Algo Compute\_LB\_rules:

NOTATION: Sort(set, criteria) returns set in sorted order by criteria

KV = {(key1, value1), (key2, value2), } denotes a set of key-value pairs

KV[key] returns value for the key

KV.keys() returns set of keys, e.g., {key1, key2, …}

Input:

Active\_servers = {s1, …}

Content = {c1, … }

Content\_demand = {(c1, d1), } , //

LB\_prev = {c1,, rule\_1 .. } where rule1 = {(s1, p1), (s2, p2)} such request is sent to server s\_i with probabilty p\_i.

t // threshold utilization

Output:

LB\_new = {} // New set of load balancing rules

Load\_estimate = {} // variable keeps track of load assigned to each server; initialized to zero for all active servers

For s\_i in Active\_servers:

Load\_estimate = Load\_estimate \cup {(s\_i, 0)}

For (c\_k, d\_k) in Sort(Content\_demand, *increasing demand d\_k*):

Prev\_rule = LB\_prev[c\_k] // previous LB rule for content

S’\_k = Prev\_rule.keys() // set of servers in previous LB rule

if S’\_k not \subset Active\_servers: // all servers in Prev\_rule are not active

Continue

Flag = True

For s\_i in prev\_rule.keys():

If load\_estimate[s1] + d\_k \* prev\_rule[s\_i] > t:

Flag = false

Break

If flag:

LB\_new[c\_k] = Prev\_rule // Copy rule

For (c\_k, d\_k) in Sort(Content\_demand, *decreasing demand d\_k*):

If c\_k not \in LB\_new:

LB\_new[c\_k] = Get\_new\_rule(d\_k)

Algo Get\_new\_rule(load\_estimate, demand):

Rule = {}

Remaining\_demand = demand

For (s\_i, l\_i) in Sorted(load\_estimate, increasing l\_i): // consider less loaded servers first

Served\_demand = min(1 – l\_i, remaining\_demand)

Rule[s\_i] = served\_demand

Remaining\_demand -= served\_demand

Return rule