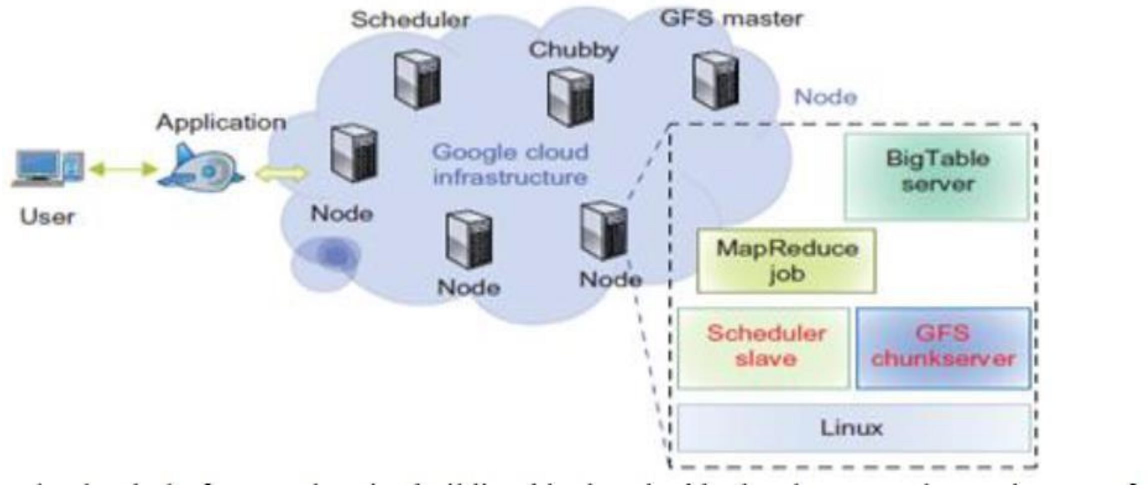
GOOGLE APPLICATION ENGINE (GAE)

- Google App Engine is a PaaS cloud that provides a complete Web service environment(Platform)
- GAE provides Web application development platform for users.
- All required hardware, operating systems and software are provided to clients.
- Clients can develop their own applications, while App Engine runs the applications on Google's servers.
- GAE helps to easily develop an Web Application
- App Engine only supports the Java and Python programming languages.
- The Google App Engine (GAE) provides a powerful distributed data storage service.

GOOGLE CLOUD INFRASTRUCTURE

- Google has established cloud development by making use of large number of data centers.
- Eg: Google established cloud services in
    - ❖ Gmail
    - ❖ Google Docs
    - ❖ Google Earth etc.
- These applications can support a large number of users simultaneously with High Availability (HA).
- In 2008, Google announced the GAE web application platform.
- GAE enables users to run their applications on a large number of data centers.
- Google App Engine environment includes the following features :
    - ❖ Dynamic web serving
    - ❖ Persistent(constant) storage with queries, sorting, and transactions
    - ❖ Automatic scaling and load balancing
- Provides Application Programming Interface(API) for authenticating users.
- Send email using Google Accounts.
- Local development environment that simulates(create) Google App Engine on your computer.
-

GAE ARCHITECTURE



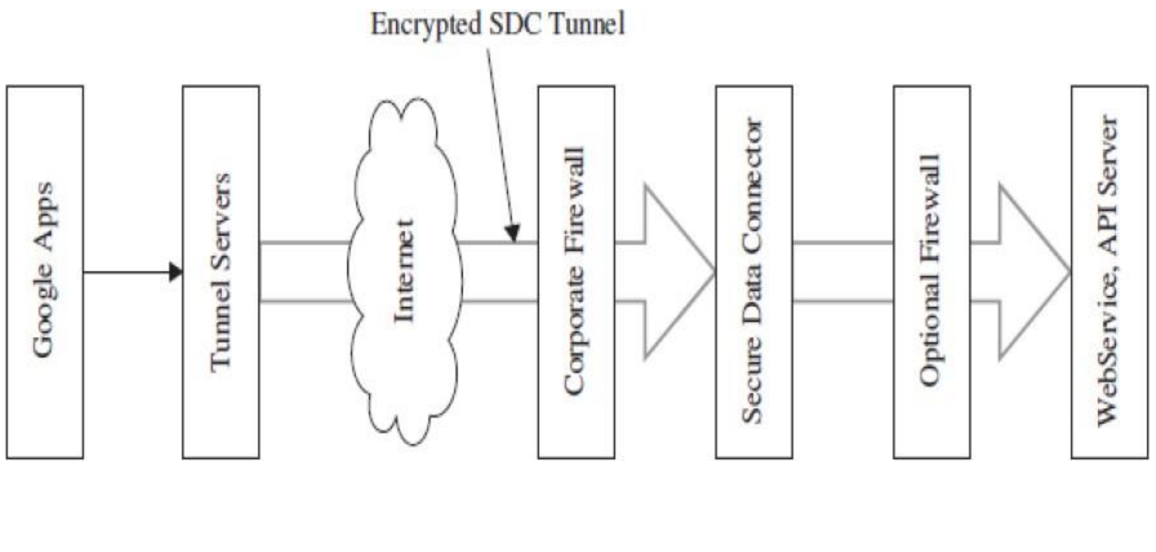
TECHNOLOGIES USED BY GOOGLE ARE

- Google File System(GFS) ->for storing large amounts of data.
- MapReduce->for application program development.
- Chubby-> for distributed application lock services.
- BigTable-> offers a storage service.
- Third-party application providers can use GAE to build cloud applications for providing services.
- Inside each data center, there are thousands of servers forming different clusters.
- GAE runs the user program on Google's infrastructure.
- Application developers now do not need to worry about the maintenance of servers.
- GAE can be thought of as the combination of several software components.
- GAE supports Python and Java programming environments.

FUNCTIONAL MODULES OF GAE

- The GAE platform comprises the following five major components.
- DataStore: offers data storage services based on BigTable techniques.
- The Google App Engine (GAE) provides a powerful distributed data storage service.
- This provides a secure data Storage.

GOOGLE SECURE DATA CONNECTOR (SDC)



FUNCTIONAL MODULES OF GAE

- When the user wants to get the data, he/she will first send an authorized data requests to Google Apps.
- It forwards the request to the tunnel server.
- The tunnel servers validate the request identity.
- If the identity is valid, the tunnel protocol allows the SDC to set up a connection, authenticate, and encrypt the data that flows across the Internet.
- SDC also validates whether a user is authorized to access a specified resource.
- Application runtime environment offers a platform for web programming and execution.
- It supports two development languages: Python and Java.
- Software Development Kit (SDK) is used for local application development.
- The SDK allows users to execute test runs of local applications and upload application code.
- Administration console is used for easy management of user application development cycles.
- GAE web service infrastructure provides special guarantee flexible use and management of storage and network resources by GAE.
- Google offers essentially free GAE services to all Gmail account owners.

- We can register for a GAE account or use your Gmail account name to sign up for the service.
- The service is free within a quota.
- If you exceed the quota, extra amount will be charged.
- Allows the user to deploy user-built applications on top of the cloud infrastructure.
- They are built using the programming languages and software tools supported by the provider (e.g., Java, Python)

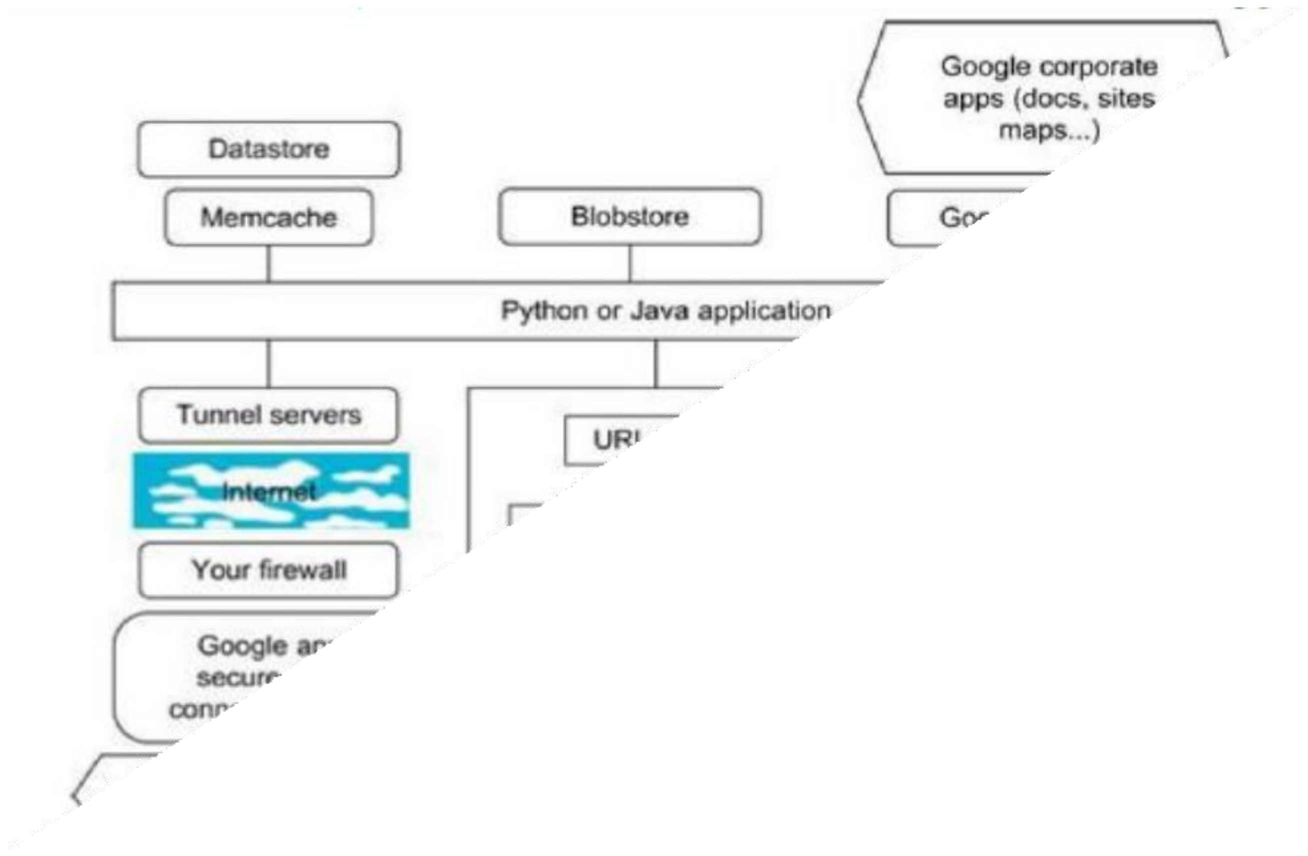
GAE APPLICATIONS

Well-known GAE applications

- Google Search Engine
- Google Docs
- Google Earth
- Gmail
- These applications can support large numbers of users simultaneously.
- Users can interact with Google applications via the web interface provided by each application.
- Applications run in the Google data centers.
- Inside each data center, there might be thousands of server nodes to form different clusters.
- Each cluster can run multipurpose servers.

5. 3.1 Programming Support of Google App Engine

GAE programming model for two supported languages: Java and Python. A client environment includes an Eclipse plug-in for Java allows you to debug your GAE on your local machine. Google Web Toolkit is available for Java web application developers. Python is used with frameworks such as Django and CherryPy, but Google also has webapp Python environment.



There are several powerful constructs for storing and accessing data. The data store is a NOSQL data management system for entities. Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the Data Nucleus Access platform, while Python has a SQL-like query language called GQL. The performance of the data store can be enhanced by in-memory caching using the memcache, which can also be used independently of the data store.

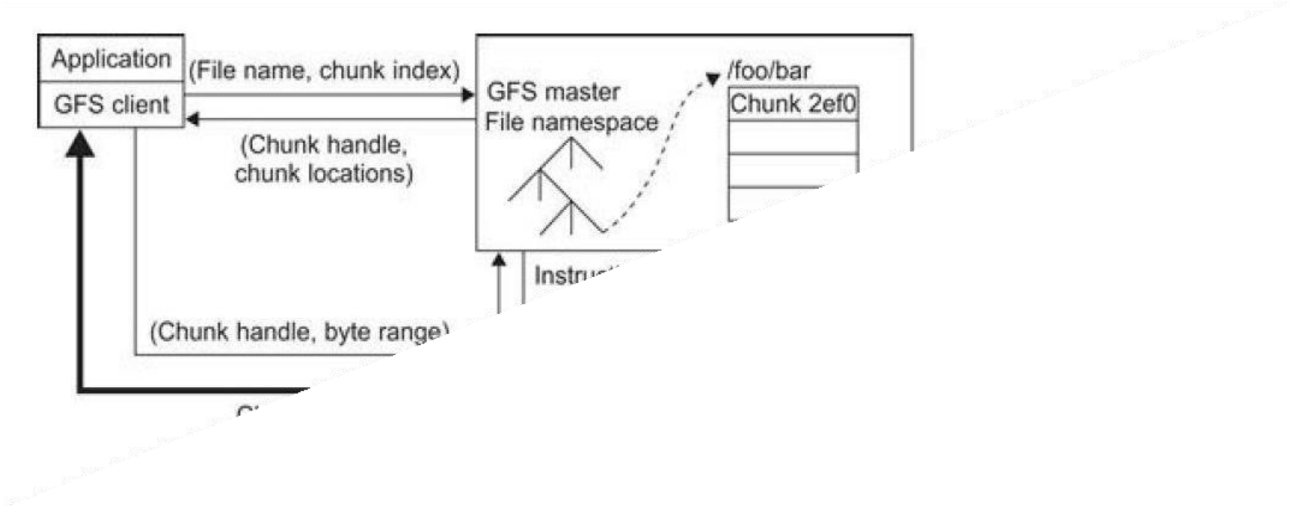
Recently, Google added the blobstore which is suitable for large files as its size limit is 2 GB. There are several mechanisms for incorporating external resources. The Google SDC Secure Data Connection can tunnel through the Internet and link your intranet to an external GAE application. The URL Fetch operation provides the ability for applications to fetch resources and communicate with other hosts over the Internet using HTTP and HTTPS requests.

An application can use Google Accounts for user authentication. Google Accounts handles user account creation and sign-in, and a user that already has a Google account (such as a Gmail account) can use that account with your app. GAE provides the ability to manipulate image data using a dedicated Images service which can resize, rotate, flip, crop, and enhance images. A GAE application is configured to consume resources up to certain limits or quotas. With quotas, GAE ensures that your application won't exceed your budget, and that other applications running on GAE won't impact the performance of your app. In particular, GAE use is free up to certain quotas.

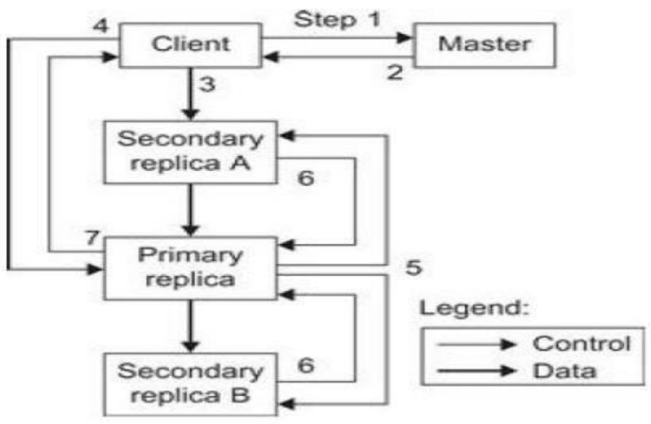
Google File System (GFS)

GFS is a fundamental storage service for Google's search engine. GFS was designed for Google applications, and Google applications were built for GFS. There are several concerns in GFS. rate). As servers are composed of inexpensive commodity components, it is the norm rather than the exception that concurrent failures will occur all the time. Other concerns the file size in GFS. GFS typically will hold a large number of huge files, each 100 MB or larger, with files that are multiple GB in size quite common. Thus, Google has chosen its file data block size to be 64 MB instead of the 4 KB in typical traditional file systems. The I/O pattern in the Google application is also special. Files are typically written once, and the write operations are often the appending data blocks to the end of files. Multiple appending operations might be concurrent. The customized API can simplify the problem and focus on Google applications.

Figure shows the GFS architecture. It is quite obvious that there is a single master in the whole cluster. Other nodes act as the chunk servers for storing data, while the single master stores the metadata. The file system namespace and locking facilities are managed by the master. The master periodically communicates with the chunk servers to collect management information as well as give instructions to the chunk servers to do work such as load balancing or fail recovery.



The master has enough information to keep the whole cluster in a healthy state. Google uses a shadow master to replicate all the data on the master, and the design guarantees that all the data operations are performed directly between the client and the chunk server. The controlmessages are transferred between the master and the clients and they can be cached for future use. With the current quality of commodity servers, the single master can handle a cluster of more than 1,000 nodes.



The mutation takes the following steps:

1. The client asks the master which chunk server holds the current lease for the chunk and the locations of the other replicas. If no one has a lease, the master grants one to a replica it chooses (not shown).

2. The master replies with the identity of the primary and the locations of the other (secondary) replicas. The client caches this data for future mutations. It needs to contact the master again only when the primary becomes unreachable or replies that it no longer holds a lease.
3. The client pushes the data to all the replicas. Each chunk server will store the data in an internal LRU buffer cache until the data is used or aged out. By decoupling the data flow from the control flow, we can improve performance by scheduling the expensive data flow based on the network topology regardless of which chunk server is the primary.
4. Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary. The request identifies the data pushed earlier to all the replicas. The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients, which provides the necessary serialization. It applies the mutation to its own local state in serial order.
5. The primary forwards the write request to all secondary replicas. Each secondary replica applies mutations in the same serial number order assigned by the primary.
6. The secondaries all reply to the primary indicating that they have completed the operation.
7. The primary replies to the client. Any errors encountered at any replicas are reported to the client. In case of errors, the write corrects at the primary and an arbitrary subset of the secondary replicas. The client request is considered to have failed, and the modified region is left in an inconsistent state. Our client code handles such errors by retrying the failed mutation

Big Table

BigTable was designed to provide a service for storing and retrieving structured and semistructured data. BigTable applications include storage of web pages, per-user data, and geographic locations. The database needs to support very high read/write rates and the scale might be millions of operations per second. Also, the database needs to support efficient scans over all or interesting subsets of data, as well as efficient joins of large one-to-one and one-to-many data sets. The application may need to examine data changes over time.

The BigTable system is scalable, which means the system has thousands of servers, terabytes of in-memory data, petabytes of disk-based data, millions of reads/writes per second, andefficient scans. BigTable is used in many projects, including Google Search, Orkut, and Google Maps/Google Earth, among others.

The BigTable system is built on top of an existing Google cloud infrastructure. BigTable uses the following building blocks:

1. GFS: stores persistent state
2. Scheduler: schedules jobs involved in BigTable serving
3. Lock service: master election, location bootstrapping
4. MapReduce: often used to read/write BigTable data.