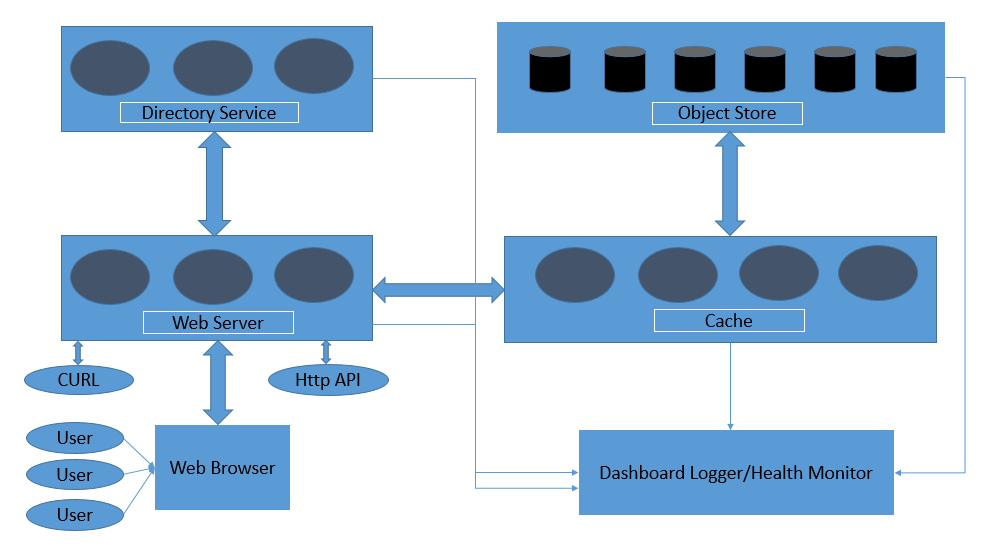
**Facebook Photo Object Store Server Design Thoughts**

**Use Cases:**

* Multiple users should be able to upload photos through web browser.
* Users should be able to view their own photos or other users photos where they got access permission uploaded through web browser.
* Object Store server should be scalable, high throughput, low latency, and fault tolerant like system failures.

**Sample Design Diagram:**



**Components:**

These are the below components which we can use to design a photo object store server design.

**Web Browser:**

Multiple users connect through the web browser to photo upload/download to web server of a photo sharing site like Facebook.com, Curl and other Http APIs can also connect with web browser.

**Web Server:**

All users connect to the web server through the web browser. As there can be multiple users simultaneously accessing the web server so we can have multiple web servers to balance the load of multiple users through load balancer between web browser and web servers.

Also web servers will be handling user authentication and user ACL permission handling for photo upload and download.

Web server will contact directory service for photo upload/download request to make the URL path from available volumes in the object store or from cache.

Web server will support minimum CRUD operation of Http server.

**Directory Service:**

Web server will contact directory service to check the available logical or physical volume on object store for a new user request for photo upload or download. Logical volumes are internally mapped to physical volumes and contains multiple physical volumes which director service maps it. This will load balance the writes across the logical volumes and reads from physical volumes in the object store. Directory checks the balances if any object store physical volumes are filled then those volumes are not counted for write request and used for only read requests. So this will do all the load balancing for write and read requests. If any physical volume is filled or its store machine health checkup failed then directory service will remove that store machine and physical volume from new photo upload requests. Those machines where physical volumes are filled those will be used only for photo read requests and all this information directory service will store as a metadata and will use for photo write and read requests.

Directory service will generate the keys for the photo id which can be of 20/16/8 bytes unique UUIDs generated from md5 or sha1 hash. Also it will generate random cookie for photo url which can be of 2 or 4 or 8 byte size. These keys will be used as photo ids for mapping. So when a photo is uploaded then it will check the available logical volume id where photos can be stored (logical volume id will contain machine id and physical volume ids of different object store machines). Directory service will generate above photo ids components like keys and cookie to upload the photo and these keys, cookie will be used as photo read requests.

**Cache:**

Cache stores the recently uploaded photos into the object store logical volumes. Cache can be like distributed hash table and key as photo id. Also there will be cached data age limit through which cached data will no longer be served to web server and it will be refreshed from store machines.

**Object Store:**

Object Store contains multiple machines which contains multiple physical volumes where the photos of users reside. These physical volumes are assigned unique id along with store machine ids. These physical volumes contains multiple photos which are appended after one another photo like in a file append only writes. So every physical volume behaves a large file which gets every photo writes in append mode. Store machine can access a particular photo if we can get the file offset and size of the photo in that physical volume. Physical volume can be uniquely identified as above with its id and machine id.

Each store machine stores all the physical volumes ids it has and also stores the information of mapping of photo ids (keys, cookie etc) with file offset, flag, size and physical volume where photo resides. These in-mapping memory is stored in an index file which is used when a store machine crashes or reboots. This index file contains photo key, all the 4 different sizes of the uploaded photo (large, medium, small, thumbnail sizes)

This append only photo store design is called haystack design with storing needle as photo in large physical volume of size of 100 GB like a haystack.

Each photos which are stored as needle in the physical volume file contains header and data which is actual photo data. Needle header can contain the photo keys, cookie (random number generated by director lookup for creating url of photo), offset, flag (photo deleted or not), size etc.

**Operations:**

**Photo write:**

When a photo is uploaded then webserver gets logical volume id, keys, cookie from Directory service which forms the photo url and webserver sends these information along with photo data to the store machine. As logical volumes contains multiple physical volumes spread across various store machines so every photo write goes to all the physical volumes associated with logical volume to maintain redundancy of data to prevent future errors of physical volumes. All these photo write is only append mode to the physical volume as a new needle and this new needle metadata mapping is updated into the corresponding store machine in-mapping memory. If photo is modified then a new needle is created either is same physical volume or different volume. If new needle is added into the different volume then this new needle photo keys, cookie and physical volume is updated into the directory service and also into the new physical volume store machine in-mapping memory.

**Photo Read:**

When web server requests the photo then photo is either served from cache through photo id (key, cookie) or from store machine. Web server gets photo url from directory and this url contains physical volume id, machine id, cookie and keys. When store machine gets photo read request from cache or web server then it checks the in-mapping memory for photo id (we get this from directory service) to physical volume id, file offset, photo size and flag (if photo is not deleted) and then exact needle data is read from the physical volume file and sent to webserver or cache.

**Photo Delete:**

Photo delete operation store machine in-mapping memory changes the flag value for the photo id as deleted and also in the physical volume needle header flag value.

**Dashboard Logger/Health Monitor:**

We can have a dashboard logger and health monitor node which will fetch logs from web server, directory service, cache and object store machine. These individual web server, directory service, cache and object store machine will individually talk or ping to their own nodes for health checkup every few seconds and if a node is not responding to multiple pings or messages then that node will be treated as down and the load balancer in these services will send the log message to dashboard logger for health issue of a node so that we can check the node issue.

Also it will collect the IO stats from individual load balancers of web server, cache, directory service and object store and report into the dash board logging through APIs or messages.

Also individual load balancers will report for any scalable issue in any of the above services if further nodes are required for addition and logs and messages will be sent to the dashboard logger and health monitor.