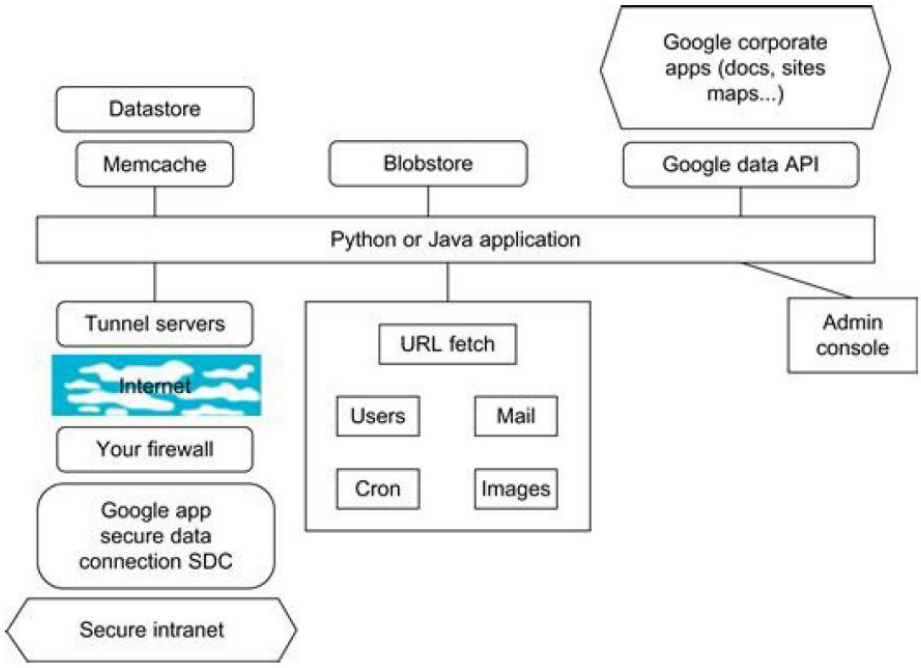
Programming Support of Google App Engine

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.

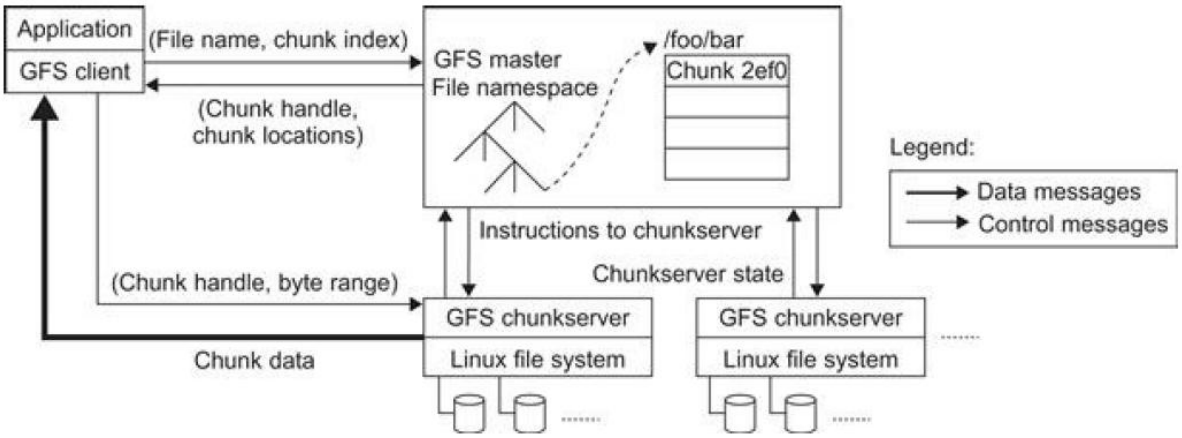
Recently, Google added the blobstore which is suitable for large files as its size limit is 2 GB. There are several mechanisms for incorporating external resources. The Google SDC Secure Data Connection can tunnel through the Internet and link your intranet to an external GAE application. The URL Fetch 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)

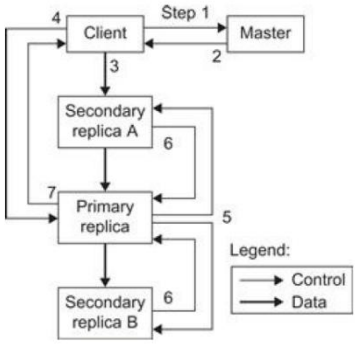
GFS is a fundamental storage service for Google's search engine. GFS was designed for Google applications, and Google applications were built for GFS. There are several concerns in GFS. rate). As servers are composed of inexpensive commodity components, it is the norm rather than the exception that concurrent failures will occur all the time. Another 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 control messages are transferred between the master and the clients and they can be cached for future use. With the current qualityof commodity servers, the single master can handle a cluster of more than 1,000 nodes.

Figure shows the data mutation (write, append operations) in GFS. Data blocks must be created for all replicas.



The goal is to minimize involvement of the master. 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. It will make a few attempts at steps 3 through 7 before falling back to a retry from the beginning of the write.

GFS was designed for high fault tolerance and adopted some methods to achieve this goal. Master and chunk servers can be restarted in a few seconds, and with such a fast recovery capability, the window of time in which the data is unavailable can be greatly reduced. As we mentioned before, each chunk is replicated in at least three places and can tolerate at least two data crashes for a single chunk of data. The shadow master handles the failure of the GFS master

Big Table

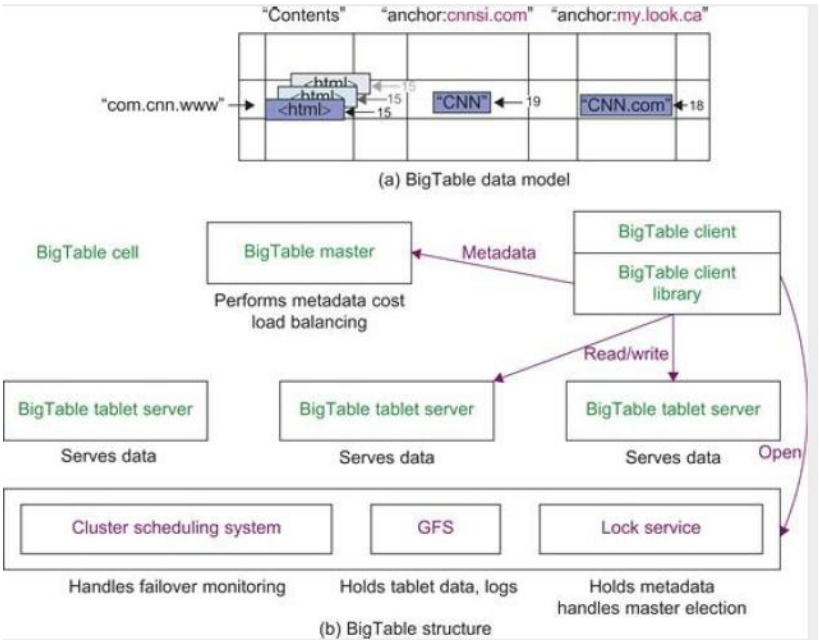
BigTable was designed to provide a service for storing and retrieving structured and semistructured data. BigTable applications include storage of web pages, per-user data, and geographic locations. The database needs to support very high read/write rates and the scale might be millions of operations 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, and efficient 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

BigTable provides a simplified data model called Web Table, compared to traditional database systems. Figure (a) shows the data model of a sample table. Web Table stores the data about a web page. Each web page can be accessed by the URL. The URL is considered the row index. The column provides different data related to the corresponding URL



The map is indexed by row key, column key, and timestamp−that is, (row:string, column: string, time:int64) maps to string (cell contents). Rows are ordered in lexicographic order by row key. The row range for a table is dynamically partitioned and each row range is called “Tablet”.

Syntax for columns is shown as a (family:qualifier) pair. Cells can store multiple versions of data with timestamps.

Figure (b) shows the BigTable system structure. A BigTable master manages and stores the metadata of the BigTable system. BigTable clients use the BigTable client programming library to communicate with the BigTable master and tablet servers. BigTable relies on a highly available and persistent distributed lock service called Chubby.

Programming on Amazon AWS

AWS platform has many features and offers many services

Features:

- Relational Database Service (RDS) with a messaging interface
- Elastic MapReduce capability
- NOSQL support in SimpleDB

Capabilities

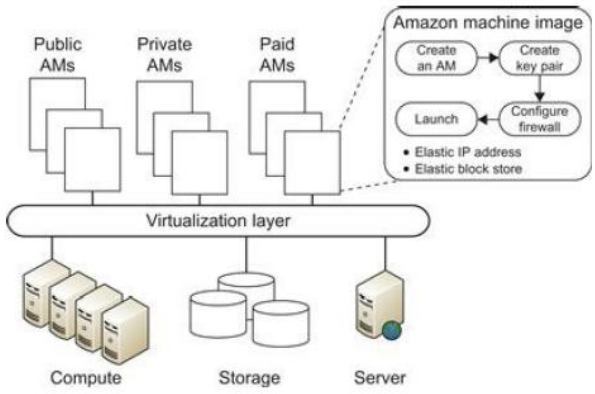
- Auto-scaling enables you to automatically scale your Amazon EC2 capacity up or down according to conditions
- Elastic load balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances
- CloudWatch is a web service that provides monitoring for AWS cloud resources, operational performance, andoverall demand patterns−including metrics such as CPU utilization, disk reads and writes, and network traffic.

A mazon provides several types of preinstalled VMs. Instances are often called Amazon Machine Images (AMIs) which are preconfigured with operating systems based on Linux or Windows, and additional software. Figure 6.24 shows an execution environment.A MIs are the templates for instances, which are running VMs. The AMIs are formed from the virtualized compute, storage, and server resource.

Private AMI: Images created by you, which are private by default. You can grant access to other users to launch your private images.

Public AMI: Images created by users and released to the AWS community, so anyone can launch instances based on them

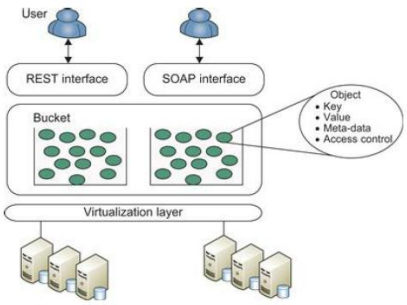
Paid QAMI: You can create images providing specific functions that can be launched by anyone willing to pay you per each hour of usage



Amazon Simple Storage Service (S3)

Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. S3 provides the object-oriented storage service for users. Users can access their objects through Simple Object Access Protocol (SOAP) with either browsers or other client programs which support SOAP. SQS is responsible for ensuring a reliable message service between two processes.

Figure shows the S3 execution environment.



The fundamental operation unit of S3 is called an object. Each object is stored in a bucket and retrieved via a unique, developer-assigned key. In other words, the bucket is the container of the object. Besides unique key attributes, the object has other attributes such as values, metadata, and access control information. Through the key-value programming interface, users can write, read, and delete objects containing from 1 byte to 5 gigabytes of data each. There are two types of web service interface for the user to access the data stored in Amazon clouds. One is a REST (web 2.0) interface, and the other is a SOAP interface. Here are some key features of S3:

- Redundant through geographic dispersion.
- Designed to provide 99.99% durability and 99.99 %availability of objects over a given year with cheaper reduced redundancy storage (RRS).
- Authentication mechanisms to ensure that data is kept secure from unauthorized access. Objects can be made private or public, and rights can be granted to specific users.
- Per-object URLs and ACLs (access control lists).
- Default download protocol of HTTP

Amazon Elastic Block Store (EBS) and SimpleDB

The Elastic Block Store (EBS) provides the volume block interface for saving and restoring the virtual images of EC2 instances. The status of EC2 can now be saved in the EBS system after the machine is shut down.Users can use EBS to save persistent data and mount to the running instances of EC2. S3 is "Storage as a Service" with a messaging interface. Multiple volumes can be mounted to the same instance. These storage volumes behave like raw, unformatted block devices, with user-supplied device names and a block device interface.

Amazon SimpleDB Service

SimpleDB provides a simplified data model based on the relational database data model. Structured data from users must be organized into domains. Each domain can be considered a table. The items are the rows in the table. A cell in the table is recognized as the value for a specific attribute (column name) of the corresponding row. it is possible to assign multiple values to a single cell in the table. This is not permitted in a traditional relational database. SimpleDB, like Azure Table, could be called "LittleTable" as they are aimed at managing small amounts of information stored in a distributed table.