

**Mid Term Exam I (CS 553: Internet Services)**  
**April 6, 2009**

1. [10 points] a stateless load balancer does not maintain a table that provides a conversion between the inbound address and the outbound address on a session basis. Instead it uses a hash function to convert the inbound address (IP, port) to outbound (serverIP, port). Stateless load balancer is simple but treats all client connections the same and hence can potentially map all sessions from the same client to the same server. Stateful load balancers maintain a session table. For each, session, a corresponding input, output address mapping is maintained. Based, on existing load, the load balancer can map any input session to any output server to better balance the load.

A load balancer with VIP = 128.6.3.4 is connected to two servers S1 and S2 with the same VIP. Assume the MAC addresses of S1 and S2 are M1 and M2 respectively. A packet from some client with IP address 198.6.4.2 can be routed to S1 at port 80 by rewriting the destination address of the packet destined for VIP, 80 with MAC address M1 [198.6.4.2, P1,128.6.3.4, 80,M1] and a packet from 198.6.4.2 can be routed to S2 at port 21 [198.6.4.2, P1,128.6.3.4, 21,M2] by rewriting the destination address of the packet destined for VIP, 21 with MAC address M2.

2. [20 points]

1. In shadowing, a log entry is created in the backup site and during recovery the log is applied to the backup to create the latest copy of the data. Writes are efficient but recovery is not instantaneous. In mirroring, writes are synchronous on the backup copy. Each write on the master is also written to the back up. Writes take more time but recovery is instantaneous as the back up copy is a mirror of the master. Chained-declustering (as in petal) stripes blocks on neighboring servers. On a failure of 1 server, the load is distributed evenly among the neighbors as opposed to mirroring or shadowing where 100% of the load is taken by the backup when the primary fails.
2. The liveness module (Figure 3 in the Petal paper) in petal is used by each server to know who else is alive so that each server knows the set of live servers. This is implemented by sending keep



With finger table, the closest preceding node is 2. so ask 2 which in turn asks node 5 that has the key.

4. If the key would be placed on a random node on the ring, then the write would be very efficient as the cost would be  $O(1)$ , assuming the randomly chosen node is alive. However, this method increases the search cost to  $O(N)$ . Hence, it would make sense to periodically reorganize so that keys migrate to  $\text{succ}(k)$  so that, using the finger table, the search cost could be reduced to  $O(\log(N))$ .

4. [20 points]

1. In GFS, clients do not cache data because GFS clients tend to read through the entire chunk of data in one pass and the working set is too large for caching to be effective but do cache metadata to reduce the load on the master server as the master server is the bottleneck for the GFS.

2. The chunk size in GFS is 64 MB so that the metadata required per chunk can fit in the main memory of the master server. This way, any request for metadata would not require disk access as the performance of the master is critical for GFS. If the chunk size doubled to 128MB, then the metadata size for a given file system would be halved. However, wastage due to internal fragmentation would double; especially for append operations.

3.

1. `buf < byterange` **true** as the data in the chunk can be less than the byte range (esp the last chunk of a file)
2. `buf=byterange` **true** as the data in the chunk can be equal to the byte range
3. `buf > byterange` **false** as the client never caches data and hence does not retrieve more than the byte range.

4. Total number of network operations is 6.

1. C- master

1. Open a file, send the pathname, get the file handle
2. send the file handle, chunkid, get the chunk server id

2. C-Chunk server

1. send read of curr-offset, 32 M to chunk server

2. (as meta data is cached) send next read of curr-offset,32 M to chunk server
3. C-master
  1. (need to access next chunk as chunk size is 64 M) send the handle, chunkid, get the chunk server id for the next 32M
4. C-Chunk server
  1. send read of curr-offset, 32 M to chunk server
5. [20 points] Map-reduce
  1. Represent the directed graph  $G=(V,E)$  as an adjacency list. Pass adjacency list for each node to the mapper  $k1,v1=(node, adjlist)$ . The mapper emits a 1 for each node in the adj list.  $k2, v2=(node, 1)$ . The reducer adds up all the values for each node and produces  $(node, sum)$  which is the in-degree of that node.
  2. Consider the relation  $D=[employee,department,salary]$ . Horizontally Partition the relation and input horizontal fragments to the mapper  $(employee,[dept,salary])$ . Mapper outputs for each tuple  $(department, salary)$ . Reducer combines salary for each department to compute average. And outputs  $(dept, average)$
6. [15 points]
  1. Bigtable is a sparse table because many cells in the big table could be empty. No, The same cannot be said for a relational table as every cell needs to have a value or a NULL value.
  2. Big table with row key as cust-id, columns are cname, cstreet, and ccity. Column families are deposits and loans. The map for column family is acct number and balance. The map for column family loans is loan number and amount.

cust id	cname	cstreet	ccity	Deposits		Loans	
				Acct#	balance	Loan#	amt
1	smith	Holly st	edison	s-1	4324	l-2 l-34	3000 1000
2	brad	Rodeo st	piscat	c-1 s-2 c-2	200 123 234	l-21	2000
3	pitt	Famous st	beverly	c-3 s-3	4432 102		
4	jolie	Lost st	hollywood	c-4	50	l-43	700

		1-44	4323
		1-48	231