Databases

- Explain the main reasons for why NoSQL data stores appeared.

Increasing number of concurrent users, many different types of data. Frequent schema changes or no schema, changing usage.

- List and describe the main characteristics of NoSQL data stores.

Non-relational database management system, relaxes some requirements of relational database management systems to gain efficiency and scalability.Ability to scale horizontally over many servers with high performance, availability and fault tolerance. This is achieved by giving up ACID properties: **A**tomicity, **C**onsistency preservation, **I**solation and **D**urability.

- Explain the difference between ACID and BASE properties.

Basically **A**vailable: system always available, **S**oft state: distributed data does not need to be in a consistent state at all times, **E**ventually consistent: state will become consistent eventually. BASE will always be available but inconsistent, ACID will not always be available but guarantees consistency.

- Explain the differences between vertical and horizontal scalability.

Data scalability - System can handle growing amount of data without performance loss

Read scalability - System can handle increasing number of read operations without performance loss

Write scalability - -||- but write

Vertical scalability is to add more resources to a server e.g. bigger computers

Horizontal scalability is to add more nodes to a distributed system e.g. more computers

- Discuss the trade-off between consistency and availability in a distribute data store setting.

In a distribute data store setting we always need partition tolerance which means that we need to choose between availability and consistency according to the CAP-theorem. If availability is chosen higher scalability and performance can be achieved at the tradeoff of having inconsistencies. This is fine if our system is okay with some inconsistencies, but this must often be handled in the code and does not work for systems like stores where you can’t

sell the last item twice. The opposite is true for choosing consistency, e.g. slower but consistent system.

- Discuss different consistency models and why they are needed.

Different consistency models are systems that dictate the rules for how the memory will behave when being read, written to or updated. Different consistency models are needed to suit applications with either consistency or increased performance depending on the needs of the application.

- Explain the CAP theorem.

The CAP-theorem dictates that in a distributed system with data replication only 2 out of the following 3 properties can be guaranteeed: **C**onsistency, **A**vailability and **P**artition Tolerance. This means that in a system without partitions both consistency and availability can be guaranteed, but in a system where we need partition tolerance we need to prioritize between if we need consistency or availability.

- List and describe the main characteristics and applications of NoSQL data stores according to their data models.

Key-Value stores: when values need to be accessed only via keys. e.g Web session info, shopping cart data, user profiles.

Document model: when we have items of similar nature but slightly different structure. e.g. blogging platforms, other content management systems, event logging.

Wide-column stores: Similar to document stores, analytics scenarios: web analytics, personalized search, inbox search.

Graph database stores: For complex networks, location based services, recommendations and fraud detection.

Paralell computing

- PAR-Q1: Define the following technical terms:
  (Be thorough and general. An example is not a definition.)
  - Cluster (in high-performance resp. big-data computing)

A cluster is a large aggregate of individual computers. This cluster needs scalable parallel algorithms, it needs to exploit multiple levels of parallelism and needs fault tolerance. Both HPC and big-data use clusters.

  - Parallel work (of a parallel algorithm)

The *parallel work*

$$w_A(n) = \sum_{i=1}^{t_A(n)} p_i(n)$$

e.g. counting the executed constant-time operations across all participating processors but not any idle time. The total number of performed elementary operations.

  - Parallel speed-up

the factor by how much faster we can solve a problem with p processors than with 1 processor, usually in range (0…p)

  - Communication latency (for sending a message from node Pi to node Pj)

time $t_{msg}(n)$ = sender overhead + latency + receiver overhead + n/bandwidth

$=: \; t_{startup} + n \cdot t_{transfer}$

Assumption: network not overloaded; no conflicts occur at routing

$t_{startup}$ = startup time (time to send a 0-byte message)

accounts for hardware and software overhead.

$t_{transfer}$ = transfer rate, send time per word sent.

c. depends on the network bandwidth.

- Temporal data locality

Re-accessing the same data element multiple times within a short time interval

- Dynamic task scheduling

Each newly created task is dispatched at runtime to an available worker processor. e.g. the tasks are assigned to workers dynamically during runtime by a scheduler.

- PAR-Q2: Explain the following parallel algorithmic paradigm: Parallel Divide-and-Conquer.

Parallel divide-and-conquer takes a problem P and if it is not trivial it divides it into smaller problems p1 … pk and then solves this problem P recursivly. These tasks p1 … pk may also be non-trivial which causes another division until the resulting problems are trivial. After these the solutions are combined recursively until the problem P is solved. Both these recursive calls and the divide and combine phases can be parallized.

- PAR-Q3: Discuss the performance effects of using large vs. small packet sizes in streaming.

Larger packet sizes will be more efficient size more data is streamed with less overhead, however the sacrifice of this is greater latency. Small packet sizes has more total overhead time but the time for a small packet to arrive is smaller. If many tasks are to be performed on data, bigger packet sizes will be better size data will not need to be moved as often, opposite is true with smaller packet sizes.

- PAR-Q4: Why should servers (cluster nodes) in datacenters that are running I/O-intensive tasks (such as file/database accesses) get (many) more tasks to run than they have cores?

To keep cores and threads busy and overlap waiting times. I/O accesses has a lot of waiting times which means that with many many tasks the nodes can better exploit parallelism by always having tasks to perform during this waiting time.

- PAR-Q5: In skeleton programming, which skeleton will you need to use for computing the maximum element in a large array? Sketch the resulting pseudocode (explain your code).

```
function max(a,b) {
        if (a > b) {
                return a
        }
        return b
}
return reduce(max)(data)
```

- PAR-Q6: Describe the advantages/strengths and the drawbacks/limitations of high-level parallel programming using algorithmic skeletons.

+ Abstraction, hiding complexity
+ Parallelization for free
+ Easy to analyze and transform
- Requires complete understanding and rewriting of a computation
- Available skeletion set does not always fit
- May lose efficiency compared to manual parallelization

- PAR-Q7: Derive Amdahl's Law and give its interpretation.

Amdahl's Law: Upper bound on Speedup

Consider execution (trace) of parallel algorithm A:

sequential part A^s where only 1 processor is active

parallel part A^p that can be sped up perfectly by p processors

$\rightarrow$ total work $w_A(n) = w_{A^s}(n) + w_{A^p}(n)$, time $T = T_{A^s} + \frac{T_{A^p}}{p}$,

Amdahl's Law

If the sequential part of A is a *fixed* fraction of the total work irrespective of the problem size n, that is, if there is a constant β with

$$\beta = \frac{w_{A^s}(n)}{w_A(n)} \leq 1$$

the relative speedup of A with p processors is limited by

$$\frac{p}{\beta p + (1-\beta)} < 1/\beta$$

- PAR-Q8: What is the difference between relative and absolute parallel speed-up? Which of these is expected to be higher?
-

The relative speedup describes how well the parallel algorithm utilizes p processors(scalability which the absolute speedup expresses how much can be gained over the best sequential implementation by parallelization. Relative speedup will be higher

- PAR-Q9: The PRAM (Parallel Random Access Machine) computation model has the simplest-possible parallel cost model. Which aspects of a real-world parallel computer does it represent, and which aspects does it abstract from?

It represents pure parallelism and abstracts from scheduling overhead. This is good for early analysis since if an algorithm does not scale well under a PRAM model it will never scale well.

- PAR-Q10: Which property of streaming computations makes it possible to overlap computation with data transfer?
-

FIFO buffering feature since it allows correct order of execution. This allows pipelining since execution of data-producing and data-consuming tasks can overlap in time as long as it runs on different data elements.

MapReduce

- MR-Q1: A MapReduce computation should process 12.8 TB of data in a distributed file with block (shard) size 64MB. How many mapper tasks will be created, by default? (Hint: 1 TB (Terabyte) = 10^12 byte)

12.8*10^12/(64*10^3)= 200 000 000 mapper tasks

- MR-Q2: Discuss the design decision to offer just one MapReduce construct that covers both mapping, shuffle+sort and reducing. Wouldn't it be easier to provide one separate construct for each phase instead? What would be the performance implications of such a design operating on distributed files?

The complexity would increase if we provided one separate construct for each phase, it would be more choices to make. There would be more parts that had to communicate, thus more things that could go wrong.

But it's a trade-off, as it would make it more flexible and it might be possible to optimize it even more.

- MR-Q4: Consider the local reduction performed by a Combiner: Why should the user-defined Reduce function be associative and commutative? Give examples for reduce functions that are associative and commutative, and such that are not.

If a function is commutative, it means that it doesn't matter in what order it's performed. If a function is associative, it means that it can be rearranged and re-grouped and still give the same result.

If the user-defined Reduce function is associative and commutative, it makes it possible to group and do some parts of the Reduce function locally in the mapper, that otherwise would be done on by the Reducer.

What does commutative mean?
The property of moving variables around in an equation.
E.g. 1 + 2 + x = 1 + x + 2
What does associative mean?
The property of grouping.
E.g. (a + b) + c = a + (b + c)

Associative and commutative:

When you want to sum something

- MR-Q8: Sometimes, workers might be temporarily slowed down (e.g. repeated disk read errors) without being broken. Such workers could delay the completion of an entire MapReduce computation considerably. How could the master speed up the overall MapReduce processing if it observes that some worker is late?
-

By marking the worker as dead and reassigning the tasks to other workers.

- Spark-Q1: Why can MapReduce emulate any distributed computation?
-

Because the steps can be chained, but this comes with limitations such as unnecessary disk I/O, bad data localit and data blocks being read multiple times from disk.

- Spark-Q2: For a Spark program consisting of 2 subsequent Map computations, show how Spark execution differs from Hadoop/Mapreduce execution.

In both the inital data is read from disk and then mapped. However after this spark will store it in memory while mapreduce will write it to disk and then have to read it again, spark reading it from memory is much faster.

Cluster Resource Management

- - YARN-Q1: Why is it reasonable that Application Masters can request and return resources dynamically from/to the Resource Manager (within the maximum lease initially granted to their job by the RM), instead of requesting their maximum lease on all nodes immediately and keeping it throughout the job's lifetime? Contrast this mechanism to the resource allocation performed by batch queuing systems for clusters.
  To be able to better utilize the resources for the job. The benefits of this are many. This is just another solution to the problem of resource allocation also solved by batch queuing, where resources are are offered when they become available.
- - YARN-Q2: Explain why the Node Manager's tasks are better performed in a daemon process controlled by the RM and not under the control of the framework-specific application.
  To be able to deallocate resources as needed during the job, on request of the resource manager. Also does tasks specific things to the node, i.e. monitoring node health etc.

MAP REDUCE PHASES

1) Record reader
- Parse an input file block from stdin into key-value pairs.

2) Mapper
- Applies user-defined function to each element
- Produces intermediate key-value elements.
Key: Index for grouping of data
Value: Data to be forwarded to reducer
- Buffered in memory

3) Combiner
- An optional local reducer
- Apply user-provided function to aggregate values in the intermediate elements of one mapper task

4) Partitioner
- Splits intermediate elements from Mapper/Combiner into blocks
- Writes blocks to local file system

5) Shuffle and sort
- Download needed files
- Sort the received (key, value) pairs into on list
- Pair equivalant keys

6) Reducer
- Run user-defined reduce function
- One per key grouping
- E.g. aggregate, filter and combine data
- Write final result to stdout and then to a file in HDFS