Cache Replacement Policies

This document contains the algorithms used for Cache Replacement in Edge Cloud.

- LRU - Least Recently Used
- FIFO - First In First Out
- LFU - Least Frequently Used
- GDSF - Greedy Dual Size Frequency
- SCRP - Service Aware Cache Replacement Policy

Algorithm Image class

▷ Class to create free & running set of images.
Image:
    String job_id ▷ Stores the job id of the service
    int frequency ▷ Used only in LFU & GDSF; Stores the frequency of a service
    int stay ▷ Used only in SCRP; stores the frequency of a service in the cache
    double priority ▷ Used only in GDSF; Stores the priority of a service
    double SCRP ▷ Used only in SCRP; Stores the SCRP score of a service
    long timestamp ▷ Stores the timestamp of the service
    long task_index ▷ Stores the task index of the service
    double cpu ▷ Stores the CPU requirement of the service
    double ram ▷ Stores the RAM requirement of the service
    double runtime_disk ▷ Stores the runtime disk requirement of the service
    double storage ▷ Stores the amount of storage required by service to be cloned in cache

Image(String job_id, long timestamp, long task_index, double cpu, double ram, double runtime_disk, double storage):
    ▷ Constructor to initialize instance of Image class
    $this.variable = variable$ ∀ variables

Image(String job_id, int frequency, long task_index, double cpu, double ram, double runtime_disk, double storage):
    ▷ Modified constructor for LFU
    $this.variable = variable$ ∀ variables

Image(String job_id, int frequency, double priority ,long task_index, double cpu, double ram, double runtime_disk, double storage):
    ▷ Modified constructor for GDSF
    $this.variable = variable$ ∀ variables
Image(String job_id, int frequency, int stay, double SCRP ,long task_index, double cpu, double ram, double runtime_disk, double storage):
    ▷ Modified constructor for SCRP
    $this.variable = variable$ ∀ variables

Algorithm Minimum Timestamp Finder

▷ Function to find service id that has minimum timestamp from the set of images for LRU & FIFO
String **min_timestamp**(Hashmap$< String, Image >$ hm)
    String **min_ts** = ""
    long **ts** = long.MAX_VALUE
    hm.forEach((Service_id, Image) → ▷ Iterating over whole set of images
        if Image.timestamp < ts **then**
            $ts = Image.timestamp$
            $min\_ts = service\_id$
        end if
    return min_ts ▷ Service id having minimum(oldest) timestamp

Algorithm Cache class to store the cloud and service data

Cache:
- int no_services = 0 ▷ no of services currently in the Edge Cache
- double CPU_{max} = 1.00 ▷ max CPU available in cache(scaled to 1)
- double RAM_{max} = 1.00 ▷ max RAM available in cache(scaled to 1)
- double $Storage_{max}$ = 1.00 ▷ max storage available in cache(scaled to 1)
- double $Runtime\_disk_{max}$ = 1.00 ▷ max runtime disk available(scaled to 1)
- double $Disk\_Storage_{max}$ = 2.00 ▷ max sum of runtime disk and storage
- double CPU_{curr} = 0 ▷ Current usage of CPU in Edge Server
- double RAM_{curr} = 0 ▷ Current usage of RAM in Edge Server
- double $Runtime\_disk_{curr}$ = 0 ▷ Current usage of runtime disk in Edge Server
- double $Storage_{curr} = 0$ ▷ Current usage of Storage in cache
- Hashmap $< String, Image >$ Running ▷ Contains images of all running services and their service id
- Hashmap $< String, Image >$ Free ▷ Contains images of all free services and their service id

Algorithm Least Frequency Finder

```
▷ Function to find service id that has least frequency from the set of images for LFU
String min_frequency(Hashmap< String, Image > hm)
    String min_freq = ""
    int freq = Integer.MAX_VALUE
    hm.forEach((Service_id, Image) →                ▷ Iterating over whole set of images
        if Image.Frequency < freq then
            freq = Image.Frequency
            min_freq = Service_id
        end if

    return min_freq                                 ▷ Service id having least frequency
```

Algorithm Priority Calculator for GDSF

```
▷ Function to calculate priority of a service for GDSF
double priority_calc(double clock, int frequency, double size)
    if size == 0 then                               ▷ Services consuming 0 disk resources
        return Double.MAX_VALUE
    end if

    return (clock + frequency/size)
```

Algorithm Least Priority Finder

```
▷ Function to find service id that has least priority from the set of images
String min_priority(Hashmap< String, Image > hm)
    String min_priority = ""
    double priority_min = Double.MAX_VALUE
    hm.forEach((Service_id, Image) →                ▷ Iterating over whole set of images
        if Image.priority < priority_min then
            priority_min = Image.priority
            min_priority = Service_id
        end if

    return min_priority                             ▷ Service id having least priority
```

Algorithm LRU Replacement in Edge Cloud

int cache_hit = 0 ▷ No of cache hit
while Cloud receives request **do**
  String[ ] $tokens$ = request.split() ▷ Contains all information in the request
  String $service\_id$ = tokens[0] ▷ Generate service id for the service
  double CPU_{req} = tokens[3], RAM_{req} = tokens[4], $Runtime\_Disk_{req}$ = tokens[5] ▷ Service requirements
  double $Storage_{req}$ = 0 ▷ Assuming it be 0 for now
  if Cache.Running.containsKey(service_id) **then** ▷ If the service is cached
    $cache\_hit++$
    Image to_update = Cache.Running.get(service_id)
    $to\_update.timestamp$ = current_timestamp ▷ Update its timestamp
    Cache.Running.put(service_id, to_update)
  else if Cache.Free.containsKey(service_id) **then** ▷ If the image of service is present in Free list
    Image $replace$ = $Cache.Free$.get(service_id)
    if There are enough resources in the edge cloud **then**
      $replace.timestamp$ = current_timestamp
      $Cache.Running$.put(service_id, replace) ▷ Adding it to the Running list
      $Cache.CPU_{curr}$ += replace.CPU ▷ Updating current CPU status
      $Cache.RAM_{curr}$ += replace.RAM ▷ Updating current RAM status
      $Cache.Runtime\_disk_{curr}$ += replace.$Runtime\_disk$ ▷ Updating current Runtime Disk status
      $Cache.Storage_{curr}$ += replace.$Storage$ ▷ Updating current storage status
      $Cache.Free$.remove(replace) ▷ Removing it from free list
    end if
  else ▷ If the service is not cached
    $Running' \leftarrow Cache.Running$
    $Free' \leftarrow Cache.Free$ ▷ Creating copies. Update cloud only if enough resources are available
    $CPU'_{curr} \leftarrow Cache.CPU_{curr}$
    $RAM'_{curr} \leftarrow Cache.RAM_{curr}$ ▷ Creating copies of Cache Status
    $Runtime\_disk'_{curr} \leftarrow Cache.Runtime\_disk_{curr}$
    $Storage'_{curr} \leftarrow Cache.Storage_{curr}$
    while There is no enough resources in edge cloud **do**
      if $CPU_{req} + CPU'_{curr} > Cache.CPU_{max} \,||\, RAM_{req} + RAM'_{curr} > Cache.RAM_{max}$ **then**
        String to_remove = min_timestamp($Running'$) ▷ Finding service id with least time stamp
        Image $remove$ = $Running'$.get(to_remove)
        $Free'$.put(to_remove, remove) ▷ Inserting its image to set of free images
        CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
        RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
        $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
        $Running'$.remove(to_remove) ▷ Removing its image from set of running images
      end if
      if $Disk\_Storage_{req} + Disk\_Storage_{curr} > Cache.Disk\_Storage_{max}$ **then**
        ▷ $Disk\_Storage_{curr} = Runtime\_disk_{curr} + Storage_{curr}$
        if $Free'$.isEmpty() **then** ▷ If there are no free images
          String to_remove = min_timestamp($Running'$) ▷ Finding service id with min timestamp
          Image $remove$ = $Running'$.get(to_remove)
          $Free'$.put(to_remove, remove) ▷ Adding to Free set of images
          CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
          RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
          $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
          $Running'$.remove(to_remove) ▷ Removing from running set of images
        else
          String to_remove = min_timestamp($Free'$) ▷ Finding service with min timestamp
          Image $remove$ = $Free'$.get(to_remove)
          $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
          $Storage'_{curr}$ -= remove.$Storage$ ▷ Updating current storage status
          $Free'$.remove(to_remove) ▷ Removing the service from cache
        end if
      end if
    end while
    ▷ continued...

Algorithm LRU Replacement in Edge Cloud - continued

▷ Continued...
$Cache.Running \leftarrow Running'$
$Cache.Free \leftarrow Free'$
$Cache.var_{curr} = var'_{curr}$ for $var \in$ {CPU, RAM, Runtime_disk, Storage}

$Cache.Running$.put(service_id, new Image(tokens[0],curr_timestamp,tokens[2],tokens[3],tokens[4],tokens[5], storage)
▷ Cloning new service with Current Timestamp
if $Cache.Free$.containsKey(service_id) **then** ▷ If service is also in free list, remove it from free list
    $Cache.Free$.remove(service_id)
end if
$Cache.CPU_{curr}$ += CPU_{req} ▷ Updating current CPU status
$Cache.RAM_{curr}$ += RAM_{req} ▷ Updating current RAM status
$Cache.Runtime\_disk_{curr}$ += $Runtime\_disk_{req}$ ▷ Updating current Runtime Disk status
$Cache.Storage_{curr}$ += $Storage_{req}$ ▷ Updating current storage in cache
$Cache.no\_services++$ ▷ Increment no of services
end if
end while

Algorithm FIFO replacement in Edge Cloud

int cache_hit = 0 ▷ No of cache hit
while Cloud receives request **do**
  String[ ] $tokens$ = request.split() ▷ Contains all information in the request
  String $service\_id$ = tokens[0] ▷ Generate service id for the service
  double CPU_{req} = tokens[3], RAM_{req} = tokens[4], $Runtime\_Disk_{req}$ = tokens[5] ▷ Service requirements
  double $Storage_{req}$ = 0 ▷ Assuming it be 0 for now

  if Cache.Running.containsKey(service_id) **then** ▷ If the service is cached
    $cache\_hit++$
  else if Cache.Free.containsKey(service_id) **then** ▷ If the image of service is present in Free list
    Image $replace$ = $Cache.Free$.get(service_id)
    if There are enough resources in the edge cloud **then**
      $replace.timestamp$ = current_timestamp
      $Cache.Running$.put(service_id, replace) ▷ Adding it to the Running list
      $Cache.CPU_{curr}$ += replace.CPU ▷ Updating current CPU status
      $Cache.RAM_{curr}$ += replace.RAM ▷ Updating current RAM status
      $Cache.Runtime\_disk_{curr}$ += replace.$Runtime\_disk$ ▷ Updating current Runtime Disk status
      $Cache.Storage_{curr}$ += replace.$Storage$ ▷ Updating current storage status
      $Cache.Free$.remove(replace) ▷ Removing it from free list
    end if
  else ▷ If the service is not cached
    $Running' \leftarrow Cache.Running$
    $Free' \leftarrow Cache.Free$ ▷ Creating copies. Update cloud only if enough resources are available
    $CPU'_{curr} \leftarrow Cache.CPU_{curr}$
    $RAM'_{curr} \leftarrow Cache.RAM_{curr}$ ▷ Creating copies of Cache Status
    $Runtime\_disk'_{curr} \leftarrow Cache.Runtime\_disk_{curr}$
    $Storage'_{curr} \leftarrow Cache.Storage_{curr}$
    while There is no enough resources in edge cloud **do**
      if $CPU_{req} + CPU'_{curr} > Cache.CPU_{max} \,||\, RAM_{req} + RAM'_{curr} > Cache.RAM_{max}$ **then**
        String to_remove = min_timestamp($Running'$) ▷ Finding service id with least time stamp
        Image $remove$ = $Running'$.get(to_remove)
        $Free'$.put(to_remove, remove) ▷ Inserting its image to set of free images
        CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
        RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
        $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
        $Running'$.remove(to_remove) ▷ Removing its image from set of running images
      end if
      if $Disk\_Storage_{req} + Disk\_Storage_{curr} > Cache.Disk\_Storage_{max}$ **then**
        ▷ $Disk\_Storage_{curr} = Runtime\_disk_{curr} + Storage_{curr}$
        if $Free'$.isEmpty() **then** ▷ If there are no free images
          String to_remove = min_timestamp($Running'$) ▷ Finding service id with min timestamp
          Image $remove$ = $Running'$.get(to_remove)
          $Free'$.put(to_remove, remove) ▷ Adding to Free set of images
          CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
          RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
          $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
          $Running'$.remove(to_remove) ▷ Removing from running set of images
        else
          String to_remove = min_timestamp($Free'$) ▷ Finding service with min timestamp
          Image $remove$ = $Free'$.get(to_remove)
          $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
          $Storage_{curr}$ -= remove.$Storage$ ▷ Updating current storage status
          $Free'$.remove(to_remove) ▷ Removing the service from cache
        end if
      end if
    end while
    ▷Continued...

Algorithm FIFO replacement in Edge Cloud - continued...

▷ Continued...
$Cache.Running \leftarrow Running'$ & $Cache.Free \leftarrow Free'$
$Cache.var_{curr} = var'_{curr}$ for $var \in$ {CPU, RAM, Runtime_disk, Storage}
$Cache.Running$.put(service_id, new Image(tokens[0],curr_timestamp,tokens[2],tokens[3],tokens[4],tokens[5], storage)
▷ Cloning new service
if $Cache.Free$.containsKey(service_id) **then** ▷ If service is also in free list, remove it from free list
    $Cache.Free$.remove(service_id)
end if
$Cache.CPU_{curr}$ += CPU_{req} ▷ Updating current CPU status
$Cache.RAM_{curr}$ += RAM_{req} ▷ Updating current RAM status
$Cache.Runtime\_disk_{curr}$ += $Runtime\_disk_{req}$ ▷ Updating current Runtime Disk status
$Cache.Storage_{curr}$ += $Storage_{req}$ ▷ Updating current storage in cache
$Cache.no\_services++$ ▷ Increment no of services
end if
end while

Algorithm LFU replacement in Edge Cloud

int cache_hit = 0 ▷ No of cache hit
while Cloud receives request **do**
  String[ ] $tokens$ = request.split() ▷ Contains all information in the request
  String $service\_id$ = tokens[0] ▷ Generate service id for the service
  double CPU_{req} = tokens[3], RAM_{req} = tokens[4], $Runtime\_Disk_{req}$ = tokens[5] ▷ Service requirements
  double $Storage_{req}$ = 0 ▷ Assuming it be 0 for now
  if Cache.Running.containsKey(service_id) **then** ▷ If the service is cached
    $cache\_hit++$
    Image to_update = Cache.Running.get(service_id)
    $to\_update.frequency$++ ▷ Increase its frequency by 1
    Cache.Running.put(service_id, to_update)
  else if Cache.Free.containsKey(service_id) **then** ▷ If the image of service is present in Free list
    Image $replace$ = $Cache.Free$.get(service_id)
    if There are enough resources in the edge cloud **then**
      $replace.frequency$ = 1
      $Cache.Running$.put(service_id, replace) ▷ Adding it to the Running list
      $Cache.CPU_{curr}$ += replace.CPU ▷ Updating current CPU status
      $Cache.RAM_{curr}$ += replace.RAM ▷ Updating current RAM status
      $Cache.Runtime\_disk_{curr}$ += replace.$Runtime\_disk$ ▷ Updating current Runtime Disk status
      $Cache.Storage_{curr}$ += replace.$Storage$ ▷ Updating current storage status
      $Cache.Free$.remove(replace) ▷ Removing it from free list
    end if
  else ▷ If the service is not cached
    $Running' \leftarrow Cache.Running$
    $Free' \leftarrow Cache.Free$ ▷ Creating copies. Update cloud only if enough resources are available
    $CPU'_{curr} \leftarrow Cache.CPU_{curr}$
    $RAM'_{curr} \leftarrow Cache.RAM_{curr}$ & ▷ Creating copies of Cache Status
    $Runtime\_disk'_{curr} \leftarrow Cache.Runtime\_disk_{curr}$
    $Storage'_{curr} \leftarrow Cache.Storage_{curr}$
    while There is no enough resources in edge cloud **do**
      if $CPU_{req} + CPU'_{curr} > Cache.CPU_{max} \,||\, RAM_{req} + RAM'_{curr} > Cache.RAM_{max}$ **then**
        String to_remove = min_frequency($Running'$) ▷ Finding service id with least frequency
        Image $remove$ = $Running'$.get(to_remove)
        $Free'$.put(to_remove, remove) ▷ Inserting its image to set of free images
        CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
        RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
        $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
        $Running'$.remove(to_remove) ▷ Removing its image from set of running images
      end if
      if $Disk\_Storage_{req} + Disk\_Storage_{curr} > Cache.Disk\_Storage_{max}$ **then**
        ▷ $Disk\_Storage_{curr} = Runtime\_disk_{curr} + Storage_{curr}$
        if $Free'$.isEmpty() **then** ▷ If there are no free images
          String to_remove = min_frequency($Running'$) ▷ Finding service id with least frequency
          Image $remove$ = $Running'$.get(to_remove)
          $Free'$.put(to_remove, remove) ▷ Adding to Free set of images
          CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
          RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
          $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
          $Running'$.remove(to_remove) ▷ Removing from running set of images
        else
          String to_remove = min_frequency($Free'$) ▷ Finding service with least frequency
          Image $remove$ = $Free'$.get(to_remove)
          $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
          $Storage'_{curr}$ -= remove.$Storage$ ▷ Updating current storage status
          $Free'$.remove(to_remove) ▷ Removing the service from cache
        end if
      end if
    end while
    ▷ Continued...

Algorithm LFU replacement in Edge Cloud - continued

▷ Continued...
$Cache.Running \leftarrow Running'$ & $Cache.Free \leftarrow Free'$
$Cache.var_{curr} = var'_{curr}$ for $var \in$ {CPU, RAM, Runtime_disk, Storage}
$Cache.Running$.put(service_id, new Image(tokens[0],1,tokens[2],tokens[3],tokens[4],tokens[5], storage)
▷ Cloning new service with frequency 1
if $Cache.Free$.containsKey(service_id) **then** ▷ If service is also in free list, remove it from free list
    $Cache.Free$.remove(service_id)
end if
$Cache.CPU_{curr}$ += CPU_{req} ▷ Updating current CPU status
$Cache.RAM_{curr}$ += RAM_{req} ▷ Updating current RAM status
$Cache.Runtime\_disk_{curr}$ += $Runtime\_disk_{req}$ ▷ Updating current Runtime Disk status
$Cache.Storage_{curr}$ += $Storage_{req}$ ▷ Updating current storage in cache
$Cache.no\_services++$ ▷ Increment no of services
end if
end while

Algorithm GDSF replacement in Edge Cloud

int cache_hit = 0 ▷ No of cache hit
double clock = 0.00 ▷ Clock for GDSF priority calculation
while Cloud receives request **do**
    String[ ] $tokens$ = request.split() ▷ Contains all information in the request
    String $service\_id$ = tokens[0] ▷ Generate service id for the service
    double CPU_{req} = tokens[3], RAM_{req} = tokens[4], $Runtime\_Disk_{req}$ = tokens[5] ▷ Service requirements
    double $Storage_{req}$ = 0 ▷ Assuming it be 0 for now
    if Cache.Running.containsKey(service_id) **then** ▷ If the service is cached
        $cache\_hit++$
        Image to_update = Cache.Running.get(service_id)
        $to\_update.frequency$++ ▷ Increase its frequency by 1
        $to\_update.priority$ = priority_calc(clock, to_update.frequency, Storage_req)
        ▷ Recalculating priority with updated frequency
        Cache.Running.put(service_id, to_update)
    else if Cache.Free.containsKey(service_id) **then** ▷ If the image of service is present in Free list
        Image $replace$ = $Cache.Free$.get(service_id)
        if There are enough resources in the edge cloud **then**
            $replace.frequency$ = 1
            $replace.priority$ = priority_calc(clock, replace.frequency, Storage_req)
            $Cache.Running$.put(service_id, replace) ▷ Adding it to the Running list
            $Cache.CPU_{curr}$ += replace.CPU ▷ Updating current CPU status
            $Cache.RAM_{curr}$ += replace.RAM ▷ Updating current RAM status
            $Cache.Runtime\_disk_{curr}$ += replace.$Runtime\_disk$ ▷ Updating current Runtime Disk status
            $Cache.Storage_{curr}$ += replace.$Storage$ ▷ Updating current storage status
            $Cache.Free$.remove(replace) ▷ Removing it from free list
        end if
    else ▷ If the service is not cached
        $Running' \leftarrow Cache.Running$
        $Free' \leftarrow Cache.Free$ ▷ Creating copies. Update cloud only if enough resources are available
        $CPU'_{curr} \leftarrow Cache.CPU_{curr}$
        $RAM'_{curr} \leftarrow Cache.RAM_{curr}$ ▷ Creating copies of Cache Status
        $Runtime\_disk'_{curr} \leftarrow Cache.Runtime\_disk_{curr}$
        $Storage'_{curr} \leftarrow Cache.Storage_{curr}$
        $clock' \leftarrow clock$
        while There is no enough resources in edge cloud **do**
            if $CPU_{req} + CPU'_{curr} > Cache.CPU_{max}$ || $RAM_{req} + RAM'_{curr} > Cache.RAM_{max}$ **then**
                String to_remove = min_priority($Running'$) ▷ Finding service id with least priority
                Image $remove$ = $Running'$.get(to_remove)
                $Free'$.put(to_remove, remove) ▷ Inserting its image to set of free images
                CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
                RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
                $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
                $clock'$ += remove.priority ▷ Updating clock
                $Running'$.remove(to_remove) ▷ Removing its image from set of running images
            end if
            if $Disk\_Storage_{req} + Disk\_Storage_{curr} > Cache.Disk\_Storage_{max}$ **then**
                ▷ $Disk\_Storage_{curr} = Runtime\_disk_{curr} + Storage_{curr}$
                if $Free'$.isEmpty() **then** ▷ If there are no free images
                    String to_remove = min_priority($Running'$) ▷ Finding service id with least priority
                    Image $remove$ = $Running'$.get(to_remove)
                    $Free'$.put(to_remove, remove) ▷ Adding to Free set of images
                    CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
                    RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
                    $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
                    $clock'$ += remove.priority ▷ Updating clock
                    $Running'$.remove(to_remove) ▷ Removing from running set of images
                else
                    String to_remove = min_priority($Free'$) ▷ Finding service with least priority
                    Image $remove$ = $Free'$.get(to_remove)
                    ▷ Continued...

Algorithm GDSF replacement in Edge Cloud - continued

▷ Continued...
$Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
$Storage'_{curr}$ -= remove.$Storage$ ▷ Updating current storage status
$Free'$.remove(to_remove) ▷ Removing the service from cache
end if
end if
end while
$Cache.Running \leftarrow Running'$ & $Cache.Free \leftarrow Free'$
$Cache.var_{curr} = var'_{curr}$ for $var \in$ {CPU, RAM, Runtime_disk, Storage}

double Pr = priority_calc(clock, 1, Storage_req) ▷ Calculating priority of new service
$Cache.Running$.put(service_id, new Image(tokens[0],1,Pr,tokens[2],tokens[3],tokens[4],tokens[5], storage)
▷ Cloning new service with frequency 1
if $Cache.Free$.containsKey(service_id) **then** ▷ If service is also in free list, remove it from free list
$Cache.Free$.remove(service_id)
end if
$Cache.CPU_{curr}$ += CPU_{req} ▷ Updating current CPU status
$Cache.RAM_{curr}$ += RAM_{req} ▷ Updating current RAM status
$Cache.Runtime\_disk_{curr}$ += $Runtime\_disk_{req}$ ▷ Updating current Runtime Disk status
$Cache.Storage_{curr}$ += $Storage_{req}$ ▷ Updating current storage in cache
$Cache.no\_services++$ ▷ Increment no of services
end if
end while

Algorithm SCRP score calculator

▷ Function to calculate SCRP score of a service for S-cache replacement
double **SCRP_calc**(Image service, Cache cache)
    double Disk = service.Runtime_disk + service.Storage
    double R = $\frac{cache.CPU_{curr}}{cache.CPU_{max}} \times service.CPU + \frac{cache.RAM_{curr}}{cache.RAM_{max}} \times service.RAM + \frac{cache.Disk_{curr}}{cache.Disk_{max}} \times service.Disk$

    if R == 0 **then** ▷ If no resources are being used
        return Double.MAX_VALUE
    end if

    return $\frac{service.frequency}{service.stay \times R}$ ▷ SCRP score of the service

Algorithm Least SCRP score Finder

▷ Function to find service id that has least SCRP score from the set of images for SCRP
String **min_SCRP**(Hashmap< $String, Image$ >hm, Cache cache)
    String min_SCRP = ""
    double SCRP_min = Double.MAX_VALUE
    hm.forEach((Service_id, Image) → ▷ Iterating over the set of images
    Image.SCRP = SCRP_calc(Image, cache) ▷ Calculating the SCRP score for the service
    if Image.SCRP < SCRP_min **then**
        SCRP_min = Image.SCRP
        min_SCRP = Service_id
    end if
    return min_SCRP ▷ Service id of the service having least SCRP score

Algorithm Least Priority Finder for GDSF in SCRP

▷ Function to find service id that has least priority from the set of images
String **min_priority**(double clock, Hashmap< $String, Image$ > hm, Cache cache)
    String min_priority = ""
    double priority_min = Double.MAX_VALUE
    hm.forEach((Service_id, Image) → ▷ Iterating over whole set of images
    double pr = cache.priority_calc(clock, Image.stay, Image.Storage) ▷ Calculating the priority for each image
        if pr < priority_min **then**
            priority_min = Image.priority
            min_priority = $Service\_id$
        end if

    return min_priority ▷ Service id having least priority

Algorithm SCRP replacement in Edge Cloud

int cache_hit = 0
double clock = 0.00 ▷ Clock for GDSF replacement
while Cloud receives request **do**
    Cache.Running.forEach((Service_id, Image) → ▷ Increase stay of each service in Running list
        $Image.stay$++
    Cache.Free.forEach((Service_id, Image) → ▷ Increase stay of each service in Free list
        $Image.stay$++
    String[ ] $tokens$ = request.split() ▷ Contains all information in the request
    String $service\_id$ = tokens[0] ▷ Generate service id for the service
    double CPU_{req} = tokens[3], RAM_{req} = tokens[4], $Runtime\_Disk_{req}$ = tokens[5] ▷ Service requirements
    double $Storage_{req}$ = 0 ▷ Assuming it be 0 for now
    Image $New\_service$ = new Image(service_id,1,1,0,tokens[2],tokens[3],tokens[4],tokens[5], $Storage_{req}$) ▷ New service image
    New_service.SCRP = SCRP_calc(New_service, cache) ▷ Calculating SCRP score for new service
    if Cache.Running.containsKey(service_id) **then** ▷ If the service is cached
        Image to_update = Cache.Running.get(service_id)
        $to\_update.frequency$++ ▷ Increase its frequency by 1
        Cache.Running.put(service_id, to_update)
        $cache\_hit$++
    else if Cache.Free.containsKey(service_id) **then** ▷ If the image of service is present in Free list
        Image $replace$ = $Cache.Free$.get(service_id)
        $replace.frequency$++ ▷ Increase its frequency by 1
        if There are enough resources in the edge cloud **then**
            $Cache.Running$.put(service_id, replace) ▷ Adding it to the Running list
            $Cache.CPU_{curr}$ += replace.CPU ▷ Updating current CPU status
            $Cache.RAM_{curr}$ += replace.RAM ▷ Updating current RAM status
            $Cache.Runtime\_disk_{curr}$ += replace.$Runtime\_disk$ ▷ Updating current Runtime Disk status
            $Cache.Storage_{curr}$ += replace.$Storage$ ▷ Updating current storage status
            $Cache.Free$.remove(replace) ▷ Removing it from free list
        else
            $Cache.Free$.put(service_id,replace)
        end if
    else ▷ If the service is not cached
        $Running' \leftarrow Cache.Running$
        $Free' \leftarrow Cache.Free$ ▷ Creating copies. Update cloud only if enough resources are available
        $CPU'_{curr} \leftarrow Cache.CPU_{curr}$
        $RAM'_{curr} \leftarrow Cache.RAM_{curr}$ ▷ Creating copies of Cache Status
        $Runtime\_disk'_{curr} \leftarrow Cache.Runtime\_disk_{curr}$
        $Storage'_{curr} \leftarrow Cache.Storage_{curr}$
        while There is no enough resources in edge cloud **do**
            if $CPU_{req} + CPU'_{curr} > Cache.CPU_{max}$ || $RAM_{req} + RAM'_{curr} > Cache.RAM_{max}$ **then**
                String to_remove = min_SCRP($Running'$,$cache$) ▷ Finding service id with least priority
                Image $remove$ = $Running'$.get(to_remove)
                if New_service.score > remove.score **then**
                    $Free'$.put(to_remove, remove) ▷ Inserting its image to set of free images
                    CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
                    RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
                    $Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
                    $Running'$.remove(to_remove) ▷ Removing its image from set of running images
                else
                    break
                end if
            end if
            if $Disk\_Storage_{req} + Disk\_Storage_{curr} > Cache.Disk\_Storage_{max}$ **then**
                ▷ $Disk\_Storage_{curr} = Runtime\_disk_{curr} + Storage_{curr}$
                if $Free'$.isEmpty() **then** ▷ If there are no free images
                    String to_remove = min_SCRP($Running'$,$cache$) ▷ Finding service id with least frequency
                    Image $remove$ = $Running'$.get(to_remove)
                    if New_service.score > remove.score **then**
                        $Free'$.put(to_remove, remove) ▷ Adding to Free set of images
                        CPU'_{curr} -= remove.CPU ▷ Updating current CPU status
                        RAM'_{curr} -= remove.RAM ▷ Updating current RAM status
                        ▷ Continued...

Algorithm SCRP replacement in Edge Cloud - continued

▷ Continued...
$Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
$Running'$.remove(to_remove) ▷ Removing from running set of images
else
break
end if
else
String to_remove = min_Priority($clock$, $Free'$, $cache$) ▷ Finding service with least priority based on GDSF
Image $remove$ = $Free'$.get(to_remove)
$Runtime\_Disk'_{curr}$ -= remove.$Runtime\_disk$ ▷ Updating current runtime disk status
$Storage'_{curr}$ -= remove.$Storage$ ▷ Updating current storage status
clock += priority_calc(clock, remove.stay, remove.storage)
$Free'$.remove(to_remove) ▷ Removing the service from cache
end if
end if
end while
$Cache.Running \leftarrow Running'$ & $Cache.Free \leftarrow Free'$
$Cache.var_{curr} = var'_{curr}$ for $var \in$ {CPU, RAM, Runtime_disk, Storage}
$Cache.Running$.put(service_id, New_service) ▷ Cloning new service with frequency 1
if $Cache.Free$.containsKey(service_id) **then** ▷ If service is also in free list, remove it from free list
$Cache.Free$.remove(service_id)
end if
$Cache.CPU_{curr}$ += CPU_{req} ▷ Updating current CPU status
$Cache.RAM_{curr}$ += RAM_{req} ▷ Updating current RAM status
$Cache.Runtime\_disk_{curr}$ += $Runtime\_disk_{req}$ ▷ Updating current Runtime Disk status
$Cache.Storage_{curr}$ += $Storage_{req}$ ▷ Updating current storage in cache
$Cache.no\_services++$ ▷ Increment no of services
end if
end while