Examples of exam question types

NoSQL data stores and techniques

- Nov 2019: Give and explain 4 V's (big data properties) and give an example for each. (2p)
  - Volume: size of data
  - Variety: type and nature of data
  - Velocity: speed of generation and processing of data
  - Veracity: quality of the data
- Nov 2019: Describe difference between vertical scalability and horizontal scalability(1p)
  - Vertical scalability: Add resources to a server (e.g., more CPUs, more memory, more or bigger disks)
  - Horizontal scalability: Add nodes (more computers) to a distributed system
- Nov 2019: Describe the differnce between read scalability and write scalability.(1p)
  - Read Scalability: system can handle increasing numbers of read operations without losing performance.
  - Write scalability: system can handle increasing numbers of write operations without losing performance
- Nov 2019: Describe a concrete application/use case for which data scalability is important. Note that "scalability" is a key word in this question; it is not enough if your application/use case simply has to do with a huge amount of data.(1p)
  - An example of such an application is a user analysis and product recommendation application in an online shopping portal. The reason is that the number of users of the portal may increase, and so may the number of products or, for instance, the number of product reviews written by users. As a consequence, the amount of data that would have to be analyzed by the application may also increase. Of course, it would be undesirable if the increased amount of data has a negative impact on the performance of the application (or even of the shopping portal as a whole).
- Nov 2019: Consider the following key-value database which contains three key-value pairs where the keys are user IDs and the values consist of a user name and an array of IDs of users that the current user likes (for instance, Alice likes Bob and Charlie).

  "alice_in-se" → "Alice, [bob95 charlie]"

  "bob95" → "Bob, [charlie]"

  "charlie" → "Charlie, [ ]"

  Describe how the types of queries typically implemented in a key-value store can be used to retrieve the user IDs of users named Alice.(1p)

- The only way to query a key-value database is by get(key) operations. Since the keys in the given database are the user IDs and we cannot assume to know the IDs of the users named Alice, we have to go through all the keys of the database. That is, for every userID, we have to request the corresponding value by doing get(userID), then we have to look into the value to check whether the name related part of the value is the string "Alice", and checking the value part with name 'Alice' is something that would have to be implemented in an application program (rather than expressed as queries for the database system).

- Nov 2019: Describe how the key-value database can be changed/extended such that retrieving the user IDs of users named Alice is more efficient.(1p)

  – A possible extension of the given database may consist of two parts: First, we extend the keys of the existing key-value pairs by prefixing them with the string "UserID:". Second, we may add an additional key-value pair for each unique user name such that the key of such a key-value pair is the corresponding user name, prefixed with the string "UserName:", and the value is an array of the user IDs of all users that have this name. For instance, for the given example database, this extension may look as follows.

"UserID:alice_in_se" → "Alice, [bob95 charlie]"

"UserID:bob95" → "Bob, [charlie]"

"UserID:charlie" → "Charlie, [ ]"

"UserName:Alice" → "[alice_in_se]"

"UserName:Bob" → "[bob95]"

"UserName:Charlie" → "[charlie]"

  Given this extension, we may now do get ("UserName:Alice"), which would give us the value that contains the array with the user IDs of all users named Alice (i.e., alice_in_se, in our current example database)

- Nov 2019: Identify two differences between the key-value database model and the document database model that was introduced in class. (1p)

- Example Solution 1:

  – In the key-value database model, users can choose what the keys are, whereas the document identifiers in the document model are typically system-generated.

  – The key-value pairs in the key-value database model cannot be grouped whereas, in a document database, we can group key-value pairs into separate documents; moreover, some forms of document databases allow us to even group these documents further, namely into so called collections or domains.

- Example Solution 2:

  – While the documents in a document database have an internal structure that is clearly defined (and, thus, can be operated on by the DBMS; for instance, to create indexes), the same is not the case for the values in a key-value database where any possible internal structure of such values is opaque from a DBMS perspective.

  – In the key-value model, access to multiple database entries (key-value pairs, in this case) requires separate requests. In the document model, on the other hand, multiple database entries (documents, in this case) may be retrieved in a single request.

- Nov 2019: Specify what the BASE properties are. (Simply writing down the names of these properties is not enough and does not earn you any points) (2p)
  - Basically Available: system available whenever accessed, even if parts of it unavailable
  - Soft state: the distributed data does not need to be in a consistent state at all times
  - Eventually consistent: state will become consistent after a certain period of time
- Explain the main reasons for why NoSQL data stores appeared.
  - Increasing numbers of concurrent users/clients
    - tens of thousands, perhaps millions
    - globally distributed
    - expectations: consistently high performance and 24/7 availability (no downtime)
  - Different types of data
    - huge amounts (generated by users and devices)
    - data from different sources together
    - frequent schema changes or no schema at all
    - semi-structured and unstructured data
  - Usage may change rapidly and unpredictably
- List and describe the main characteristics of NoSQL data stores.
  - Ability to scale horizontally over many commodity servers with high performance, availability, and fault tolerance
    - and by partitioning and replication of data
  - Non-relational data model, no requirements for schemas
    - data model limitations make partitioning effective
- Explain the difference between ACID and BASE properties.
  - ACID:
    - Atomicity: Everything in a transaction need to excute successfully otherwise the whole transaction will not be excuted.
    - Consistency preservation: A transaction cannot leave the database in an inconsistent state.
    - Isolation: Transactions cannot interfere with each other and can be excuted concurrently.
    - Durability:Completed transactions persist, even when servers crash or restart the transactions remain committed.
  - BASE:
    - Basically Available: system available whenever accessed, even if parts of it unavailable
    - Soft state:the distributed data does not need to be in a consistent state at all times
    - Eventually consistent:state will become consistent after a certain period of time
  - BASE properties suitable for applications for which some inconsistency may be acceptable
- Discuss the trade-off between consistency and availability in a distribute data store setting.
  - When choosing consistency over availability, the system will return an error or a time out if particular information cannot be guaranteed to be up to date due to network partitioning. When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.

- Discuss different consistency models and why they are needed.
  - Strong consistency: after an update completes, every subsequent access will return the updated value, usually needed because
  - Weak consistency: no guarantee that all subsequent accesses will return the updated value
    - A type of Weak Consistency is eventual Consistency in which if no new updates are made, eventually all accesses will return the last updated value

- Explain the CAP theorem.
  Only 2 of the following 3 properties can be guaranteed at the same time in a distributed system with data replication
  - Consistency: the same copy of a replicated data item is visible from all nodes that have this item
  - Availability: all requests for a data item will be answered
  - Partition Tolerance: system continues to operate even if it gets partitioned into isolated sets of nodes

- Explain the differences between vertical and horizontal scalability.
  - Vertical scalability: Add resources to a server (e.g., more CPUs, more memory, more or bigger disks)
  - Horizontal scalability: Add nodes (more computers) to a distributed system

- List and describe the main characteristics and applications of NoSQL data stores according to their data models.
  - Key-value model:
    Characteristics:
    - Database is simply a set of key-value pairs
      keys are unique - values of arbitrary data types
    - Values are opaque to the system
    - Only CRUD (create, read, update, delete) operations in terms of keys
    - No support for value-related queries (no secondary index over values) because values are opaque to the system
    - Accessing multiple items requires separate requests, often not possible with one transaction
    - partition the data based on keys ("horizontal partitioning", also called "sharding") and distributed processing can be very efficient

    Applications:
    Whenever values need to be accessed only via keys:
    - Storing Web session information
    - User profiles and configuration
    - Shopping cart data
    - Caching layer that stores results of expensive operations (e.g., complex queries over an underlying database, user-tailored Web pages)
  - Document model:
  - Wide-column models:
  - Graph database models:

Parallel Computing

- Nov 2019: How a does a distributed file (in a distributed file system like HDFS) differ from a traditional file (with respect to how is it technically and logically structured, stored and accessed), and what are the two main advantages for the processing of a big-data computation over a distributed file compared to a traditional file? Be thorough!(1.5p)

  – Distributed files are split into one or more blocks and these blocks (in a distributed file system like HDFS which follows a master/slave architecture) are stored in Data Nodes. An HDFS cluster consists of a single Name Node (a pair of Name Nodes to ensure high availability), a master server that manages the file system namespace and regulates access to files by clients. On the other hand there are one or more Data Nodes that manage the data stored in each node.

  HDFS runs on top of the native file system and exposes the locations of file blocks via API.

  When writing data the client first contact the Name Node, which checks the client privileges, if it pass it will provide the client with address of DataNodes where the client can write, then it make 3 replicas in other DataNodes located on different clusters. NameNode also control file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data Nodes.

  When reading the NameNode will provide the address of the data to the client where the DataNodes will serve the read request (as well as write requests). The Data Nodes also perform block creation, deletion, and replication upon instruction from the Name Node.

  Main advantages of distributed file system are:

  Scalability: It can store large amounts of data on commodity hardware

  Fault Tolerance: Because we can replicate data on multiple nodes, in the event of failur there is another replica availaple to use.

- Nov 2019: Define and shortly explain the following terms: (1p)

  – Parallel Programming model (0.5p)
  – Algorithmic skeleton (0.5p)

  Be general and thorough. An example is not a definition, but can illustrate a definition.

  - Parallel Programming model:

    System-software-enabled programmer's view of the underlying hardware

    Abstracts from details of the underlying architecture, e.g. network topology

    Focuses on a few characteristic properties, e.g. memory model → Portability of algorithms/programs across a family of parallel architectures

  - Algorithmic Skeleton:

Reusable, parameterizable software components with well defined semantics for which efficient parallel implementations may be available. like map, reduce, pipe and scan.

- Nov 2019: Describe (by an annotated drawing and text) the hardware structure of a modern hybrid clusters (used for both HPC and distributed parallel bid-data processing). In particular, specify and explain their memory structure and how the different parts are connected to each other. (1p)
  - Answer on page 9, lecture 6
- 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)
    Aggregation of many computers/servers connected together as a single unit such that it can be controlled and scheduled to work on similar task
  - Parallel work (of a parallel algorithm)
    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)
    The time interval for sending a message from node Pi to node Pj where high latency favors larger transfer block sizes (cache lines , memory pages, file blocks, messages) for amortization over many subsequent accesses
  - Temporal data locality
    The re-access of the same data element multiple times within a short time interval
  - Dynamic task scheduling
    Task scheduling method in which the priorities are calculated during the execution of the program
- PAR-Q2: Explain the following parallel algorithmic paradigm: Parallel Divide-and-Conquer.
  - If given problem instance P is trivial , solve it directly. Otherwise:
  - Divide: Decompose problem instance P into one or several smaller independent instances of the same problem, $P_1, ..., P_k$
  - For each i: solve P_i by recursion.
  - Combine the solutions of the P_i into an overall solution for P

  Where:
  - Recursive calls can be done in parallel.
  - Divide and combine phases can parallized when possible
  - Switch to sequential divide and conquer when enough parallel tasks have been created.
- PAR-Q3: Discuss the performance effects of using large vs. small packet sizes in streaming.
  - When the packet size is small the throughput (operations per second) will be small and might not utilize the available memory where as large packet size might overflow the memory and also cause a delayed streaming
- 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?

- It helps with load balancing

- 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).

```
# instantiating a skeleton template in a user-provided function without parallelism concerns
float max(int a, int b)
{
  return (a > b) ? a : b;
}

# Using the user-provided function in a Reduce skeleton
auto array_max = Reduce(max);

# Excuting the code
array_max(array);
```

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

  Advantages:

  - Abstraction, hiding complexity (parallelism and low level programming)
  - Parallelization for free
  - Easier to analyze and transform

  Advantage/Drawback:

  - Enforces structuring, restricted set of constructs

  Drawbacks:

  - Requires complete understanding and rewriting of a computation
  - Available skeleton set does not always fit
  - May lose some efficiency compared to manual parallelization

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

For parallel algorithm A

Where:
sequential part A^s works only on one processor
parallel part A^p can be sped up by p processors

$$\text{Total Work } w_A(n) = w_{A^s}(n) + w_{A^p}(n)$$

Total Work = number of elementary operations performed by the sequential part + number of elementary operations performed by the parallel part

$$\text{Time } T = T_{A^s} + \frac{T_{A^p}}{p}$$

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

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

The relative speed up of A with p processors is limited by:

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

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

$$\text{Absolute Speedup } S_{abs} = \frac{T_s}{T_{(p)}}$$

$$\text{Absolute Speedup } S_{rel} = \frac{T_{(1)}}{T_{(p)}}$$

Where for algorithm A:
T_s: Time to excute the best serial algorithm for a problem on one processor of the parallel machine
$T_{(1)}$: Time to excute parallel algorithm A on 1 processor
$T_{(p)}$: Time to excute parallel algorithm A on p processors

$$S_{abs} \leq S_{rel}$$

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

MapReduce & Spark

- Nov 2019: Why is it important to consider (operand) data locality when scheduling tasks (e.g., mapper tasks of a MapReduce program) to nodes in a cluster? (0.5p)
  - To amortize high access cost of lower levels (DRAM and Disk) which allows to access/re-access the data within short time intervals.
- Nov 2019: What (mathematical) properties do functions need to fulfill that are to be used in Combine or Reduce steps of MapReduce, and why? (1p)
  - Functions need to be associative and often commutative in order to be used in Combine or Reduce steps, because associativity allow us to group operations together which makes parallelizem possible, and give us consistent results

    Commutativity is important if there is a master distributing work to several processors, because the results could arrive back to the master processor in any order. Commutativity guarantees that the result will be the in the same order.
- Nov 2019: The MapReduce construct is very powerful and consists of 7 substeps as presented in the lecture. Which one(s) of these substeps may involve network I/O, and for what purpose? Be thorough! (1p)
  - Only Record Reader and Output Formatter involve network I/O

Record Reader: Parses an input file block from stdin into key value pairs that define input data records

Output Formatter: Translates the final (key,value) pair from the reduce function and writes it to stdout to a file in HDFS

- Nov 2019: Why and in what situations can it be beneficial for performance to use a Combiner in a MapReduce instance?(1p)
  - Combiner is used for data locality, key/value pairs still in cache of the same node, and for data reduction where aggregated information is often smaller.

    Recommended if there is significant repetition of intermediate keys produced by each Mapper task and applicable if the user-defined Reduce function is commutative and assocatative.

- Nov 2019: What is the RDD lineage graph and how is it used in Spark for the efficient excution of Spark programs? (1p)
  - It is a graph of all the parent RDDs of a RDD. It is built as a result of applying transformations to the RDD and creates a logical execution plan of a distributed computation that is created and expanded every time we apply a transformation on any RDD.

    It get excuted when Action is reached and that gives more flexibility to the scheduler by keeping data in memory as capacity permits and skipping unnecessary disk storage of temporary data and data replication for fault tolerance, since we could re-compute it easily using the RDD lineage graph

- Nov 2019: What is an (RDD) "transformation" in Spark? Give also one example operation of a transformation.(0.5p)
  - Transformation is a lazy and parallelizable operation that is usually a variant of Map and reading from file, like map, filter and sample.

- Nov 2019: What does collect operation in Spark do with its operand RDD?(0.5p)
  - Send all the elements of the RDD to the driver program

- Nov 2019: What is straming, in general? For what type of computations can it be suitably used? And how does Spark support streaming?(1p)
  - Streaming applies pipelining to processing of large (possibly, infinite) data streams from or to memory, network or devices, usually partitioned in fixed sized data packets, in order to overlap the processing of each packet of data in time with access of subsequent units of data and/or processing of preceding packets of data.

    like Video streaming from network to display, Surveillance camera, face recognition and Network data processing e.g. deep packet inspection

    In Spark there is extension for streaming which enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streamingreceives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

- 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)
- 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?
- MR-Q3: Reformulate the wordcount example program to use no Combiner.
- 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.
- MR-Q5: Extend the wordcount program to discard words shorter than 4 characters.
- MR-Q6: Write a wordcount program to only count all words of odd and of even length. (There are several possibilities.)
- MR-Q7: Show how to calculate a database join with MapReduce.
- 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?
- Spark-Q1: Why can MapReduce emulate any distributed computation?
- Spark-Q2: For a Spark program consisting of 2 subsequent Map computations, show how Spark execution differs from Hadoop/Mapreduce execution.
- Spark-Q3: Given is a text file containing integer numbers. Write a Spark program that adds them up.
- Spark-Q4: Write a wordcount program for Spark. (Solution proposal: see last slide in lecture 8.)
- Spark-Q5: Modify the wordcount program by only considering words with at least 4 characters.

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.
- 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.

Machine Learning for Big Data

- Nov 2019: Why Spark more suitable than MapReduce for implementing many machine learning algorithms?(1p)
  - Many machine learning algorithms are iterative in nature and if MapReduce to be used each iteration will load the data from disk and that very time consuming, while in Spark iterations load data from memory or even cache if it is small enough which is much faster

- Nov 2019: Implement in Spark (PySpark) the following k-means algorithm

1 Assign each point to a cluster at random
2 Compute the cluster centroids as the averages of the points assigned to each cluster
3 Repeat the following lies l times
4 Assign each point to the cluster with the closest centroid
5 Update the cluster centroids as the averages of the points assigned to each cluster

You can use the functions randint (A,B) which produces a random integer in the given interval, an distance (A,B) which returns the distance between two points.

```
def closestPoint(p, centers):
  bestIndex = 0
  closest = float("+inf")
  for i, center in enumerate(centers):
    tempDist = np.sum((p - center) ** 2)
    if temDist < closest:
      closest = tempDist
      bestIndex = 1
  return bestIndex

kPoints = data.takeSample(False, K, 1)
tempDist = 1.0

while tempDist > convergeDist:
  closest = data.map(lambda p: (closestPoint(p, kPoints), (p, 1)))
  pointStats = closest.reduceByKey(lambda p1_c1, p2_c2: (p1_c1[0] + p2_c2[0], p1_c1[1] + p2_c2[1]))
  newPoints = pointStats.map(lambda st: (st[0], st[1][0] / st[1][1])).collect()

  tempDist = sum(np.sum((kPoints[iK] - p) ** 2) for (iK, p) in newPoints)

  for (iK, p) in newPoints:
    kPoints(iK) = p

print("Final centers: " + str(kPoints))
```

- Implement in MapReduce or Spark a machine learning algorithm, e.g. logistic regression, k-means, EM algorithm, support vector machines, neural nets, etc. (The pseudo-code of the algorithm will be provided in the exam.)
- Logistic Regression

```
# Reading from file => getting the required data by map() =>
# persisting in memory for faster access
points = sc.textFile(...).map(...).persist()

# Intial random weights
w = np.random.ranf(size = D)

# Compute logistic regression gradient for a matrix of data points
def gradient(matrix, w):
  Y = matrix[:, 0]  # Labels
  X = matrix[:, 1:] # Coordinates
```

```
  # For each point (x,y), compute gradient function, then sum these up
  return ((1.0/(1.0 + np.exp(-Y * X.dot(w))) - 1.0) * Y * X.T).sum(1)

for i in range(iterations):
  w -= points.map(lambda m: gradient(m, w).reduce(lambda a,b: a+b))
```

- KNN

```
import numpy as np
from math import sqrt
from pyspark import SparkContext

sc = SparkContext(appName = "KNN")

# getting the data in a (y, (x1,...xn)) format where y is the class label
# and x1,...,xn are the predictive attribute values
# mydata = sc.textFile().split().map(lambda x: (x[0], (x[1:])))

# example of mydata
mydata = ((0,(2,1)), (1,(4,5)), (0,(1,3)), (0,(-2,1)), (1,(5,3)), (0,(1,1)), (1, (2,2.5)))
# we use parallelize to partition the tuple so spark can work on it in parallel
mydata = sc.parallelize(mydata)

def getDistance(x1, x2):
  """
  calculates the distance between two points
  """
  n = len(x1)
  distance = 0.0
  for i in range(n):
    distance += (x1[i] - x2[i])**2
  return sqrt(distance)


# (k, (x1,...,xn)) where k is the number of nearest neighbors
# and (x1,...,xn) is the point of interest, in this case it have 2 attributes
k = 3
parameters = (3, (2,3))
# Broadcast the parameters to all nodes
bc = sc.broadcast(parameters)

# map the data to (class, (distance, 1)) => sort it from smallest to largest
# => and take the ones with the k shortest "smallest" distances
kNeighbors = mydata.map(lambda x: (x[0], ((getDistance(x[1], bc.value[1]), 1)))) \
                          .sortBy(lambda k: k[1][0], ascending=True) \
                          .take(bc.value[0])
kNeighbors = sc.parallelize(kNeighbors)
print(kNeighbors.collect())

# map to (class, 1) => sum the counts => sort from largest to smallest this time
# => and take the largset one
pred = kNeighbors.map(lambda x: (x[0], x[1][1])) \
                 .reduceByKey(lambda a,b: a+b) \
```

```
                .sortBy(lambda x:x[1], ascending=False) \
                .take(1)

print(f"predicted class is: {pred[0][0]}")
```

- Cross-Validation

```
# number of folds
K = 3
fold_size = int(len(mydata) / K)

CVdata = tuple((mydata[k:k + fold_size],k) for k in range(0, len(mydata), fold_size))
CVdata = sc.parallelize(CVdata)

error = sc.emptyRDD()
p = (2,3)
for i in range(0, len(mydata), fold_size):
    train = CVdata.filter(lambda x: x[1] != i).flatMap(lambda x: x[0])
    test  = CVdata.filter(lambda x: x[1] == i).flatMap(lambda x: x[0])
    testError = test.foreach(lambda x: (x[1][0] - LR(p,i))**2, 1)
    testError = sc.parallelize(testError).reduce(lambda a,b: (a[0]+b[0], a[1]+b[1]))
    error[i] = x[0]/x[1]

MSE = mean(error)

print(f"Average Test Error: {MSE}")
```

Fragmentation/Sharding

Advantages

- Data Locality (put data in the same fragment when often accessed together)
- Minimal communication costs
- Query performance can be better on smaller fragments
- Indexes for fragments can be smaller
- Storage capacities and load can be balanced in the network

Disadvantages

- Queries over different fragments costly ti compute
- Backup and recovery more difficult in distributed systems

Allocation

Ranged Based Allocation - With ranged based allocation each server is responsible for a range of values, for example: the range of values of the row key or the document identifier or the like.

Cost Based Allocation - Each fragment has assigned cost for example: the storage size or the expected query work load

Hash Based Allocation - The document identifiers are hashed and based on the hash are assigned to some server

Data Replication

Advantages

- reliability / higher data availability: for example if a server crashes the copies can still be accessed
- lower latency / (load balancing, data locality and parallelization)

Disadvantages

- consistency problems (update propagation): for example updates made on one replica have to be propagated to all other replicas
- concurrency problems (multi-user write) : user transactions can interfere

Master-slave Replication

- updates performed at a single master site
- propagated to slave sites after some time
- at slave site, data may only be read never updated
- master is single point of failure
- master for one replica, slave for another

Eager vs. lazy replication

- Immediate update of all replicas vs. delayed update (maybe periodically)

Multi-master Replication ("update-anywhere")

- updates permitted at any server holding a replica
- automatic propagation to all replicas
- resolution of conflicts necessary
- better response time
- increased availability for write access because we avoid the bottle neck of a master server

two methods to handle temporary failures and outdated copies - **Hinted handoff**: handle temporary failures - if a replica is unavailable, write requests are delegated to another available server - hint that these requests should be relayed to the replica server as soon as possible - **Read repair**: a coordinator node sends out the read request to couple of replicas - coordinator contacts a set of replicas larger than the needed quorum - returns the majority response to the client - sends repair (update) instruction to those replicas that not yet synchronized

Distributed Concurrency Control

2-Phase Commit Control

is used to finalize a distributed transaction between a set of servers called agents, the servers are agents had to participate in a voting phase and a decision phase

- Execution of a distributed transaction where all agents have to acknowledge a successful finalization of the transaction
- 2PC has a voting phase and a decision phase

- Failure of a single agent will lead to a global abort of the transaction
- The state between the two phases - before the coordinator sends his decision to all agents - is called the in-doubt state
- In case the coordinator irrecoverrably fails before sending his decision, the agents cannot proceed to either commit or abort the transaction

The Consensus Problem

- Failure-tolerant protocols are needed
- What happens if replicas are stored in a distributed database with multi-master replication but suddenly some update messages are lost? What will be the correct value and the right order of updates?
- A **consensus algorithm** ensures that a single one among the proposed values is chosen
- Paxos algorithm can be applied to keep a distributed DBMS in a consistent state under certain failures
  - A client can issue a request
  - Database servers have to come to a consensus on what the current state (and hence the most recent value) of the record is
- Database severs as agents
  - Proposer
  - Leader
  - Acceptor
  - Learner

Consensus Algorithm: Paxos

- Phase 1 (prepare):
  - One proposer (leader) selects a proposal number n and sends prepare request to acceptors
  - If acceptor receives a prepare request with n greater than proposal number of any previous prepare request: respond promise (to not accept any proposals numbered less than n) and include the highest-numbered proposal it has accepted
- Phase 2 (accept):
  - If proposer receive promise from a majority of acceptors: send an accept request to acceptors with value v (which is the value of the highest-numbered proposal among the responses)
  - If acceptor receives an accept request: send accepted message unless it has already responded to a prepare request having a number greater than n
- A value is chosen at proposal number n iff majority of acceptors accept that value in phase 2 of the proposal number.
- Paxos's properties
  - P1: Any proposal number is unique
  - P2: Any two sets of acceptors have at least one acceptor in common.
  - P3: the value sent out in phase 2 is the value of the highest-numbered proposal of all the responses in phase 1.

Failure and paxos

- Failure of acceptor:
  - No problem as long as majority of acceptors agree on a value

  - For N = 2F + 1 acceptors in total, F acceptors can fail as long as F + 1 acceptors reach a consensus
- Failure of learner;
  - There must be at least one learner returning the result to the client
- Failure of leader:
  - Another proposer takes role of leader starting with a higher proposal number
  - Problem: dueling leaders sending prepare messages with increasing proposal numbers (no consensus is reached in this case0)

Version Vectors

- It's important to note we indeed need version vectors with counters for each individual client or accessing application, in particular it is not sufficient just to just keep counters for each replica for example if client "a" and "b" write to one replica we only have a counter for the replica then client "b" will overwrite all previous writes of client "a" in this case all write of client "a" will be lost and not be reflected in the system

Properties of Distributed Databases

- Consistence
  - When a server modifies a value in one replica, the value must be immediately modified in all replicas (at least before someone else reads it)
  - Costly for great number of replicas
  - Might block a read access until all replicas are updated 9bad for availability)
- Availability
  - High performance of query answering but also for write requests
  - Very low response time
- Partition Tolerance
  - Failures in distributed systems (communication link or server failure, network partition) do not lead to failure of the entire system
- Strong consistency is hard to achieve in distributed systems in particular in terms of availablitiy of the system, if after an update any subsequent access returns the updated value read accesses must be blocked which will affect the availability of the system
- A read quorum is defined as a subset of replicas that have to be contacted when reading data; for successful read, all replicas in a read quorum have to return the same answer value.
- A write quorum is defined as a subset of replicas that have to be contacted when writing data; for a successful write, all replicas in the write quorum have to acknowledge the write request.
- Quorum rules for N replicas, read quorum size R, write quorum size W
  - $R + W > N$ (consistent reads)
  - $2W > N$ (consistent writes)
- Partial quorums only ensure weak consistency (for example, during failures)
- Read-one Write-all (ROWA): $R = 1$, $W = N$
- Majority: $R > N/2$, $W > N/2$