TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Exercises for *Foundations in Data Engineering*, WiSe 23/24

Alexander Beischl, Maximilian Reif (i3fde@in.tum.de)
http://db.in.tum.de/teaching/ws2324/foundationsde

Sheet Nr. 12

Exercise 1 Regarding the CAP theorem, it was mentioned in the lecture, that only two of the three "wishes" (consistency, availability, partition tolerance) can be met simultaneously.

However, which of the three combinations CA, CP, AP are very similar?

Solution:
The CAP theorem describes the problem that in distributed systems, only two of the three important properties **C**onsistency, **A**availability, and **P**artition tolerance can be guaranteed. Therefore, there should be three kinds of systems:

CA: Consistend and available, but not partition tolerant.

CP: Consistent and patition tolerant, but not available.

AP: Available and partition tolerant, but not consistent.

As Daniel Abadi describes in his Blog article, there is no practical difference between CP and CA systems. A CP System dismisses availability in case the network is partitioned (according to the definition, it could also never be available, but that would be a useless system). CA systems per definition can not tolerate network partitions. In reality, that means that CA systems are not available in case of network partitioning. This is the only way to ensure consistency.
Therefore, there are only two kinds of systems: CP/CA and AP. They can be distinguished by how they react to network partitions.

Exercise 2 Which guarantees that ACID databases provide can noSQL databases give up to achieve higher performance? For every trick: How does it work and why is it faster?

Solution:

- *Atomicity*: Do not provide the capability to roll back.
- *Consistency*: Propagate changes slower, smaller consistency units: object wise, group of objects
- *Isolation*: Don't separate users, dirty reads, write are possible, one transaction can see something that another transaction later reverts.
- *Durability*: Batch writes to disk, prolong write until after transaction commit. Thus changes can get lost.

Exercise 3 For the following cases: For which database guarantees would it make sense to give them up to gain performance without jeopardizing the usefulness of the whole process?

1. All cars of manufacturer X send traffic information to X's servers. X uses it to predict traffic jams.

   Solution: Durability, Availability, Consistency, Atomicity

2. All cars of X have an emergency call function to connect to the company's head quarter and also send location information.

   Solution: Don't give up guarantees. But how to scale this? Partition!

3. Telecom provider Y saves all call statistics in a database in order to bill clients.

   Solution: May opt to lose call data, but should not use too many.

4. University U collects weather information from weather stations around the country.

   Solution: Better not lose data for scientific credibility.

5. Small company C stores lists of parts in a database.

   Solution: Dataset is small, no need to give up guarantees.

Exercise 4 Imagine a document database that is distributed over many computers. Every document has a key that uniquely identifies it. Whenever a new document is added to the database, this is done using a key and the document content. Documents can be retrieved with the key.

1. Can you find real-world examples of such a database?
   Solution:

   - Napster
   - mongoDB
   - Cassandra

2. The documents in the system should be distributed over n participating computers so that roughly the same number of documents reside on every machine. Furthermore, it should be quite simple to determine from a given key on which machine the corresponding document resides, e.g. using hashing. Describe a strategy to accomplish this.
   Solution:
   Hash of $key \, mod \, n$ determines the machine the document should reside on.

3. How can the above strategy be modified so that computers can be added to the system while documents are still distributed evenly?
   Solution:
   Rehash with $mod \, n + 1$ when a new machines joins. Redistribute data.

4. Describe a strategy in this system to find the document that corresponds to a given key.
   Solution:
   Either

   - Use lookup table on central server
   - Ask all nodes

5. How can this strategy be changed, so that only very few documents need to be moved when a machines joins the system?

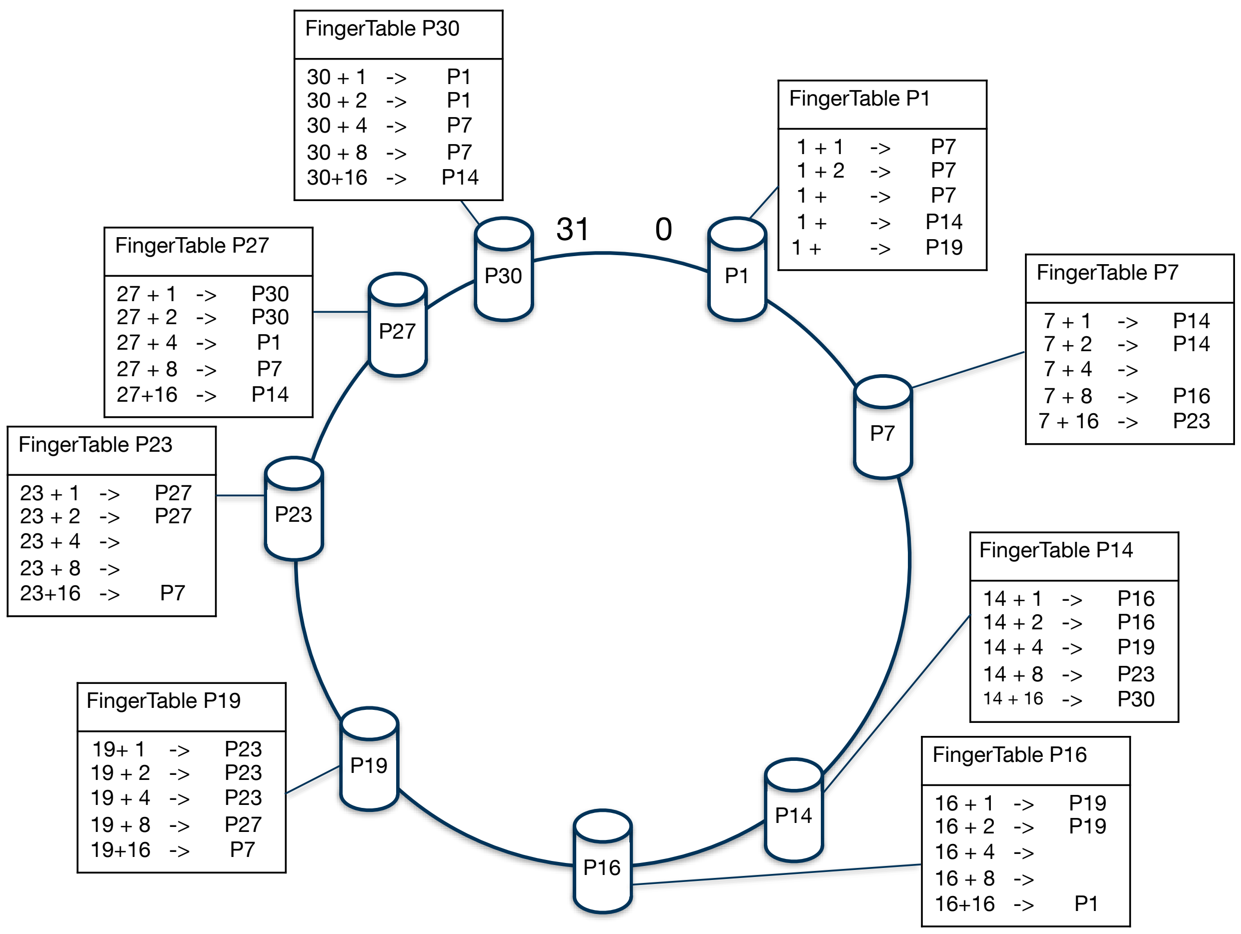
Solution: CHORD: Hash to a ring, new machines take from adjacent machines on the ring.

6. Can you devise a strategy in which no node is a single point of failure, but that does not need to send network request to all participating machines?

   Solution:
   CHORD, CAN

Exercise 5 Figure 1 shows a Chord network with 8 participating peers (compute nodes).

1. Complete the missing entries in the finger tables.
2. Search key 24 starting at peer 30.
3. Show that the chord search scheme always results in a maximum of $log(n)$ steps, where n is the ring size. (Hint: Have a look at the steps needed to find key 24.)



FingerTable P30		
30 + 1	->	P1
30 + 2	->	P1
30 + 4	->	P7
30 + 8	->	P7
30+16	->	P14

FingerTable P1		
1 + 1	->	P7
1 + 2	->	P7
1 +	->	P7
1 +	->	P14
1 +	->	P19

FingerTable P27		
27 + 1	->	P30
27 + 2	->	P30
27 + 4	->	P1
27 + 8	->	P7
27+16	->	P14

FingerTable P7		
7 + 1	->	P14
7 + 2	->	P14
7 + 4	->	
7 + 8	->	P16
7 + 16	->	P23

FingerTable P23		
23 + 1	->	P27
23 + 2	->	P27
23 + 4	->	
23 + 8	->	
23+16	->	P7

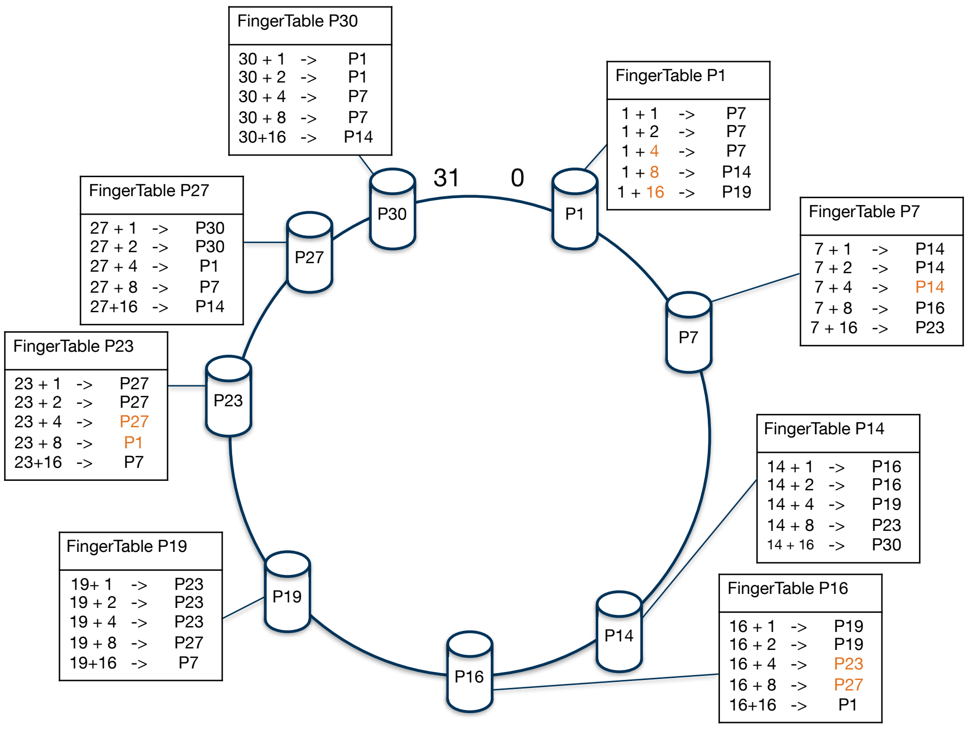
FingerTable P14		
14 + 1	->	P16
14 + 2	->	P16
14 + 4	->	P19
14 + 8	->	P23
14 + 16	->	P30

FingerTable P19		
19+ 1	->	P23
19 + 2	->	P23
19 + 4	->	P23
19 + 8	->	P27
19+16	->	P7

FingerTable P16		
16 + 1	->	P19
16 + 2	->	P19
16 + 4	->	
16 + 8	->	
16+16	->	P1

Figure 1: Example of a Chord overlay-network.

Solution:
Task 1:



FingerTable P30		
30 + 1	->	P1
30 + 2	->	P1
30 + 4	->	P7
30 + 8	->	P7
30+16	->	P14

FingerTable P1		
1 + 1	->	P7
1 + 2	->	P7
1 + 4	->	P7
1 + 8	->	P14
1 + 16	->	P19

FingerTable P7		
7 + 1	->	P14
7 + 2	->	P14
7 + 4	->	P14
7 + 8	->	P16
7 + 16	->	P23

FingerTable P14		
14 + 1	->	P16
14 + 2	->	P16
14 + 4	->	P19
14 + 8	->	P23
14 + 16	->	P30

FingerTable P16		
16 + 1	->	P19
16 + 2	->	P19
16 + 4	->	P23
16 + 8	->	P27
16+16	->	P1

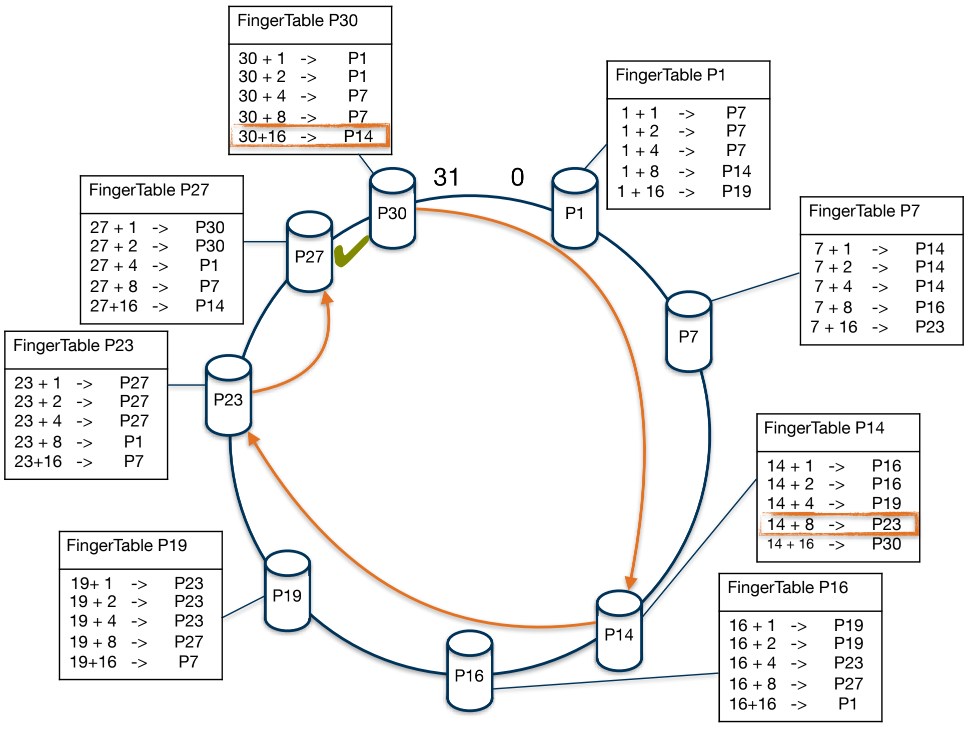
FingerTable P19		
19+ 1	->	P23
19 + 2	->	P23
19 + 4	->	P23
19 + 8	->	P27
19+16	->	P7

FingerTable P23		
23 + 1	->	P27
23 + 2	->	P27
23 + 4	->	P27
23 + 8	->	P1
23+16	->	P7

FingerTable P27		
27 + 1	->	P30
27 + 2	->	P30
27 + 4	->	P1
27 + 8	->	P7
27+16	->	P14

Figure 2: Chord-overlaynetwork with complete finger tables.

Task 2:



FingerTable P30		
30 + 1	->	P1
30 + 2	->	P1
30 + 4	->	P7
30 + 8	->	P7
30+16	->	P14

FingerTable P1		
1 + 1	->	P7
1 + 2	->	P7
1 + 4	->	P7
1 + 8	->	P14
1 + 16	->	P19

FingerTable P7		
7 + 1	->	P14
7 + 2	->	P14
7 + 4	->	P14
7 + 8	->	P16
7 + 16	->	P23

FingerTable P14		
14 + 1	->	P16
14 + 2	->	P16
14 + 4	->	P19
14 + 8	->	P23
14 + 16	->	P30

FingerTable P16		
16 + 1	->	P19
16 + 2	->	P19
16 + 4	->	P23
16 + 8	->	P27
16+16	->	P1

FingerTable P19		
19+ 1	->	P23
19 + 2	->	P23
19 + 4	->	P23
19 + 8	->	P27
19+16	->	P7

FingerTable P23		
23 + 1	->	P27
23 + 2	->	P27
23 + 4	->	P27
23 + 8	->	P1
23+16	->	P7

FingerTable P27		
27 + 1	->	P30
27 + 2	->	P30
27 + 4	->	P1
27 + 8	->	P7
27+16	->	P14

Figure 3: Searching key 24 starting at peer 30.

Task 3:
Why is searching a peer in the Chord-overlaynetwork by using finger tables performed with in a logarithmic number of steps of the size of the ring size?

The logarithmic number of steps for searching a key is achieved by the design of the *finger table*.
Each peer is responsible for all keys larger than the previous peer's number until its own peer-number p. Therefore, our peer 30 is responsible for the keys $[28; 30]$. To search efficiently for keys managed by another peer, each peer has a finger table containing the peers responsible for the keys. Instead of storing an entry for each key in the finger table, only the keys with the following values are stored with its responsible peers:

$$k_0 : p + 2^0$$
$$k_1 : p + 2^1$$
$$k_2 : p + 2^2$$
$$k_3 : p + 2^3$$

Therefore, the distance between the keys always increases by the factor 2.

When searching for a certain key k, the peer checks its finger table for the largest peer l, whose number is smaller than k. Then the search is directed to peer l, knowing we can narrow the search by using its finger table, because it will contain the peers responsible for larger keys. Therefore, with each hop to another peer the distance to k's peer is halved due to the entries in the finger table.
This procedure is repeated until the finger table only contains peers, whose value is larger than k. Then, the next peer is responsible for k and we can directly hop to this peer finding our key.

In result, for searching a key in the Chord-overlaynetwork by using finger table the logarithmic number of steps is achieved by decreasing the distance to the final peer by at least the factor two with each hop.