Question 1 (1p)

Consider the following claim:

> *Write scalability can be achieved not only by scaling horizontally (scale out), but also by scaling vertically (scale up).*

Is this claim correct or wrong? Justify your answer in about two to four sentences.

答案：

I think it's correct. Scaling vertically will enhance a server's capacity including parallel computing ability and the speed of manipulating queries. So when the number of read operations increase , it will not lose performance.

额外 tip：

Read and write scalability can both be achieved horizontally and vertically

The structure is like Systems-nodes-processors chips-cores. And the data is partitioned and distributed in different nodes.

Question 2 (1p)

Consider the following relational database which consists of two relations (Project and Report). Notice that the attribute project in the relation Report is a foreign key that references the primary key (attribute name) in the relation Project. Notice also that multiple reports may be for the same project.

Project

name	budget
UsMis	1,000,000
AMee3	3,700,000
Bee	1,300,000

Report

id	project	filename
121	Bee	beerep.pdf
391	UsMis	rep391.pdf
699	Bee	OldRep.pdf

Capture *all* the data in this relational database as a key-value database.

答案：

key	value
UsMis	['1,000,000','121','rep391.pdf']
AMee3	['3,700,000','391',]
Bee	['1,300,000','699',['beerep.pdf','OldRep.pdf']]

如果是 column-database，多变量的 column 要变成 dummy variable 的形式用 True 和 False 来表征，然后再用列家族来表征这些 dummy variable。见 NoSql ppt。

Question 3 (1p)

Remember that values in a key-value database are opaque to the key-value store. What exactly does this mean both

i) from the perspective of the key-value store and

ii) for a user of such a system?

Answer this question in two to five sentences *by using your key-value database from the previous question as an example.*

I) No secondary indexes for values. In the previous example, we cant only get report id of project Bee without getting whole row of that data.
II) It means db cannot be queried over the value of it's entry. We can only access the data by key not by values. E.g. although we only want to compare the budget of these three projects, we still have to access all the data by it's key.

Question 4 (1p)

Data warehouses are usually much bigger in size than operational databases. Explain, in two to five sentences, why that is.

Data warehouses generally need to store data going back longer in time than an operational database in order to do analytics of information going back longer in time. It is also common for data warehouses to have data from multiple sources, not only the operational database and this of course increases the size of the warehouse as well.

Question 5 (1p)

Assume a (multi-master) system in which each database object is replicated at 4 nodes. We want to allow the system to require a quorum of only 2 nodes when we *write* a database object (i.e., for the write to be considered successful, 2 nodes have to confirm that they have completed the write). Then, in order to achieve strong consistency (for reads), how many nodes have to agree on the value to be returned in response to a read request for a database object? Justify your answer in about two to four sentences.

If we assume the total nodes are N, the number of nodes for read is R, the number of nodes for write is W. Then the strong consistency require R+W>N. Here we have N=4 and W=2, so R have to be 3.

Question 6 (1p)

We learned about the concept of high-level parallel programming using *algorithmic skeletons*. Give one example of such an algorithmic skeleton that occurs in the Spark API. (Hint: it is *not* MapReduce.)

Explain shortly what the given algorithmic skeleton does, describe its input-output dependence structure and whether Spark evaluates it lazily or not.

To answer this question write a maximum of 100 words.

The map function is an example of a algorithmic skeleton available in the Spark API. What it does is apply a function (given by parameters) and applies that function each element in a list/RDD. It has no dependencies between the elements as each element is operated on in isolation. In spark map is evaluated lazily.
迟钝执行：只有当数据被使用时才进行计算

Question 7 (1 + 1 = 2p)

The block size used in the Hadoop distributed file system is, by default, many Megabytes large (typically, 64MB). Considering the way how cluster architectures and parallel computations atop HDFS are organized, give the technical reasons why this size is usually a reasonable choice from a performance point of view:

(a) why should one not use much smaller block sizes (e.g. just a few dozen bytes)?

(b) why should one not use much larger ones (e.g. many Gigabytes) either?

For each of these two questions, write a maximum of 100 words, respectively.

a) Too small block size will result in too many blocks which will create too many metadata to store the relation between each blocks. And also the tasks will be divided into many too small piece which may be difficult to parallelize.
b) With to large blocks we might get to few tasks to keep all our nodes occupied or not give us enough flexibility in our scheduling. We want many more tasks than we have nodes. It also means if a node fails or is very slowed down because of some problem the amount of work we must redo is not as large.

Question 8 (1p)

We learned about different node interconnection networks that can be used in clusters, and the implications of their topology for network cost, throughput, and scalability. Give the name and describe these properties (be thorough) for a very cheap interconnection network which has very limited throughput and which does not scale up to many nodes.

To answer this question write a maximum of 100 words.

Cost: 1 lines and no switch
Scalability: The structure is very simple so it's good in cost scalability.
Throughput: Each nodes need to communicate with each other, so the throughput will be limited by bus line width.That's also the reason why this network cannot scale up too much because bus width will be bottleneck.

See: CK-F1 P12
Bus interconnection.

Question 9 (1p)

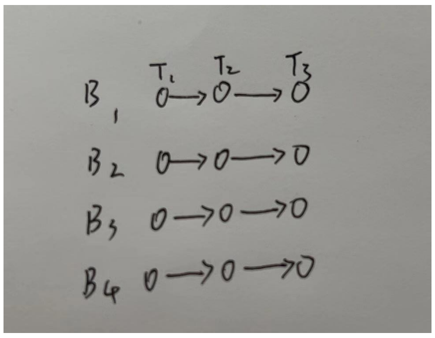
Given a Spark program that consists of a chain of $K > 1$ dependent *transformations* $T_1, ..., T_K$, where each transformation T_i uses the result of the previous transformation T_{i-1} as its input. The input (of T_1) is a HDFS file with B blocks. Assume that each transformation takes time t_B per block.

First, draw the *task (dependence) graph* for this computation for $K = 3$ and $B = 4$, where each transformation over each block constitutes a task.

If $P \geq B$ workers are available for this computation, what is the fastest possible parallel time (general formula, using the above symbols) for the entire computation? Explain your calculation.

To answer this question write a maximum of 100 words.

Since each transformation takes t_B time to complete and there are 3 transformation tasks that are to be done in serial the amount of time to calculate transformations for one block is 3*t_B. And since the different blocks are fully parallelizable(available P>=B) the fastest possible parallel time is 3*t_B.



Question 10 (4p)

In this course, we have seen examples of classification (e.g., logistic regression), regression (e.g., kernel methods) and clustering (e.g., k-means). Yet another important task in machine learning is that of density estimation. The goal in this task is estimating the density function $f(X = x)$. This task can be solved via kernel methods as

$$f(X = x) = \frac{1}{N} \sum_n k\Big(\frac{x - x_n}{h}\Big)$$

assuming that the kernel function $k(u)$ satisfies the following two conditions:$^{(*)}$ i) $k(u) \geq 0$ for all u and ii) $\int k(u)du = 1$.

Assume that we have access to some training data of the form

$$\{(x_n, y_n) | n = 1, \dots, N\},$$

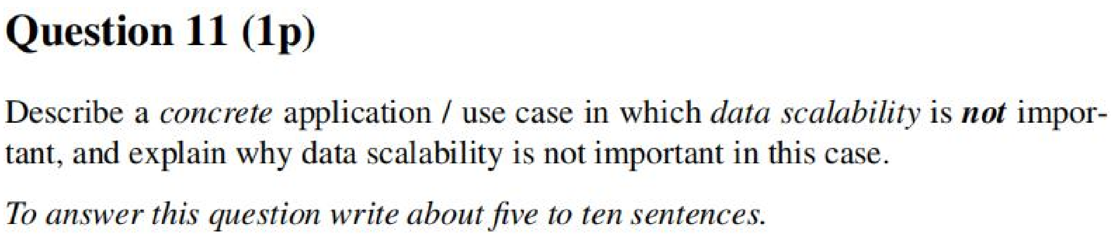
where X_n is a unidimensional continuous predictor random variable and Y_n is a binary class random variable. You are asked to implement in PySpark the kernel method above to estimate $f(X = 1|Y = 0)$ and $f(X = 2|Y = 0)$. Hence, you have to estimate the density function values for the points $X = 1, 2$ for class $Y = 0$. The training data is in the text file `data.csv`. You can call the PySpark function `kernel(u)`, which implements a kernel function that satisfies the conditions $(*)$ mentioned above.

To get full points you need to comment your code.

```
sc = SparkContext(appName="lab_kernel")

stations = sc.textFile("data.csv")

lines_stations = stations.map(lambda line: line.split(";"))

data_0 = stations.filter(stations[1] == 0) # Get data where Y = 0

# do the kernel calculation, there are two kernel value list containing x=1 and x=2,

# the second element 1 here means count,we use this to count the number of data.

density_1 = data_0.map(lambda x: (kernel((1 - x[0])/h), 1))

density_2 = data_0.map(lambda x: (kernel((2 - x[0])/h), 1))
```

```
# use reduce to get the sum of kernel value and also the number of data

d1 = density_1.reduce(lambda x, y: (x[0] + x[1], y[0] + y[1]))

d2 = density_2.reduce(lambda x, y: (x[0] + x[1], y[0] + y[1]))

# get the density value respectively

d1 = d1[0]/d1[1]

d2 = d2[0]/d2[1]
```

Part2

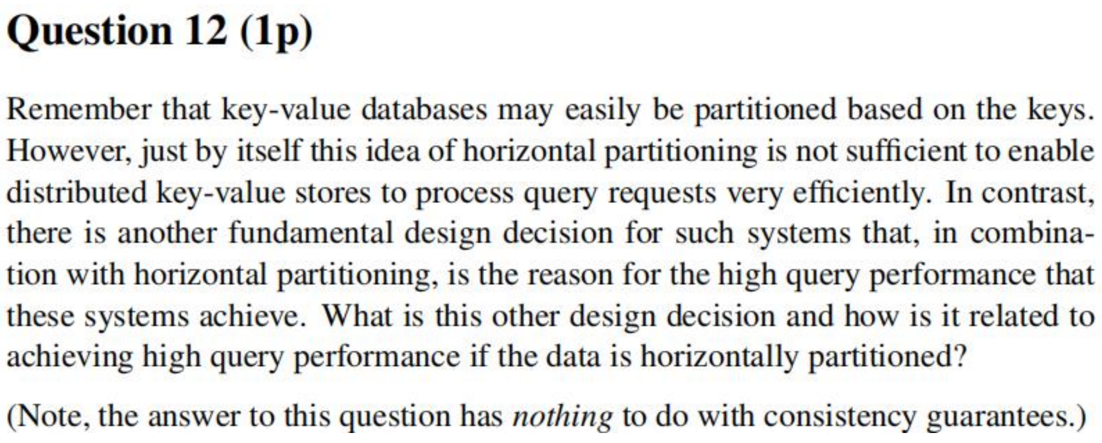


Question 11 (1p)

Describe a *concrete* application / use case in which *data scalability* is ***not*** important, and explain why data scalability is not important in this case.

To answer this question write about five to ten sentences.

A blog website of only one persons blog can be a application where data scalability is not important. Although many read requests might be sent all requests are of the same small amount of data so we don’t need good data scalability.



Question 12 (1p)

Remember that key-value databases may easily be partitioned based on the keys. However, just by itself this idea of horizontal partitioning is not sufficient to enable distributed key-value stores to process query requests very efficiently. In contrast, there is another fundamental design decision for such systems that, in combination with horizontal partitioning, is the reason for the high query performance that these systems achieve. What is this other design decision and how is it related to achieving high query performance if the data is horizontally partitioned?

(Note, the answer to this question has *nothing* to do with consistency guarantees.)

The decision to have values be opaque. This help achieve high query performance as each query has to specify what key its operation is on, and this means you can partition based on keys and it will be very easy and fast to know what node a query needs to go to. If value-based queries were allowed, a query could request an entry with budget 1.000.000 (in previously mentioned DB) and the system would not know which node that item resides on without complex indexing

Question 13 (1p)

Consider a NoSQL system which does not provide strong consistency, but only some form of weak consistency. Remember that weak consistency means that, after updating the value of a data item in a database managed by such a system, there is no guarantee that all subsequent requests to retrieve the value of that data item will return the updated value. Explain in two to five sentences how it may happen that the old value will still be returned in response to such a request.

If the data item is replicated over multiple nodes the value can be updated on one node and then a request for that data item might be received on another node that has not yet been updated with the new value of the data item. The request will then receive the old value in response

Question 14 (1p)

Recall that the data of a data warehouse may be represented using the multidimensional data model where numeric measures are captured in a multidimensional array and the attributes of some dimensions form hierarchies. Figure 1a illustrates an example of such a multidimensional array with three dimensions: a product dimensions, a city dimensions, and a date dimension. The numeric measures in this example are sales values; that is, any value in this array represents how often the corresponding product was sold in the corresponding city at the corresponding date. Figure 1b illustrates possible hierarchies for the three dimensions.

Considering the typical types of operations over multidimensional data (e.g., slicing, dicing, drill-down), specify which operation(s) has/have to be applied to this example data if you want to analyze and compare the total sales of a specific product, say toothpaste, in a specific city, say Linköping, for all the different years. While you do not have to define the operation(s) formally, try to be as concrete as possible. For instance, do not just write that drill-down has to be applied, but be

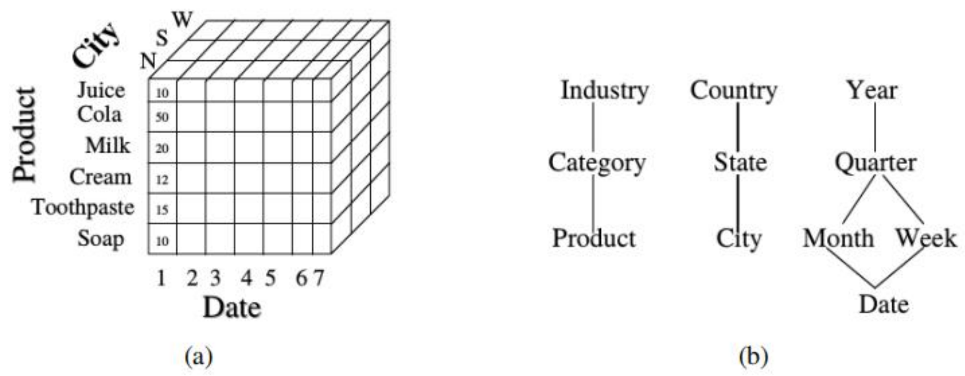


Figure 1: Example of a multidimensional array with corresponding hierarchies for the dimensions.

concrete about the specific drill-down operation that you have in mind. If multiple operations have to be applied one after another, specify them in the order in which you would apply them.

1 slicing the specific product dimension: toothpaste.
2 slicing the specific city dimension: Linkoping
3 roll-up along the date dimension to aggregate the date in year level.

Question 15 (1p)

What is the advantage for the *owner* of a cluster resource with many big-data computing users to use a system like Mesos or Yarn, compared to a batch reservation scheduler as known from HPC clusters? Explain your answer (technical reasons).

To answer this question write a maximum of 150 words.

When using a system like Mesos or YARN we can run multiple jobs (which use different frameworks) simultaneously on a node. This allows us to both better utilize the resources of the cluster and to divide the resources each user can use to be smaller than an entire node. When using batch reservation scheduler each job reserves all the resources of a node for the duration of it's job. This might be very inefficient if e.g. one job is very I/O heavy and thus don't use the CPU much while another job that is very CPU intensive is waiting in queue to run.

Question 16 (1p)

Given is a distributed HDFS text file that contains the complete log of all LiU student course registrations up to now. The file consists of N lines that are each of the form `LiU-ID:coursecode:year`, e.g.,

```
...
liuid123:TDDE31:2020
abcde789:TDDE31:2021
liuid123:TDDE99:2020
...
```

Assume you would write a *MapReduce* program (you do *not* have to actually write it!) that produces an HDFS text file that contains one line for each course code, showing the number of registrations; for example:

```
...
TDDE30:174
TDDE31:123
...
TDDE99:27
...
```

Notice, as the HDFS file is not extremely large, the number of mapper tasks will be rather low but larger than one.

Do not write that program. Instead, answer the following question *and* motivate your answer: Will it be beneficial here to use a combiner or not?

To answer this question write a maximum of 100 words.

Yes. Since we will have many entries with same key, and the operation here is only sum and sum is commutative and associative. So, combiner will be a good choice to avoid mass intermediate data.

Question 17 (1p)

For HDFS with 64MB blocks and the usual replication scheme for fault tolerance, we run a single MapReduce step using only the mapping phase. The input is a large HDFS text file containing N lines. Each line contains 64 bytes of text consisting of a log message and a measured floating point value. The mapper user function leaves the log message unchanged, keeps positive values unchanged and changes negative values to zeroes.

How many HDFS block reads and how many HDFS block writes are performed in total for $N = 10^9$? Give a short explanation and calculation.

To answer this question write a maximum of 100 words.

For N = 10^9 the total file size is 64 * 10^9 = 64GB. With blocks of size 64MB = 64 * 10^6 we have (64*10^9) / (64*10^6) = 1000 blocks. The record reader phase will read each block, meaning we will do 1000 HDFS block reads. And because we don't reduce the output, we will have the same number of blocks as output and so will write 1000 new blocks, and since we have 3 replicas for fault tolerance, and 3000 HDFS block writes will be done.

Assuming a block is written even if no changes were made to the data (which would happen if no entries in an entire block contained a negative reading).

Question 18 (1p)

On an HDFS cluster, according to which main criterion does the MapReduce scheduler in the master process assign mapper tasks to the workers? And why is this even more important if the cluster's interconnection network has very limited throughput capacity?

To answer this question write a maximum of 100 words.

The main criterion the scheduler uses is data locality. That is, it tries to schedule a task on or close to the node where the data the task needs is currently residing. This is even more important if the interconnection network has limited throughput since it reduces the amount of communication needed and on a limited network the communication will be slow. It might also happen the network is overloaded, increasing communication wait time even more.

Question 19 (1p)

On iterative computations that consist of many transformations, Spark significantly outperforms MapReduce. Why? Provide a technical explanation.

To answer this question write a maximum of 150 words.

Because if we run multiple MapReduce tasks the results after each tasks will be written to disk that is then read in the next MapReduce task. When using Spark the map tasks will be lazily evaluated and this gives Spark the ability to optimize the chain of transformations. E.g. when they are done sequentially close in time the data is available in main memory and we don’t need to read as much from disk. E.g. after the first map the result will be in main memory and the next map task will therefor be able to access it’s input data quickly.

Question 20 (6p)

Once we have a kernel method solution to the density estimation problem, it is easy to turn it into a probabilistic classifier. To see it, say that we have to predict the class label for a point $X = x$. Then, you can do the following:

1. Estimate $f(X = x|Y = 0)$ and $f(X = x|Y = 1)$ using kernel methods.
2. Estimate $p(Y = 0)$ and $p(Y = 1)$ using maximum likelihood, i.e. estimate these quantities as the proportions of training points labeled with $Y = 0$ and $Y = 1$, respectively.
3. Use Bayes theorem to compute $p(Y = 0|X = x)$ as

$$p(Y = 0|X = x) = \frac{f(X = x|Y = 0)p(Y = 0)}{f(X = x|Y = 0)p(Y = 0) + f(X = x|Y = 1)p(Y = 1)}$$

   and analogously for $p(Y = 1|X = x)$.

Your task is to implement in PySpark the kernel-based probabilistic classifier described above, in order to predict $p(Y = 0|X = 1)$.

To get full points you need to comment your code.

```
data = sc.readFrile('data.csv')

data = data.map(lambda x: x.split(';'))

# x[0] is x; x[1] is [y]

data = data.map(lambda x: (x[0], x[1], 1))

# get the number of observation where Y=1

data_y = data.reduce(lambda x, y: (x[1] + y[1], x[2] + y[2]))

y_1 = data_y[0] / data_y[1]

y_0 = 1 - y_1



# calculate maximum likelihood of f(X=1,Y=0)

data_y0 = data.filter(data[1] == 0)
```

```
data_y0 = data.map(lambda x: (kernel((1 - x[0]) / h), x[1]))

ml_y0 = data_y0.reduce(lambda x, y: (x[0] + y[0]))[0]

# calculate maximum likelihood of f(X=1,Y=1)

data_y1 = data.filter(data[1] == 1)

data_y1 = data.map(lambda x: (kernel((1 - x[0]) / h), x[1]))

ml_y1 = data_y0.reduce(lambda x, y: (x[0] + y[0]))[0]

res = ml_y0*y_0/(ml_y1*y_1+ml_y0*y_0)
```