to allow applications to handle objects that are much larger than the size allowed for objects in the datastore service. Blob is short for binary large object and is designed to serve large files, such as video or high quality images. Although blobs can have up to 2 GB they have to be processed in portions, one MB at a time. This restriction was introduced to smooth the curve of datastore traffic. To enable queries for blobs, each has a corresponding blob info record which is persisted in the datastore (8), e. g. for creating an image database.

ADMINISTRATION CONSOLE

The administration console acts as a management cockpit for GAE applications. It gives the developer real-time data and information about the current performance of the deployed application and is used to upload new versions of the source code. At this juncture it is possible to test new versions of the

application and switch the versions presented to the user. Furthermore, access data and logfiles can be viewed. It also enables analysis of traffic so that quota can be adapted when needed. Also

the status of scheduled tasks can be checked and the administrator is able to browse the applications datastore and manage indices (8).

App Engine for Business

While the GAE is more targeted towards independent developers in need for a hosting platform for their medium-sized applications, Google`s recently launched App Engine for Business tries to target the corporate market. Although technically mostly relying on the described GAE, Google added some enterprise features and a new pricing scheme to make their cloud computing platform more attractive for enterprise customers (21). Regarding the features, App Engine for Business includes a central development manager that allows a central administration of all applications deployed within one company including access control lists. In addition to that Google now offers a 99.9% service level agreement as well as premium developer support. Google also adjusted the pricing scheme for their corporate customers by offering a fixed price of $8 per user per application, up to a maximum of $1000, per month. Interestingly, unlike the pricing scheme for the GAE, this offer includes unlimited processing power for a fixed price of $8 per user, application and month. From a technical point of view, Google tries to accommodate for established industry standards, by now offering SQL database support in addition to the existing Bigtable datastore described above (8).

APPLICATION DEVELOPMENT USING GOOGLE APP ENGINE

General Idea

In order to evaluate the flexibility and scalability of the GAE we tried to come up with an application that relies heavily on scalability, i.e. collects large amounts of data from external sources. That way we hoped to be able to test both persistency and the gathering of data from external sources at large scale. Therefore our idea has been to develop an application that connects people`s delicious bookmarks with their respective Facebook accounts. People using our application should be able to see what their

Facebook friends‘ delicious bookmarks are, provided their Facebook friends have such a delicious account. This way a user can get a visualization of his friends‘ latest topics by looking at a generated tag cloud giving him a clue about the most common and shared interests.

PLATFORM AS A SERVICE: GOOGLE APP ENGINE:--

The Google cloud, called Google App Engine, is a _platform as a service‘ (PaaS) offering. In contrast with the Amazon infrastructure as a service cloud, where users explicitly provision virtual machines and control them fully, including installing, compiling and running software on

them, a PaaS offering hides the actual execution environment from users. Instead, a software platform is provided along with an SDK, using which users develop applications and deploy them on the cloud. The PaaS platform is responsible for executing the applications, including servicing external service requests, as well as running scheduled jobs included in the application. By making the actual execution servers transparent to the user, a PaaS platform is able to share *application* servers across users who need lower capacities, as well as automatically scale resources allocated to applications that experience heavy loads. Figure 5.2 depicts a user view of Google App Engine. Users upload code, in either Java or Python, along with related files, which are stored on the Google File System, a very large scale fault tolerant and redundant storage system. It is important to note that an application is immediately available on the internet as soon as it is successfully uploaded (no virtual servers need to be explicitly provisioned as in IaaS).

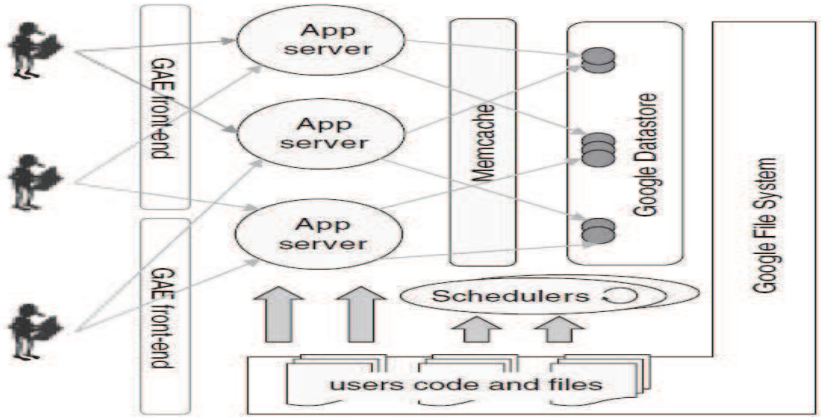


FIGURE 5.2. Google App Engine

Resource usage for an application is metered in terms of web requests served and CPU-hours actually spent executing requests or batch jobs. Note that this is very different from the IaaS model: A PaaS application can be deployed and made globally available 24×7, but charged only when *accessed*

(or if batch jobs run); in contrast, in an IaaS model merely making an application continuously available incurs the full cost of keeping at least some of the servers running all the time. Further, deploying applications in Google App Engine is free, within usage limits; thus applications can be developed and tried out free and begin to incur cost only when actually accessed by a sufficient volume of requests. The PaaS model enables Google to provide such a free service because applications do not run in

dedicated virtual machines; a deployed application that is not accessed merely consumes storage for its code and data and expends no CPU cycles.

GAE applications are served by a large number of web servers in Google‘s data centers that execute requests from end-users across the globe. The web servers load code from the GFS into memory and serve these requests. Each request to a particular application is served by any one of GAE‘s web servers; there is no guarantee that the same server will serve requests to any two requests, even from the same HTTP session. Applications can also specify some functions to be executed as batch jobs which are run by a scheduler.

Google Datastore:--

Applications persist data in the Google Datastore, which is also (like Amazon SimpleDB) a non-relational database. The Datastore allows applications to define structured types (called _kinds‘) and store their instances (called _entities‘) in a distributed manner on the GFS file system. While one can view Datastore _kinds‘ as table structures and entities as records, there are important differences between a relational model and the Datastore, some of which are also illustrated in Figure 5.3.

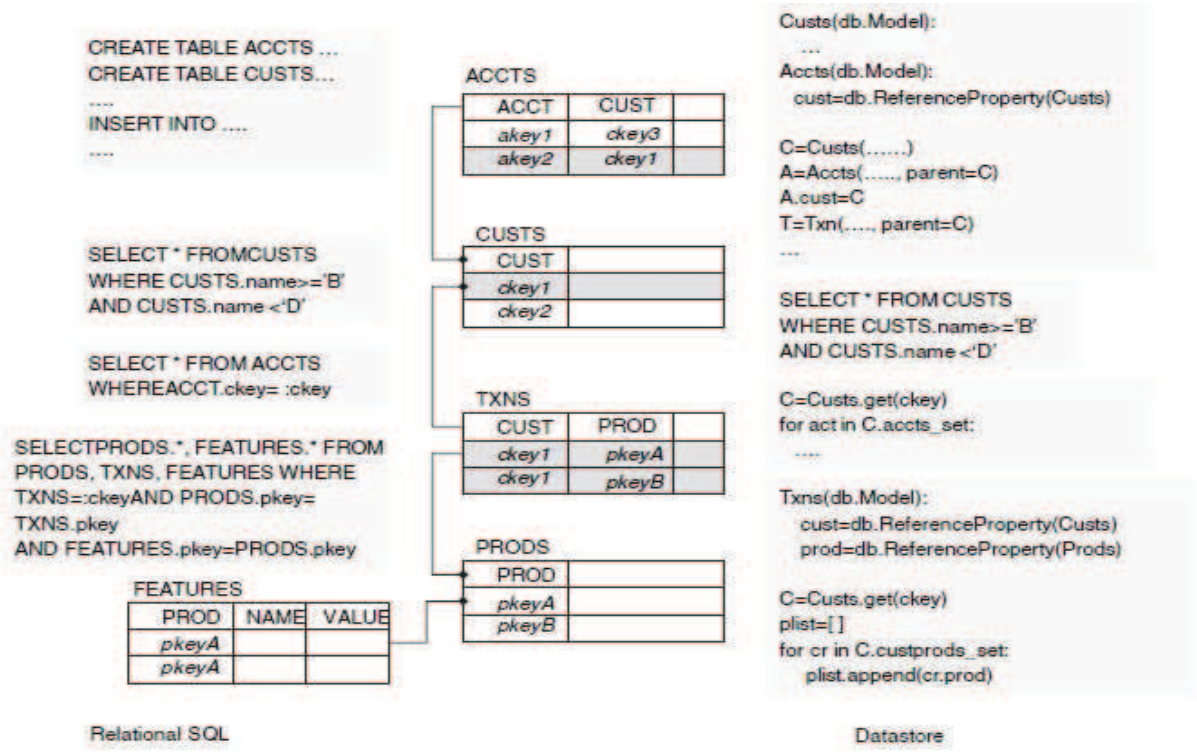


FIGURE 5.3. Google Datastore

Unlike a relational schema where all rows in a table have the same set of columns, all entities of a _kind‘ need not have the same properties. Instead, additional properties can be added to any entity. This feature is particularly useful in situations where one cannot foresee all the potential properties in

a model, especially those that occur occasionally for only a small subset of records. For example, a model storing

_products‘ of different types (shows, books, etc.) would need to allow each product to have a different set of features. In a relational model, this would probably be implemented using a separate FEATURES table, as shown on the bottom left of Figure 5.3. Using the Datastore, this table (_kind‘) is not required; instead, each product entity can be assigned a different set of properties at runtime. The Datastore allows simple queries with conditions, such as the first query shown in Figure 5.3 to retrieve all customers having names in some lexicographic range. The query syntax (called GQL) is essentially the same as SQL, but with some restrictions. For example, all inequality conditions in a querymust be on a single property; so a query that also filtered customers on, say, their _type‘, would be illegal in GQL but allowed in SQL.

Relationships between tables in a relational model are modeled using foreign keys. Thus, each account in the ACCTS table has a pointer *ckey* to the customer in the CUSTS table that it belongs to. Relationships are traversed via queries using foreign keys, such as retrieving all accounts for a particular customer, as shown. The Datastore provides a more object-oriented approach to relationships in persistent data. Model definitions can include references to other models; thus each entity of the Accts

_kind‘ includes a reference to its customer, which is an entity of the Custs _kind.‘ Further, relationships defined by such references can be traversed in *both* directions, so not only can one directly access the customer of an account, but also *all* accounts of a given customer, without executing any query operation, as shown in the figure.

GQL queries *cannot* execute joins between models. Joins are critical when using SQL to efficiently retrieve data from multiple tables. For example, the query shown in the figure retrieves details of all products bought by a particular customer, for which it needs to join data from the transactions (TXNS), products (PRODS) and product features (FEATURES) tables. Even though GQL does not allow joins, its ability to traverse associations between entities often enables joins to be avoided, as shown in the figure for the above example: By storing references to customers and products in the Txns model, it is possible to retrieve all transactions for a given customer through a reverse traversal of the customer reference. The product references in each transaction then yield all products and their features (as discussed earlier, a separate Features model is not required because of schema

flexibility). It is important to note that while object relationship traversal can be used as an alternative to joins, this is not always possible, and when required joins may need to be explicitly executed by application code.

The Google Datastore is a distributed object store where objects (entities) of all GAE applications are maintained using a large number of servers and the GFS distributed file system. From a user perspective, it is important to ensure that in spite of sharing a distributed storage scheme with many other users, application data is (a) retrieved efficiently and (b) atomically updated. The Datastore provides a mechanism to group entities from different _kinds‘ in a hierarchy that is used for both these purposes. Notice that in Figure 5.3entities of the Accts and Txns _kinds‘ are instantiated with a parameter _parent‘ that specifies a particular customer entity, thereby linking these three entities in an _entity group‘. The Datastore ensures that all entities belonging to a particular group are stored close together in the distributed file system (we shall see how in Chapter 10). The Datastore allows processing steps to be grouped into transactions wherein updates to data are guaranteed to be

atomic; however this also requires that each transaction only manipulates entities belonging to the same entity group. While this transaction model suffices for most on line applications, complex batch updates that update many unrelated entities cannot execute atomically, unlike in a relational database where there are no such restrictions.

Amazon SimpleDB:--

Amazon SimpleDB is also a nonrelational database, in many ways similar to the Google Datastore.

SimpleDB_domains‘ correspond to _kinds‘, and _items‘ to entities; each item can have a number of attribute-value pairs, and different items in a domain can have different sets of attributes, similar to Datastore entities. Queries on SimpleDB domains can include conditions, including inequality conditions, on any number of attributes. Further, just as in the Google Datastore, joins are not permitted. However, SimpleDB does not support object relationships as in Google Datastore, nor does it support transactions. It is important to note that all data in SimpleDB is replicated for redundancy, just as in

GFS. Because of replication, SimpleDB features an _eventual consistency‘ model, wherein data is guaranteed to be propagated to at least one replica and will eventually reach all replicas, albeit with some delay. This can result in perceived inconsistency, since an immediate read following a write may not always yield the result written. In the case of Google Datastore on the other hand, writes succeed only when all replicas are updated; this avoids inconsistency but also makes writes slower.

PAAS CASE STUDY: FACEBOOK

Facebook provides some PaaS capabilities to application developers:--

- Web services remote APIs that allow access to social network properties, data,Like button, etc.
- Many third-parties run their apps off Amazon EC2, and interface to Facebook via its APIs PaaS
    - IaaS
- Facebook itself makes heavy use of PaaS services for their own private cloud
- Key problems: how to analyze logs, make suggestions, determine which ads to place.

Facebook API: Overview:--

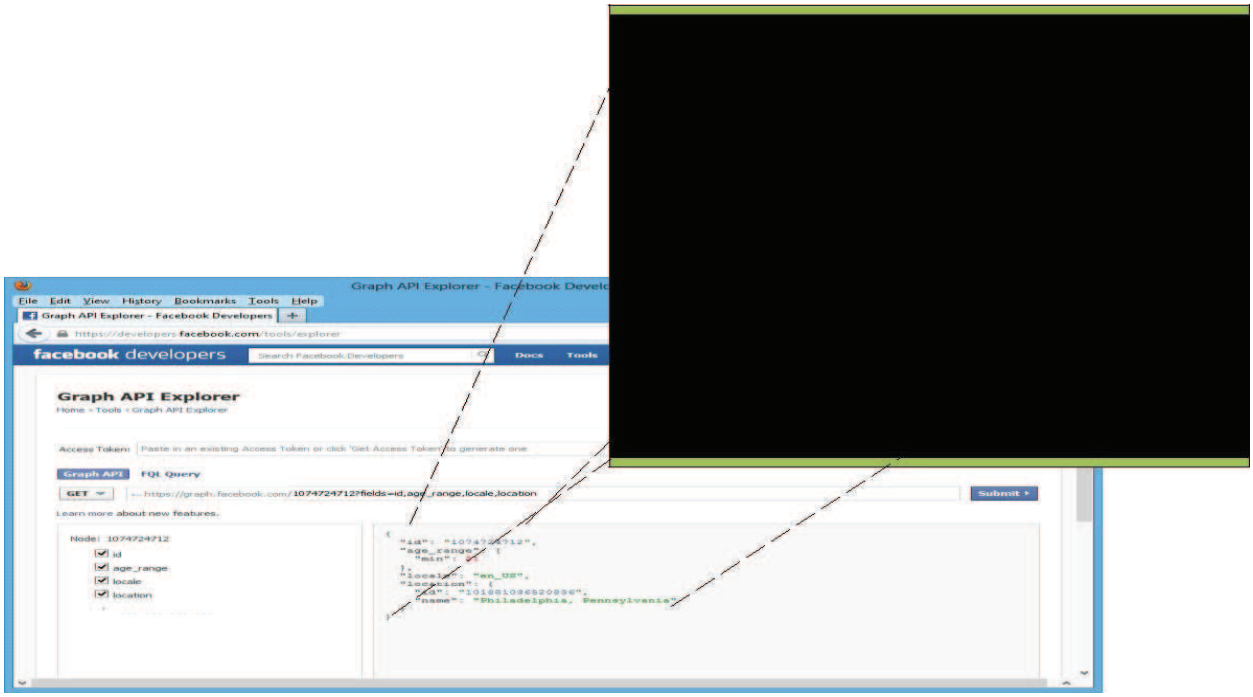
What you can do:

- Read data from profiles and pages
- Navigate the graph (e.g., via friends lists)
- Issue queries (for posts, people, pages, ...)

Facebook API: The Graph API :

```
{
  "id":
  "1074724712",
  "age_range": {
    "min": 21
  },
  "locale":
  "en_US",
  "location": {
    "id": "101881036520836",
    "name": "Philadelphia,
Pennsylvania"
  }
```

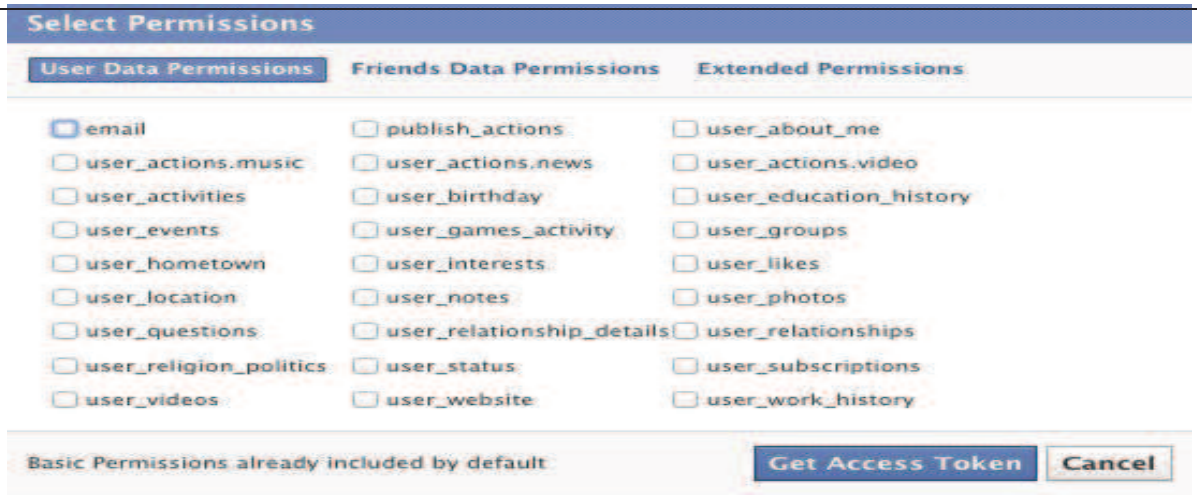
}



- Requests are mapped directly to HTTP:
    - https://graph.facebook.com/(identifier)?fields=(fieldList)
- Response is in JSON

Uses several HTTP methods:

- GET for reading
- POST for adding or modifying
- DELETE for removing
- IDs can be numeric or names
- /1074724712 or /andreas.haeberlen
- Pages also have IDs
- Authorization is via 'access tokens'
- Opaque string; encodes specific permissions (access user location, but not interests, etc.)
- Has an expiration date, so may need to be refreshed



Facebook Data Management / Warehousing Tasks

Main tasks for "cloud" infrastructure:

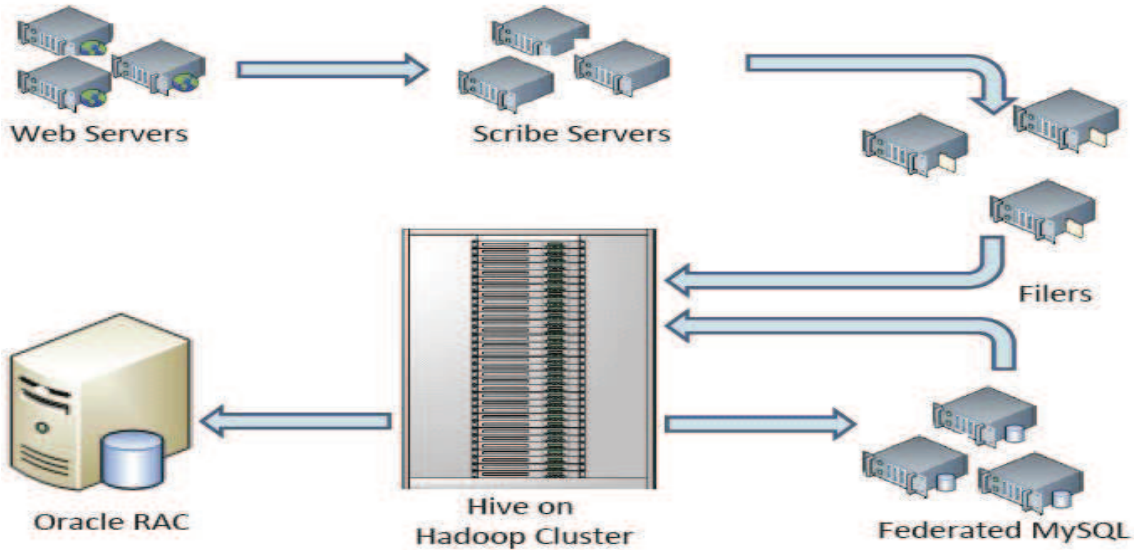
- Summarization (daily, hourly)
    - to help guide development on different components
    - to report on ad performance
    - recommendations

Ad hoc analysis:

- Answer questions on historical data – to help with managerial decisions
- Archival of logs
- Spam detection
- Ad optimization

- Initially used Oracle DBMS for this
    - But eventually hit scalability, cost, performance bottlenecks just like Salesforce does now

Data Warehousing at Facebook:



PAAS AT FACEBOOK:

- Scribe – open source logging, actually records the data that will be analyzed by Hadoop
- Hadoop (MapReduce – discussed next time) as batch processing engine for data analysis
    - As of 2009: 2nd largest Hadoop cluster in the world, 2400 cores, > 2PB data with > 10TB added every day
- Hive – SQL over Hadoop, used to write the data analysis queries
- Federated MySQL, Oracle – multi-machine DBMSs to store query results

Example Use Case 1: Ad Details

- Advertisers need to see how their ads are performing
    - Cost-per-click (CPC), cost-per-1000-impressions (CPM)
    - Social ads – include info from friends
    - Engagement ads – interactive with video
- Performance numbers given:
    - Number unique users, clicks, video views, …
- Main axes:

- Account, campaign, ad
- Time period
- Type of interaction
- Users

- Summaries are computed using Hadoop via Hive

Use Case 2: Ad Hoc analysis, feedback

- Engineers, product managers may need to understand what is going on
  - e.g., impact of a new change on some sub-population
- Again, Hive-based, i.e., queries are in SQL with database joins
  - Combine data from several tables, e.g., click-through rate = views combined with clicks
- Sometimes requires custom analysis code with sampling

CONCLUSION :

Cloud Computing remains the number one hype topic within the IT industry at present. Our evaluation of the Google App Engine and facebook has shown both functionality and limitations of the platform. Developing and deploying an application within the GAE is in fact quite easy and in a way shows the progress that software development and deployment has made. Within our application we were able to use the abstractions provided by the GAE without problems, although the concept of Bigtable requires a big change in mindset when developing. Our scalability testing showed the limitations of the GAE at this point in time. Although being an extremely helpful feature and a great USP for the GAE, the built-in scalability of the GAE suffers from both purposely-set as well as technical restrictions at the moment. Coming back to our motivation of evaluating the GAE in terms of its sufficiency for serious large-scale applications in a professional environment, we have to conclude that the GAE not (yet) fulfills business needs for enterprise applications at present.

*****