TUM I 13
Chair of Application and Middleware Systems
Prof. Dr. H.- A. Jacobsen

Tutorial
Distributed Systems (IN2259)

Sample Solution: Exercises on Consistent Hashing and MapReduce

Exercise 1 Traditional Hashing vs. Consistent Hashing

A set of N servers (cf. Table 1.2) should be used to cache a collection of M different web sites (cf. Table 1.1) belonging to an application provider. In order to reduce the load on the web server hashing is applied to distribute the web sites among the web caches.
Note: For subtasks (a) and (b) you can neglect the hash values given in the tables.

Name	ID	Hash Value
WebSite_0	0	1A2E
WebSite_1	1	C649
WebSite_2	2	431C
WebSite_3	3	1665
WebSite_4	4	61B3
WebSite_5	5	9271
WebSite_6	6	1CF3
WebSite_7	7	214D
WebSite_8	8	8715
WebSite_9	9	ECA2

Web sites

Name	ID	Hash Value
WebCache_0	0	7912
WebCache_1	1	CAD4
WebCache_2	2	C23E

Web caches

(a) In a first approach, the hash function $h(id) = 7 \cdot \mathrm{id} + 4 \bmod \mathrm{N}$ is used to associate web sites with caches. Each page is hashed based on its ID and the result of the hash operation is supposed to determine the ID of the corresponding cache. List the set of web sites that is managed by each cache.

Solution:
We have 3 caches so the hash function $h(ID) = 7 \cdot \mathrm{ID} + 4 \; mod \; 3$

Website	ID	$h(ID)$	Webcache
WebSite_0	0	$h(0) = 7 \cdot 0 + 4 \; mod \; 3 = 1$	WebCache_1
WebSite_1	1	$h(1) = 7 \cdot 1 + 4 \; mod \; 3 = 2$	WebCache_2
WebSite_2	2	$h(2) = 7 \cdot 2 + 4 \; mod \; 3 = 0$	WebCache_0
WebSite_3	3	$h(3) = 7 \cdot 3 + 4 \; mod \; 3 = 1$	WebCache_1
WebSite_4	4	$h(4) = 7 \cdot 4 + 4 \; mod \; 3 = 2$	WebCache_2
WebSite_5	5	$h(5) = 7 \cdot 5 + 4 \; mod \; 3 = 0$	WebCache_0
WebSite_6	6	$h(6) = 7 \cdot 6 + 4 \; mod \; 3 = 1$	WebCache_1
WebSite_7	7	$h(7) = 7 \cdot 7 + 4 \; mod \; 3 = 2$	WebCache_2
WebSite_8	8	$h(8) = 7 \cdot 8 + 4 \; mod \; 3 = 0$	WebCache_0
WebSite_9	9	$h(9) = 7 \cdot 9 + 4 \; mod \; 3 = 1$	WebCache_1

Cache	Websites served by cache	#
WebCache_0	WebSite_2, WebSite_5, WebSite_8	3
WebCache_1	WebSite_0, WebSite_3, WebSite_6, WebSite_9	4
WebCache_2	WebSite_1, WebSite_4, WebSite_7	3

(b) How does the situation change when a new server ('WebCache_3', 3, 2F69) is added to the set of caches? Again, list the allocation of web sites to caches. What do you observe? Quantify the degree of reallocation (i.e., the percentage of web sites that need to be transferred).

Solution:
Now, we have 4 caches so the hash function $h(ID) = 7 \cdot \mathrm{ID} + 4 \; mod \; 4$. This results in a total reallocation of web sites

to caches.

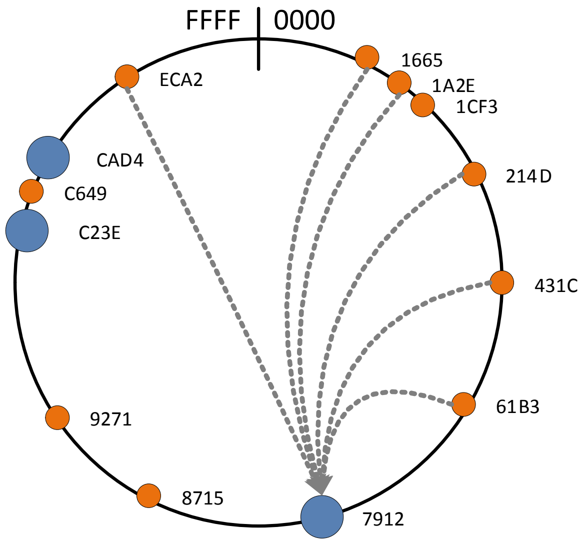
Website	ID	$h(ID) = 7 \cdot \text{ID} + 4$ *mod* 4	Webcache
WebSite_0	0	$h(0) = 7 \cdot 0 + 4 \text{ mod } 4 = 0$	WebCache_0
WebSite_1	1	$h(1) = 7 \cdot 1 + 4 \text{ mod } 4 = 3$	WebCache_3
WebSite_2	2	$h(2) = 7 \cdot 2 + 4 \text{ mod } 4 = 2$	WebCache_2
WebSite_3	3	$h(3) = 7 \cdot 3 + 4 \text{ mod } 4 = 1$	WebCache_1
WebSite_4	4	$h(4) = 7 \cdot 4 + 4 \text{ mod } 4 = 0$	WebCache_0
WebSite_5	5	$h(5) = 7 \cdot 5 + 4 \text{ mod } 4 = 3$	WebCache_3
WebSite_6	6	$h(6) = 7 \cdot 6 + 4 \text{ mod } 4 = 2$	WebCache_2
WebSite_7	7	$h(7) = 7 \cdot 7 + 4 \text{ mod } 4 = 1$	WebCache_1
WebSite_8	8	$h(8) = 7 \cdot 8 + 4 \text{ mod } 4 = 0$	WebCache_0
WebSite_9	9	$h(9) = 7 \cdot 9 + 4 \text{ mod } 4 = 3$	WebCache_3

Cache	Websites served by cache	#
WebCache_0	WebSite_0, WebSite_4 WebSite_8	3
WebCache_1	WebSite_3, WebSite_7	2
WebCache_2	WebSite_2, WebSite_6	2
WebCache_3	WebSite_1, WebSite_5 WebSite_9	3

Altogether, only 2 web sites out of 10 (i.e., 3 and 8) remain in their cache. This results in 80% ($\frac{10-2}{10}$) of the content that needs to be reallocated.

(c) In a second approach, consistent hashing is used to associate web sites to caches. Similar to MD5, the hash function that is used for this example produces hex numbers comprised of 4 hex digits. Hence, the range of this function is $[0000, FFFF]$. The hash values for caches and web sites are given in Table 1.2 and Table 1.1, respectively. Based on these values associate web sites to the corresponding caches.

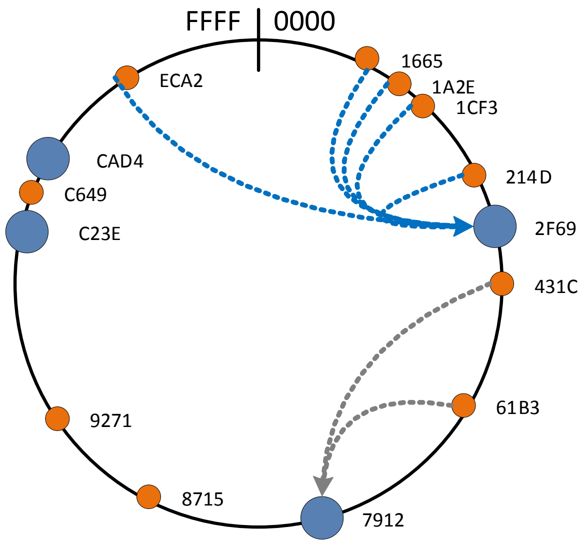
Solution:



Cache	Websites served by cache	#
WebCache_0	WebSite_0, WebSite_2, WebSite_3, WebSite_4, WebSite_6, WebSite_7, WebSite_9	7
WebCache_1	WebSite_1	1
WebCache_2	WebSite_5, WebSite_8	2

(d) How does the situation change when the new server ('WebCache_3', 3, 2F69) is added to the set of caches? Again, list the allocation of web sites to caches. What do you observe? Quantify the degree of reallocation (i.e., the percentage of web sites that need to be transferred).

Solution:



Cache	Websites served by cache	#
WebCache_0	WebSite_2, WebSite_4	2
WebCache_1	WebSite_1	1
WebCache_2	WebSite_5, WebSite_8	2
WebCache_3	WebSite_0, WebSite_3, WebSite_6, WebSite_7, WebSite_9	5

Altogether, 5 out of 10 web sites (i.e., 9, 3, 0, 6 and 7) need to be transferred to the new cache. This results in 50% ($\frac{10-5}{10}$) of the content that needs to be reallocated.

(e) Compare the uniformity of the distribution in subtask (c) and (d) by calculating the standard deviation. What do you conclude from the result? The standard deviation should be calculated with the following formula:

$$S = \sqrt{S^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2} \qquad \text{, with} \qquad \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Solution:
For subtask (c):

$$\bar{X}_c = \frac{1}{3}\cdot 10 \approx 3.33$$

$$S_c = \sqrt{\frac{1}{2}\cdot((7-3.33)^2 + (2-3.33)^2 + (1-3.33)^2)}$$

$$S_c = \sqrt{\frac{1}{2}\cdot((3.67)^2 + (-1.33)^2 + (-2.33)^2)}$$

$$S_c = \sqrt{\frac{1}{2}\cdot(13.46 + 1.76 + 5.42)}$$

$$S_c \approx 3.21$$

For subtask (d):

$$\bar{X}_d = \frac{1}{4}\cdot 10 = 2.5$$

$$S_d = \sqrt{\frac{1}{3}\cdot((5-2.5)^2 + (2-2.5)^2 + (2-2.5)^2 + (1-2.5)^2)}$$

$$S_d = \sqrt{\frac{1}{3}\cdot((2.5)^2 + (-0.5)^2 + (-0.5)^2 + (-1.5)^2)}$$

$$S_d = \sqrt{\frac{1}{3}\cdot(6.25 + 0.25 + 0.25 + 2.25)}$$

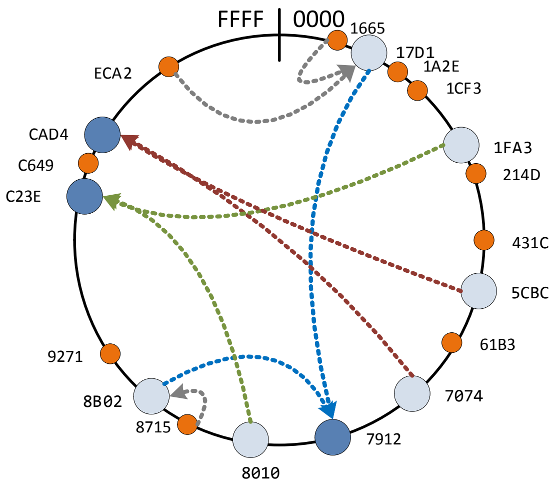
$$S_d \approx 1.73$$

The result shows that with increasing number of caches the standard deviation decreases. This indicates that the distribution of elements/web sites among caches gets more uniform with higher number of available caches.

(f) As we saw, it is possible to have a non-uniform distribution of web sites between caches if there are not enough caches in the system. A possible solution to deal with this problem is to introduce the concept of "virtual nodes", i.e., replicas of web caches, where each real cache corresponds to several virtual caches in the circle. Whenever a cache is added, also a number of virtual nodes is created for the new cache, and when a cache is removed, all its virtual nodes are removed from the circle. However, all objects that are handled by a virtual cache are actually handled by the real cache they are associated with.
For the sake of this example the replication factor should be 2 (i.e., for each cache there are two additional virtual caches). The virtual caches are given in Table 1.3.
Give the allocation of web sites to virtual caches and (real) caches and calculate the standard deviation for the real caches. How does the distribution compare to the above scenarios?

Cache	Virtual Cache	Hash Value
WebCache_0	VirtualCache_0_1	8B02
	VirtualCache_0_2	17D1
WebCache_1	VirtualCache_1_1	5CBC
	VirtualCache_1_2	7074
WebCache_2	VirtualCache_2_1	1FA3
	VirtualCache_2_2	8010

Web caches and their virtual nodes.



Cache	Websites served by cache	#
WebCache_0		
VirtualCache_0_1	WebSite_8	3
VirtualCache_0_2	WebSite_3, WebSite_9	
WebCache_1	WebSite_1	
VirtualCache_1_1	WebSite_2, WebSite_7	4
VirtualCache_1_2	WebSite_4	
WebCache_2	WebSite_5	
VirtualCache_2_1	WebSite_0, WebSite_6	3
VirtualCache_2_2		

$$\bar{X}_f = \frac{1}{3} \cdot 10 \approx 3.3$$

$$S_f = \sqrt{\frac{1}{2} \cdot ((3-3.3)^2 + (4-3.3)^2 + (3-3.3)^2)}$$

$$S_f = \sqrt{\frac{1}{2} \cdot ((-0.3)^2 + (0.7)^2 + (-0.3)^2}$$

$$S_f = \sqrt{\frac{1}{2} \cdot (0.09 + 0.49 + 0.09)}$$

$$S_f \approx 0.58$$

Due to the (virtually) increased number of caches the probability for an even distribution also increases. Compared to the above scenarios the distribution here is better.

EXERCISE 2 MapReduce Algorithms

Formulate the following algorithms for MapReduce. Explain how the input is mapped into (*key, value*) pairs by the map stage, i.e., specify what is the key and what is the associated value in each pair and how the key(s) and value(s) are computed. Also explain how the (key, value) pairs produced by the map stage are processed by the reduce stage to get the final result. Please use the below scheme:

MAP: <What the map function does>

Input: <define input>

Output: <define output>

REDUCE: <What the reduce function does>

Input: <define input>

Output: <define output>

(a) Word count: Count the frequency of word apperances in a set of documents.

Solution:

MAP: Each map gets a document. The mapper generates many key/value pairs for the appearance of a word, i.e., (word, "1")

Input: `(fileName, fileContent)` , where `fileName` is the *key* and `fileContent` is the *value*

Output: `(word, wordApparence)`-pairs, where `word` is the *key*, and `wordApparence` is the *value*

REDUCE: Combines the values for a key and computes the sum

Input: `(word, List<wordApparence>)`

Output: `(word, sum(wordApparence))`

(b) Search for a pattern: Data is a set of files containing lines of text. Output the file names that contain this pattern.

Solution:

MAP: Given (fileName, fileContent) and "pattern", if "text" matches "pattern" output (fileName, _)

Input: `(fileName, fileContent)` , where `fileName` is the *key* and `fileContent` is the *value*

Output: `(fileName, _)`-pairs, where `fileName` is the *key* and `value` is not relevant

REDUCE: Identity function. Data is not changed

Input: `(fileName, _)`

Output: `(fileName, _)`

(c) Sorting: Given a set of files, one value per line, sort the values. Assume that all values are unique.

Solution:

MAP: Identity function for value. Output is the value as key

Input: `(filename, fileContent)`-pairs, where `fileName` is the *key* and `fileContent` is the *value*.

Output: `(fileContentLine, _)`, where `fileContentLine` is the *key* and *value* is not relevant.

REDUCE: Identity function. Data is not changed.

Input: `(fileContentLine, _)`

Output: `(fileContentLine, _)`

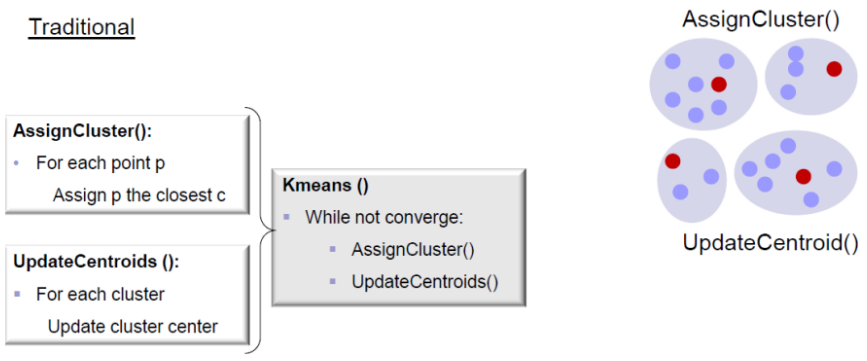
Takes advantage of reducer properties: The (key, value) pairs are processed in order by key; reducers are themselves ordered. The (key, value) pairs from mappers are sent to a particular reducer based on hash(key). Must pick the hash function for your data such that $k_1 < k_2 \Rightarrow hash(k_1) < hash(k_2)$.

EXERCISE 3 K-means in MapReduce

K-means clustering aims to partition n observations into K clusters in which each observation belongs to the cluster with the nearest mean. The basic procedure for K-means clustering is:

- Partition $\{x_1, \ldots, x_n\}$ into K clusters, where K is predefined.
- Initialization: Specify the initial cluster centers (centroids), i.e., K.
- Iteration until no change:
  - For each object x_i
    - Calculate the distance between x_i and the K centroids
    - (Re)assign x_i to the cluster whose centroid is closest to x_i
  - Update centroids based on current assignment

We want to develop the MapReduce formulation for K-means clustering. Bellow you can see a traditional implementation of K-means, where p is the observation x_i and c is the centroid.



(a) The MapReduce formulation of K-means needs a driver or wrapper around the normal execution framework. What is the reason for this?

Solution:
Because of the iterations required in the K-means procedure, we need to loop MapReduce.

(b) The MapReduce K-means algorithm can be formulated using the following components:

- Driver or wrapper
- Mapper
- Reducer

Consider that a single file contains the predefined cluster centers K and the data points are distributed in several files. Provide a short description defining the task performed by each component of MapReduce K-means and define their input and output.

Solution:

- Driver or wrapper
  - Runs multiple iteration jobs using mapper+reducer
- Mapper

  - Task: Assign each point to closest centroids
  - Configure: A single file containing cluster centers
  - Input: Input data points
  - Output: (cluster id, data id)
- Reduces
  - Task: Update each centroid with its new location
  - Input:(cluster id, List<data id>)
  - Output: (cluster id, cluster centroid)

(c) What characteristic of K-means could cause a large computation overhead?

Solution:
Since we are looping MapReduce, this means constant access to the IO (Hard-drive), if the K-means requires a large amount of iterations this could be a major source for time delay.