Question 1.

OLTP systems support simple, well-defined operations (transactions), therefore, they have a fixed-schema and, in general, they should support ACID properties. OLAP, on the other hand, support complex (aggregation) queries where speed is not a crucial factor, therefore, they may have more 'relaxed' properties than ACID. 1

NoSQL systems have other properties than traditional RDBMS, therefore, they support BASE properties (as opposed to ACID), tend not to have schema, or have schema-on-read. New SQL systems attempt to solve the problem of programming requirement for No-SQL, and offer SQL interface. The underlying data structure can be dependent on implementation. E.g., more structure if built on top of HBase, and less, if built on top of Hadoop.

Question 2.

Vertical scalability means the ability to improve performance by adding more resources, e.g. more memory, more cores, to single nodes.

Horizontal scalability means the ability to improve performance by adding more units of the same type, e.g. more nodes into the cluster.

Question 3.

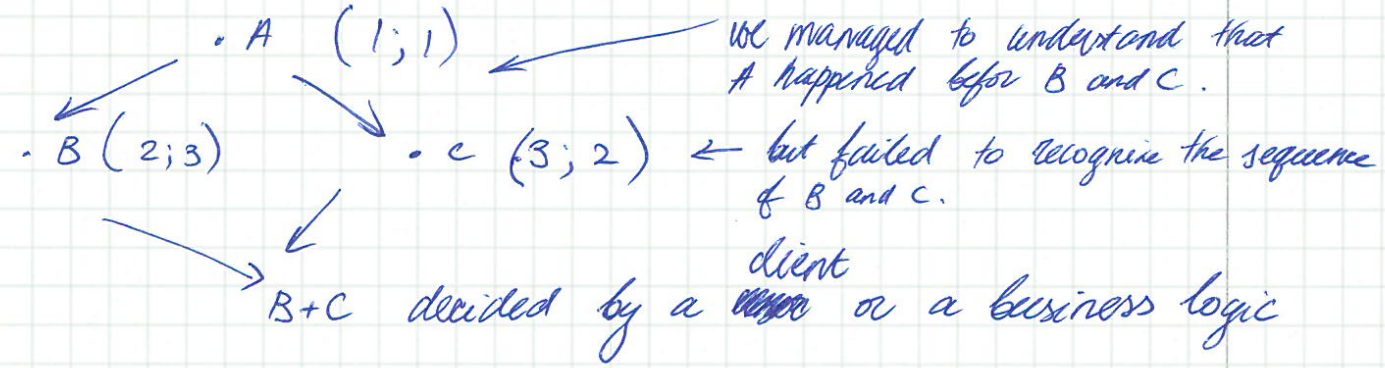
Dynamo DB tracks the order of events by the use of vector clocks. If an inconsitency is detected, it attempts to use vector clocks to decide which version is the most recent (syntactic reconciliation). If vector clocks fail (neither of vectors is larger a equal to all others) a semantic reconcilliation is attempted: throwing both versions at the client and letting him to decide what's the correct version

Syntactic, assume we have 3 versions and try to recover their sequence:

1.5

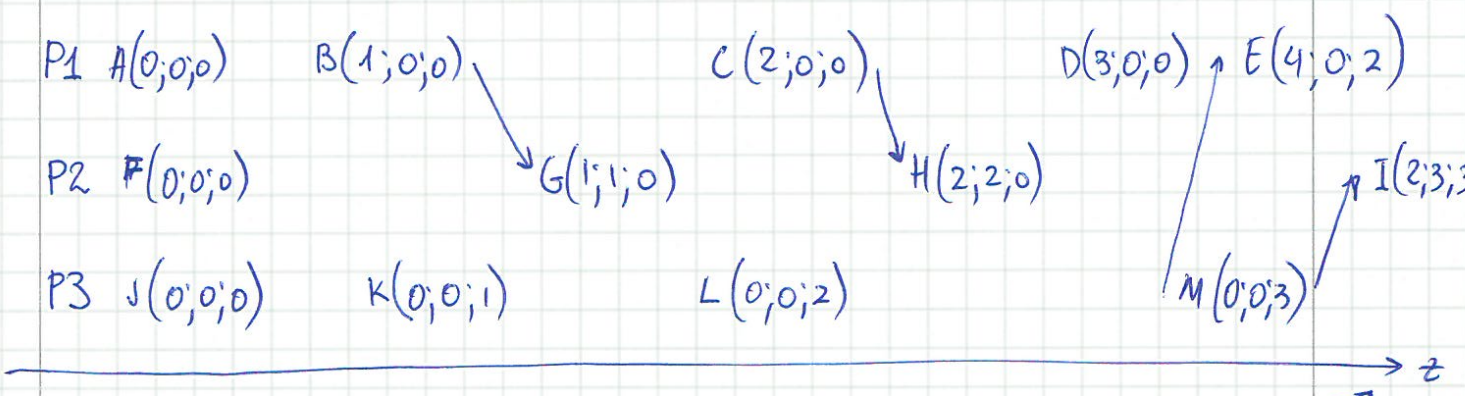
- A (1;1)
- B (2;1)
- C (3;3) ← is the current, latest version

Semantic, assume the same:



Question 4

a)



t is not common, therefore we know that E happened after M, and after D; but we can't say whether L happened before or after H.

b) Events K and B are concurrent, because neither $B \geq K$, nor $K \geq B$, therefore we cannot say which one happened earlier.

Question 5

I/O operations are slow, therefore in I/O-intensive tasks, this is a bottleneck. Thus, we have to utilize other resources (CPU) — more. (✓) Having more tasks than cores can ~~[illegible]~~ solve this problem, if one I/O operation can serve multiple CPU tasks.

0,5

Avoid idle CPU cores by overlapping I/O with computational tasks

Question 6

There are 2 underlying reasons:

1) there's a lot of data (files are usually larger than 64 MB)
2) reading and writing each new block (unless they are consecutive) takes start-up time (≈ latency).

Therefore, (1) allows us to decrease the impact of (2) by increasing the block size: we search for the beginning of a new block less often. ✓

1

Question 7

Let us say we have data on people. It includes person ID, wage-income and non-wage income. We want to find total income of each person, within

total_income_by_person = MapReduce(map = lambda: (ID, wage + non-wage), reduce = identity_function, data = people_data)

total_income_by_person = MapReduce(
map = lambda ID, value : (ID, value[wage] + value[non_wage]),
reduce = identity_function,
data = data_about_people)

Reduce makes aggregations, but we do not need aggregated data, therefore, we ask it to do nothing (identity_function), because all the work is done in the mapper. ✓

1

Question 8

a) combiner ~~applies~~ applies the function that was passed to reducer? locally on each node. ✓ 1

b) It can be omitted, because it's output is in the same format as mapper's output. Therefore, a reducer would still be able to deal with mapper's output only, but it would take more time. ✓ 1

c) When reducer function is commutitative and associative, then it is possible. This increases performance because:
- combiner might start working while other nodes? still run a map function (while reducer should wait for all mappers to finish) ✓
- combiner is done locally and usually output size is decreased, therefore, there's less communication in the shuffle phase ✓ (Also improves data locality) ✓ 1

E.g. wordcount. Mapper produces a list of $(w_i; 1)$, but combiner produces $(w_i, \#w_i$ on this node), which is highly likely to be a shorter list.

Question 2

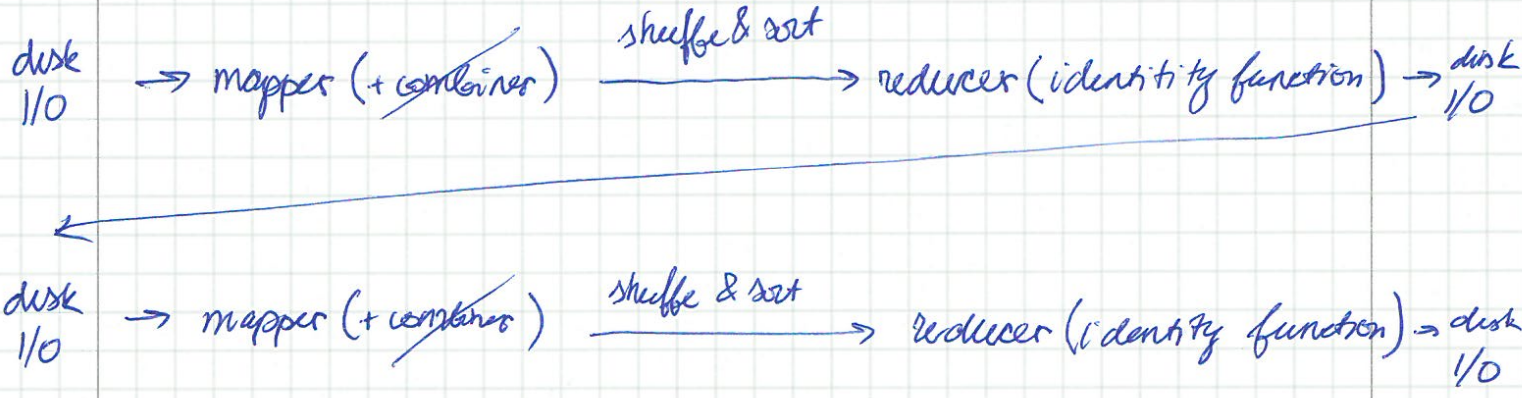
RDD is a container for distributed data. However, it does not really store the data, but rather the lineage graph of how to compute it. The actual data is only materialized when an action function is called. In case of a node failure, RDD also uses its lineage to recompute missing parts.

(lazy evaluation)

1

Question 10

Hadoop:



Spark:

disk I/O → mapper → mapper → disk I/O

therefore, Spark ~~do~~ stores intermidiate results in memory (less I/O operations), and performs as large sequence of operations, called action, as possible locally (less communication).

Since I/O and communication is slow, Spark is faster.

✓ 2

Question 11

Mesos and Yarn manage cluster resources, allocating them to frameworks. But unlike batch-scheduling, they allow for multi-tenancy. This is beneficial for the following reasons:

- During the lifetime of a task, it requires different amount of resources. E.g., mappers usly hit the limits of a cluster, while reducers may occupy just a few (or even 1) node. Thus, resources are underutilised and Mesos/Yarn can assign other task to occupy free resources.
- This (the reason above) could be done by a framework itself, but Yarn/Mesos allows running multiple frameworks (run MPI on nodes that are available, while Spark performs reducing) and resolving task priorities from different frameworks.

✓ good! 1*

Question 12:

Assume that Spark context is initialised in sc. Also assume that the data is in $y, x_1, x_2, \ldots, x_D$ csv format:

```
def distance(x1, x2):
    n = len(x1)
    cumsum = 0
    for i in range(0, n):
        cumsum = cumsum + (x1[i] - x2[i])**2
    return sqrt(cumsum)

# Distribute closed ball radius, h, and new X, new X
distributed = sc.distribute({"h" = ... ; "new X" = (..., ..., ...)})

# Read file and format it into RDD with
# (y, (tuple_with_x)) structure
observations = (sc
    .textFile("hdfs://...")
    .split(",")
    .map(lambda a: (float(a[0]), float(a[1:D])))

observationsInBall = (observations
    .map(lambda a: (a[0], distance(a[1], distributed.value["new X"])))
    .filter(lambda a: a[1] <= distributed.value["h"])
    .map(lambda : (a[0], int(1))))

# observationsInBall contain only observations inside the ball
# with (y, 1) structure.
```

I expect this produces 0, 1, ..., n-1.

I expect this to produce a tuple of floats

Question 12 (continued):

```
# now we sum all ys and all 1s
summation = (observationsInBall
  . reduce (lambda cumsum, a: (cumsum[0] + a[0],
                               cumsum[1] + a[1])))

yHat = ifelse(summation[0] / summation[1] <= 0.5, 0, 1)
```

I'm afraid I might have borrowed this construction from R, but I hope it's clear what I expect to get.

Question 13:

Assume ~~K~~ $K << n$, otherwise splitting by randomly assigning a fold number will not work correctly. Also, assume that we have RDD 'observations' as the one in Q12.

```
# Assign a fold to each observation and obtain RDD
# with the following structure: (fold, (y, (tuple_of_xs)))
obsWithFolds = (observations
    .map(lambda obs: (random(1, k), obs)).cache())
```

← assume this returns an integer uniformly between 1 and k

```
MSE = []
for i in range(1, k):
```

← assume this is from 1 to k (not k-1, not from 2...)

```
    # Split data into test and train:
    # filter only appropriate folds and then remove fold
    # number from the structure: (y, (tuple_of_x))
    train = (obsWithFolds
        .filter(lambda a: not a[0] == i)
        .map(lambda a: a[1]))
    test = (obsWithFolds
        .filter(lambda a: a[0] == i)
        .map(lambda a: a[1]))
```

I would have preferred map still

```
    # Predict ~~train~~ ~~test~~ test data
    # we run foreach, not map, because MWC (if implemented
    # as in Q12) uses map, and map inside map is not supported
```

Q13 (continued):

```
SE = test
    .foreach(lambda obs: ((obs[0] - MWC(obs[1], train, h))^2, 1))
# SE has the following structure: (squared_error, 1), and it's a list not RDD.
# Sum errors and ones:
currentMSE = parallelize(SE)
    .reduce(lambda cumul, a: (cumul[0] + a[0], cumul[1] + a[1]))
MSE[i] = currentMSE[0] / currentMSE[1]
```

this should be inside the for loop, but I am bad with indents on paper. The following is outside for loop:

```
MSE = mean(MSE)
```

Question 14:

Spark attempts to avoid unnecessary I/O and communication operations by creating a lineage graph and trying to optimize it (running as many operations locally as possible).

It occurs whenever we call multiple sequential transformation operations. E.g. when we compute ObservationsInBall (Q12) none of the operations occur immidiatly (only lineage is created) but when we eventually call an action (in sumSummation) all operations occur lokally.

In Q13, we also use cache() in obsWithFolds to hint that this data should be stored in memory (if possible), thus also avoiding unnessary computations / IO-operations.