BDM-MIRI - Big Data Management - Exercises

Jose Antonio Lorencio Abril

Spring 2023

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Professor: Alberto Abelló

Student e-mail: jose.antonio.lorencio@estudiantat..upc.edu

This are the exercises of the course *Big Data Management* taught at the Universitat Politècnica de Catalunya by Professor Alberto Abelló in the academic year 22/23. Some exercises have been solved in class (marked with *).

Contents

1 Big Data Design

1.1 Theoretical questions

1.1.1 Briefly explain what 'physical independence' is, which is the general position of NOSQL systems on this, and why they take this position.

'Physical independence' in the context of databases refers to the ability to modify the physical storage of data (like its distribution across disks or its indexing strategy) without affecting the conceptual view of the data. NoSQL systems generally advocate for physical independence, mainly because they are designed to work with massive volumes of data which can be distributed across multiple nodes. Physical independence facilitates horizontal scaling (adding more machines) to handle larger datasets and workload, which is a core feature of NoSQL databases.

1.1.2 In the framework of the RUM conjecture, name six data structures we saw in the course (not concrete tool implementations) and place them in the corresponding category:

- **Read optimized:** B-Trees, Hash Indexes
- **Write optimized:** Log-structured merge-trees (LSM trees), Hash Map
- **Space optimized:** Bitmap Indexes, Trie

Remember that the RUM (Read, Update, Memory) conjecture is a framework that stipulates a trade-off between read efficiency, write (update) efficiency, and memory efficiency.

1.1.3 Compare a B-tree and a LSM-tree in the context of the RUM conjectura (i.e. as an answer to this question, three brief explanations of the form 'From the perspective of X, Y-tree is better than Z-tree, because of this and that' are expected).

- **R:** A B-Tree is generally better than an LSM-Tree, as B-Trees provide faster read operations due to the hierarchical nature and sorted order of their data.
- **U:** An LSM-Tree is generally better than a B-Tree in terms of updates. LSM-Trees are write-optimized and handle write operations better because they buffer changes in memory and then write large blocks to disk.
- **M:** An LSM-Tree is more memory-optimized than a B-Tree. LSM-Trees use less memory because they compact and compress old data, thus reducing storage footprint.

1.2 Problems

1.2.1 Consider

1. **['BCN', POPULATION:{VALUE:'2 000 000'}, REGION{VALUE:'CAT'}]**
2. **['BCN', ALL:{VALUE:'2 000 000;CAT'}]**
3. **['BCN', ALL:{POPULATION:'2 000 000'; REGION:'CAT'}]**

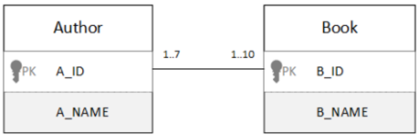
Place {1,2,3} in the table *:

	Less variable schema	→	**More variable schema**
Less explicit schema			2.
↓		3.	
More explicit schema	1.		

Name two criteria you would use to choose among them.

- Variety of data.
- Variability of the data.

1.2.2 Consider the following conceptual schema and propose several design alternatives to translate it to a logical representation(e.g. using JSON notation). Then, explain which is the best alternative and why. *



```
Author: {A_ID, A_NAME}
Book: {B_ID, B_NAME}
AuthorBooks: {A_ID, B_ID}
```

```
Author: {A_ID, A_NAME,
        Books: {
              Book1: {B_ID, B_NAME},
              ...
              BookN: {B_ID, B_NAME}
            }
    }
```

```
Book: {B_ID, B_NAME,
        Authors: {
              Author1: {A_ID, A_NAME},
              ...
              AuthorN: {A_ID, A_NAME}
            }
    }
```

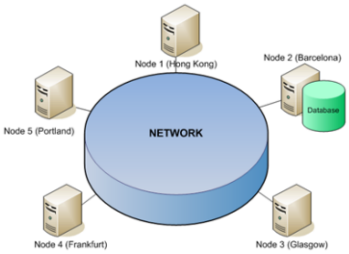
```
Author: {A_ID, A_NAME, Book1:{B_ID,B_NAME},..., BookN:{B_ID,B_NAME}}
```

And many more...

2 Distributed Data

2.1 Theoretical questions

2.1.1 Which kind of database is this according to the distribution of data?*



This is a centralized system with respect to the data, because the database is completely handled in Node 2.

2.1.2 Explain what is (a) a distributed system, and (b) a parallel system. Compare both of them (i.e. what has one and not the other and vice-versa).

a) A **distributed system** is a system where components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other to appear as a single coherent system to the end user.

b) A **parallel system**, on the other hand, is a type of computational system that performs many operations or processes simultaneously. Parallel systems can be a single computer with multiple processors or an array of computers working together to achieve a common goal.

While both systems process tasks concurrently, a distributed system spreads the tasks across different networked machines that may be geographically dispersed. In contrast, a parallel system typically uses multiple processors within a single machine or across a tightly coupled cluster of machines to perform tasks simultaneously.

2.1.3 Which two kinds of schema information contains the Global Conceptual Schema that does not contain the Local Conceptual Schema in the Extended ANSI-SPARC Architecture for DDBMS? *

1. Fragmentation
2. Allocation

2.1.4 In the context of distributed data management, name the four big challenges that need to be carefully considered in the presence of distribution from the tenant/user point of view. *

1. Data design.
2. Catalog management.
3. Transaction management.
4. Query processing.

2.1.5 Name the three characteristics of fragmentation that make it correct. *

1. Disjoint.
2. Complete.
3. Reconstructible.

2.1.6 Which is the main problem in having replicas, and which is the innovation introduced by some NOSQL tools to solve it? *

- **Problem:** Consistency.
- **Innovation:** Eventual consistency.

2.1.7 Given N replicas, let's call R the ReadConcern parameter of MongoDB and W the WriteConcert (which indicate, respectively, the number of copies it reads and writes before confirming the operation to the user). Give the equation involving those variables that corresponds to the eventually consistent configuration. *

The formula is:

$$W + R \leq N.$$

2.1.8 What is the difference between query cost and query response time... *

1. **In centralized systems?**
2. **In distributed systems?**

Query cost refers to the total amount of resources (such as CPU, memory, disk I/O, network I/O) consumed to execute a query, whereas query response time is the time taken to execute a query and return a result.

- In **centralized systems**, query cost and query response time are tightly correlated. As the system is centralized, resource allocation and scheduling are predictable and straightforward, hence the cost directly impacts the response time. Roughly, $RT \propto C$.
- In **distributed systems**, the correlation is more complex. Query cost is not the only factor impacting query response time. Network latency, load balancing, data locality, and the consistency model also significantly affect the response time. Thus, a query may have a low cost in terms of resources but may still experience high response time due to network delays or synchronization overhead. In this case, $RT \propto USL(C)$ (the universal scalability law).

2.1.9 Name the two factors that make it impossible to have linear scalability according to the Universal Scalability Law. *

1. Programs are not fully parallelizable.
2. There are communication costs.

2.2 Problems

2.2.1 Briefly explain (a) which fragmentation strategy has been applied for the tables below and whether this fragmentation strategy is (b) complete, (c) disjoint and (d) allows to reconstruct the global relations (if so, (e) indicate the operation). *

Global Relations

———————————————————————

`Kids(kidId, name, address, age)`
`Toys(toyId, name, price)`
`Request(kidId, toyId, willingness)`

———————————————————————

Note that `requests(kidId)` is a FK to `kids(kidId)` and `requests(toyId)` is a FK to `toys(toyId)`.

Fragments

———————————————————————

K1 = Kids[kidId, name]
K2 = Kids[kidId, address, age]

———————————————————————

T1 = Toys(price $\geq$ 150)
T2 = toys(price $<$ 150)

———————————————————————

R1 = Requests $\ltimes$ T1
R2 = Requests $\ltimes$ T2

Fragment	a)	b)	c)	d)	
K1-K2	Vertical	Complete	Disjoint	Reconstructible	
T1-T2	Horizontal	Complete (if there are no NULLs)	Disjoint	Reconstructible (if complete)	
R1-R2	Derived Horizontal	Complete (if no NULLs)	Disjoint	Reconstructible (if complete)	Proj

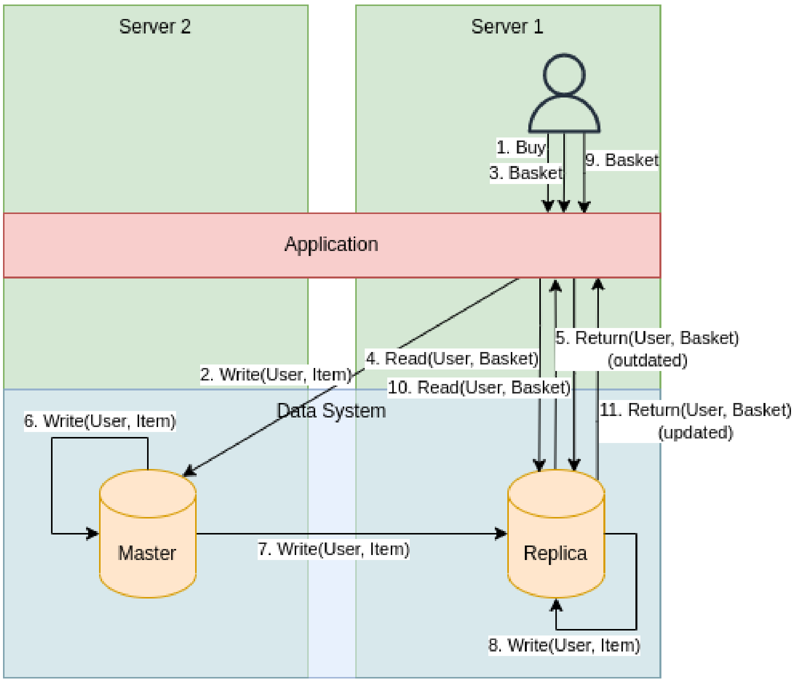
2.2.2 You are a customer using an e-commerce based on heavy replication (e.g. Amazon): *

1. **Show a database replication strategy (e.g. sketch it) where:**
   (a) **You buy an item, but this item does not appear in your basket.**

(b) **You reload the page: the item appears.**

What happened?

This can happen in a Primary/Lazy setup, as the following sketch (the number indicate the order of each action):



2. **Show a database replication strategy (e.g. sketch it) where:**

(a) **You delete an item from your command, and add another one: the basket shows both items.**

What happened? Will the situation change if you reload the page?

This can happen in a Secondary/Lazy setup, in which different replicas are used for each of these purposes, as the following sketch:

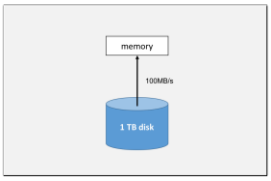
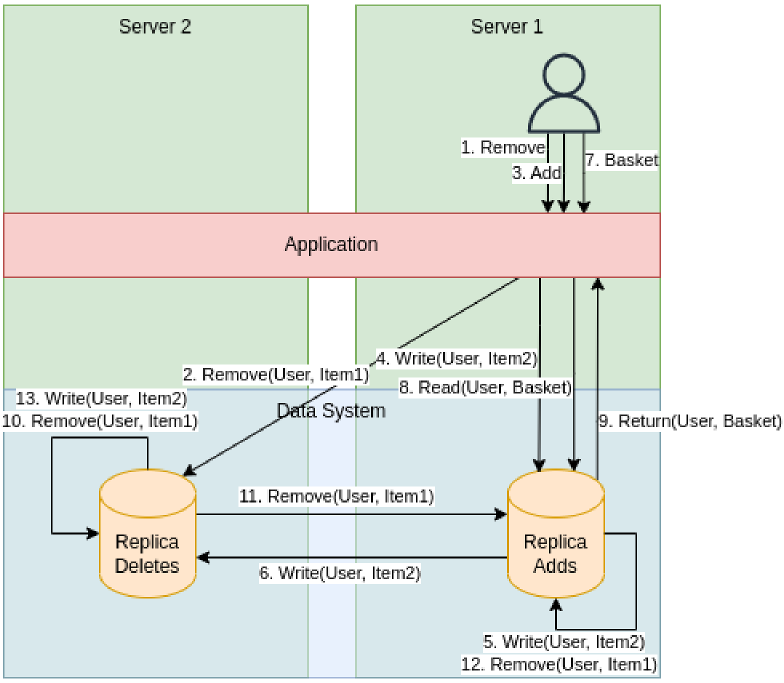


Figure 1: Centralized architecture.



2.2.3 Consider the following architecture and answer the questions[1]: *

Type	Latency	Bandwidth
Disk	$\approx 5 \times 10^{-3}s\,(5\ millisec.)$	At best 100 MB/s

1. **How long would it take (i.e. response time) to read 1TB with sequential access (Figure 1)? (in secs)**

$$T_{transfer} = \frac{1TB}{100MB/s} = 10^4 s = 10000s$$

Therefore,

$$T_{total} = T_{transfer} + T_{latency} = 10000.005s.$$

2. **How long would a single random access (i.e. reading one tuple, of for example 100 B, through an index) take (i.e. response time), assuming we already have the physical address? (in secs)**

$$T_{total} = \frac{100B}{100MB/s} + 0.005 = 0.005001s.$$

Type	Latency	Bandwidth
Disk	$\approx 5 \times 10^{-3}s\,(5\ millisec.)$	At best 100 MB/s

[1] From S. Abiteboul et al. *Web Data Management.* Cambridge Press, 2011.

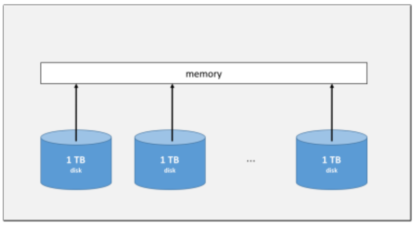


Figure 2: Shared-memory architecture.

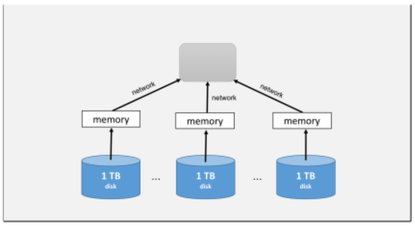


Figure 3: Shared-nothing architecture.

3. **How long would it take (i.e. response time) to read 1TB with parallel access (Figure 2)? Assume 100 disks (i.e. 100 replicas of the whole data) on the same machine with shared-memory and infinite CPU capacity.**

   In this case, we can read a portion of data from each machine. The latency remains still for all machines, therefore:
$$T_{total} = \frac{S_{data}}{N_{machines} \cdot B} + T_{latency} = \frac{1TB}{100 \cdot 100MB/s} + 0.005 = 100,005s.$$

4. **How long would a single random access (i.e. reading one tuple, of 100 B, through an index) take (i.e. response time), assuming we already have the physical address? (in secs)**

   We would know the machine in which the tuple is. Therefore, the situation would be the same as doing it in one machine_
$$T_{total} = 0.005001s.$$

Type	Latency	Bandwidth
Disk	$\approx 5 \times 10^{-3}s\,(5\ millisec.)$	At best 100 MB/s
LAN	$\approx 1/2\ \times 10^{-2}s\,(1-2\ millisec.)$	$\approx 1\ GB/s\,(single\ rack) \approx 10\ MB/s\,(switched)$
Internet	Highly variable (10-100ms)	Highly variable (few MB/s, 10MB/s)

Remark 2.1. It is approximately one order of magnitude to exchange main memory data between 2 machines in a data center, that to read on the disk.

Remark 2.2. Exchanging through the Internet is slow and unreliable with respect to LANs.

5. **How long would it take (i.e. response time) to read 1TB with distributed access (Figure 3)? Assume 100 shared-nothing machines (with all data replicated in each of them) in a star-shape LAN in a single rack where all data is sent to the center of the star in only one network hop.**

   In this case, the network is very fast, and the bottleneck is on the disk. The time is
$$T_{total} = \max\left(T_{disk}, T_{LAN}\right) + T_{latency,disk} + T_{latency,LAN} = 100 + 0.005 + 0.002 = 100.007s.$$

   The reason we take the max is because once we have data in the memory taken from disk, we can send it through the LAN. Therefore, we can consider that both processes are done simoultaneously.

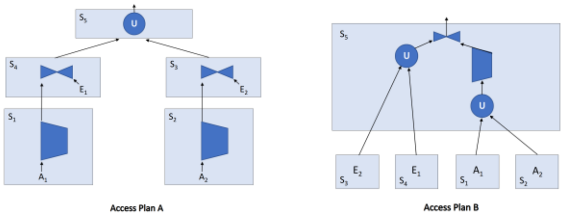


Figure 4: Distributed Access Plans

6. **How long would it take (i.e. response time) to read 1TB with distributed access (Figure 3)? Assume 100 shared-nothing machines (with all data replicated in each of them) in a star-shape cluster of machines connected through the Internet where all data is sent ot the center of the star in only one network hop.**

   In this case, the network is the bottleneck, so

$$T_{total} = T_{internet} + T_{latency,disk} + T_{latency,internet} = \frac{1TB}{100 \cdot 10MB/s} + 0.005 + 0.1 = 1000.05s.$$

7. **How long would a single random access (i.e. reading one tuple, of 100 B, through an index) take (i.e. response time), assuming we already have the physical address? (in secs) In the LAN (switched) scenario.**

   In this case, we cannot take advantage of the paralelization, because only one tuple is sent. Therefore, we need to take the time for all sequential steps into account:

$$T_{total} = T_{disk} + T_{latency,disk} + T_{LAN} + T_{latency,LAN} = 0.000001 + 0.005 + 0.00001 + 0.002 = 0.007011s.$$

2.2.4 What are the main differences between these two distributed access plans? Under which assumptions is one or the other better? *

```
SELECT *
FROM employee e, assignedTo a
WHERE e.#emp = a.#emp AND a.responsability = 'manager'
```

```
The database is:
————————————————————————————————

Employee(#emp, empName, degree)
S4:  E1 = Employee(#emp ≤ 'E3')
S3:  E2 = Employee(#emp > 'E3')
————————————————————————————————
AssignedTo(#emp, #proj, responsability, fullTime)
FK: #emp references Employee
S1:  A1 = AssignedTo(#emp ≤ 'E3')
S2:  A2 = AssignedTo(#emp > 'E3')
```

1. In plan A, firstly, the data is reduced by means of selections. Then, they are joined, increasing the data. Finally, these joined tables are transferred through the network.

2. In plan B, the plain tables are transferred to one node, and then they are combined, selected and joined. All in one machine.

There are several considerations that need to be taken into account to decide between these two approaches:

- In plan B, if the individual tables are large, it is possible that one node does not have enough capacity to store them. In these cases, we can only go for A.
- If that's not the case, everything would depend on how big the joined tables in plan A are compared to the raw tables, since the transfer cost is probably the dominating cost here. If the joined tables are smaller than the individual tables, then plan A would be better, else B.

Therefore, we need statistics to be able to decide which option is better in the context of the execution.

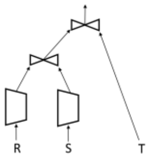
2.2.5 Compute the fragment query (data location stage) for the database setting and query below and find in how many ways we can assign the operations to the different sites.

The database setting is:

- **A distributed database with 5 sites (i.e. database nodes): S_1, S_2, S_3, S_4 and S_5.**
- **3 relations in the databaset: R, S and T.**
- **Each relation is horizontally fragmented in two fragments (we refer to them by the name of the relation and a subindex, for example: R_1, R_2). You can consider them to be correct (i.e. complete, disjoint and reconstructible).**
- **Each fragment is replicated at all sites.**
- **We have the following query**

$$Q_1 = \sigma(R) \bowtie \sigma(S) \bowtie T.$$

The process tree of the query is: *

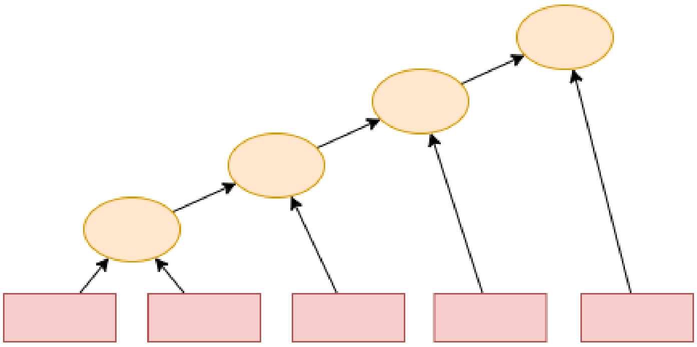


Notice that since each relation is horizontally fragmented into two fragments, we need to perform the union between those to get the relations. For example, $R = R_1 \cup R_2$. Therefore, the tree has 3 more nodes for each relation (one for each segment, and one for the union).

Thus, the tree is formed of 13 nodes, and there are 5 posible sites in which each node can execute. This means that there is a total of 5^{13} possible arrangements.

2.2.6 Consider a left-deep process tree corresponding to a query, where each internal node is a join, and every leaf a data source (e.g. relational table). Knowing that the tree contains 9 nodes (including leaves), the system has as much parallelism capacity as needed to run all the joins in pipelining mode (no other kind of parallelism is available), which is the occupancy if the overall cost of the serial query is 4 seconds? Explicit any assumption you need to make.

The tree is the following:



Then, there are 4 operators, so the occupancy is

$$O = \frac{T}{N} = \frac{4s}{4op} = 1s/op.$$

There are no stalls since we assumed an infinite parallelism capacity for pipelining.

2.3 Extra

2.3.1 Give five types of data or software resources that can be usefully shared. Give examples of their sharing as it occurs in practice in distributed systems.*

- Data: a distributed database.
- Information: Internet.
- Query: a distributed database.
- Processing: edge computing.
- RAM: supercomputers.

2.3.2 How might the clocks in two computers that are linked by a local network by synchronized without reference to an external time source? What factors limit the accuracy of the procedure you have described? How could the clocks in a large number of computers connected by the Internet be synchronized? Discuss the accuracy of that procedure. * (kinda)

Two computers linked by a local network can synchronize their clocks using a protocol like the **Network Time Protocol (NTP)**, even without reference to an external time source. One of the computers is selected as the master clock, and it periodically sends the current time to the other computer, which then adjusts its clock accordingly.

The accuracy of this procedure can be limited by network latency and clock drift. **Network latency** causes a delay between when the time message is sent and when it is received. **Clock drift** can cause the clock to speed up or slow down slightly, introducing a discrepancy over time.

To synchronize clocks in a large number of computers connected by the Internet, a **hierarchical time distribution method** can be used. At the top of the hierarchy is a primary server synchronized to an atomic clock or GPS. The primary server synchronizes time with secondary servers, which in turn synchronize with tertiary servers, and so on. Each layer of servers synchronizes with a large number of clients.

This procedure's accuracy can be affected by network latency and jitter, especially over long distances on the internet. However, protocols like NTP have built-in mechanisms to estimate and correct these errors.

2.3.3 Consider the implementation strategies for massively multiplayer online. In particular, what advantages do you see in adopting a single server approach for representing the state of the multiplayer game? What problems can you identify and how might they be resolved?

Advantages:

- Simplified state management: The game's state is stored and processed centrally, simplifying the logic needed to manage and update it.
- Consistent game state: Since there's only one source of truth, all players can see the same game state, reducing the chances of discrepancies and conflicts.

Disadvantages:

- Scalability issues: A single server may struggle to handle a large number of players, affecting game performance.
- Single point of failure: If the server fails, the game becomes inaccessible for all players.

To address these issues, techniques like server sharding (splitting the game world into different sections, each managed by a separate server) or using a distributed server architecture can be used.

2.3.4 A server program written in one language (for example, C++) provides the implementation of a BLOB object that is intended to be accessed by clients that may be written in a different language (for example, Java). The client and server computers may have different hardware, but all of them are attached to the Internet. Describe the problems due to each of the aspects of heterogeneity that need to be solved to make it possible for a client object to invoke a method on the server object.

- **Programming languages:** Due to the language differences, a common protocol like HTTP or a language-independent data format like JSON or XML might be necessary for communication. A method invocation from the Java client to the C++ server will have to be serialized into a standard format that both can understand.
- **Implementations by different developers:** Different developers might have different programming styles, conventions, and assumptions which could introduce bugs and incompatibilities. Standardized interfaces, thorough documentation, and stringent testing practices can help mitigate these issues.

2.3.5 The INFO service manages a potentially very large set of resources, each of which can be accessed by users throughout the Internet by means of a key (a string name). Discuss an approach to the design of the names of the resources that achieves the minimum loss of performance as the number of resources in the service increases. Suggest how the INFO service can be implemented so as to avoid performance bottlenecks when the number of users becomes very large. * (kinda)

Approach A: Hash Table

An efficient naming scheme could be a distributed hash table (DHT), where resource names (keys) are hashed to unique identifiers. This ensures a roughly uniform distribution of resources, allowing for efficient lookup even as the number of resources increases.

To avoid performance bottlenecks with a large number of users, the INFO service could be implemented in a distributed manner, where requests are load balanced across multiple servers. Each server would be responsible for a subset of the resources, reducing the load on any one server.

Approach B: Prefix Tree

If we want to accommodate a potentially infinite amount of users, then we can make use of a prefix tree, like the one used by Internet, in which each resource is positioned in a leave of the subtree that can be accessed very fast thanks to the prefixed structure.

2.3.6 A server process maintains a shared information object such as the BLOB object of exercise 2.3.4. Give arguments for and against allowing the client requests to be executed concurrently by the server. In the case that they are executed concurrently, give an example of possible 'interference' that can occur between the operations of different clients. Suggest how such interference may be prevented. *

Allowing concurrent client requests can increase the system's throughput but can also lead to data inconsistencies or conflicts if not properly managed. For example, if two clients try to modify the same data at the same time, one client's changes could overwrite the other's.

This interference can be prevented using concurrency control techniques such as locking (where a resource is locked when a client is modifying it and other clients have to wait), or optimistic concurrency control (where changes are checked for conflicts before being committed).

2.3.7 A service is implemented by several servers. Explain why resources might be transferred between them. Would it be satisfactory for clients to multicast all requests to the group of servers as a way of achieving mobility transparency for clients?

Resources might be transferred between servers for load balancing, fault tolerance, or data locality purposes.

While multicasting all requests to all servers might seem like a way to achieve mobility transparency, it would lead to significant overhead and could result in conflicts if multiple servers try to respond to the same request. Instead, a more efficient approach might be to use a load balancer or a similar mechanism to distribute requests among the servers.

3 Distributed File System

3.1 Theoretical questions

3.1.1 How does HDFS decide where to place the different chunks of a file? *

Randomly.

3.2 Problems

3.2.1 Given a table with ten blocks ($B = 10$) and 17 tuples per block ($R = 17$) for a total of 170 tuples, compare the cost of reading the whole table sequentially, against the cost of accessing seven random (potentially repeated) tuples in an unknown order. Assume that seek time is $12ms$, average rotation time is $3ms$ and transferring one block is $0.03ms$. You should consider which is the probability that two tuples accessed consecutively are actually in the same block, but ignore the presence of any cache or buffer pool mechanism. *

Sequentially:

$$T_{seq} = T_{seek} + T_{rot} + B \cdot T_{block} = 12 + 3 + 10 \cdot 0.03 = 15.3ms.$$

Seven random accesses:
First, we compute the probability that the tuple is in the same block as the previous one:

$$P_i = P\left(B_i = B_{i-1}\right) = \frac{1}{10}.$$

Then, the first block needs to be read for sure, the followings 6 will depend if they coincide or not:

$$T_{random} = [12 + 3 + 1 \cdot 0.03] + 6 \cdot [(1 - P_i) \cdot 3.03 + P_i \cdot 0] = 15.03 + 6 \cdot \frac{9}{10} \cdot 3.03 = 31.392ms.$$

The 3.03 is not a 15.03 because 10 blocks fit in one cylinder, so seek only happens the first time.

3.2.2 Consider a table stored in HDFS with the following characteristics:*

$size\,(T)$	256 MB
$\lvert T \rvert$	64
$size\,(row)$	4 MB
$cols\,(T)$	4
$size\,(cell)$	1 MB
$size\,(Header)$	0
$size\,(Footer)$	0

1. **Given the configuration parameters, how much space would you need to store it in a horizontal layout (i.e. Avro)?**

$size\,(MetaRow)$	0.1 MB
$size\,(MetaBody)$	0.1 MB

$$S\left(Body_{Hor}\right) = S\left(metaBody\right) + |T| \cdot \left[S\left(metaRow\right) + S\left(dataRow\right)\right] = 0.1 + 64 \cdot [0.1 + 4] = 262.5\ MB.$$

$$N_{chunks} = \left\lceil \frac{262.5}{128} \right\rceil = 3\ chunks.$$

2. **Given the configuration parameters, how much space would you need to store it in a hybrid layout (i.e. Parquet)?**

$size\,(MetaCol)$	0.2 MB
$size\,(MetaRowGroup)$	0.5 MB
$size\,(RowGroup)$	8 MB

$$UsedRowGroups = \frac{Cols\left(T\right) \cdot \left|T\right| \cdot S\left(cell\right)}{S\left(RowGroup\right) - S\left(metaRowGroup\right) - Cols\left(T\right) \cdot S\left(metaCol\right)} = \frac{4 \cdot 64 \cdot 1}{8 - 0.5 - 4 \cdot 0.2} = 38.209.$$

$$\begin{aligned} S\left(Body_{Hyb}\right) = & \left\lceil UsedRowGroups \right\rceil \cdot \left[S\left(metaRowGroup\right) + Cols\left(T\right) \cdot S\left(metaCol\right)\right] + Cols\left(T\right) \cdot \left|T\right| \cdot S\left(cell\right) \\ = & 39 \cdot \left[0.5 + 4 \cdot 0.2\right] + 4 \cdot 64 \cdot 1 = 306.7\ MB. \end{aligned}$$

3. **Given the configuration parameters, how much data would you need to retrieve it in a hybrid layout (i.e. Parquet) to project two columns?**

$size\left(MetaCol\right)$	0.2 MB
$size\left(MetaRowGroup\right)$	0.5 MB
$size\left(RowGroup\right)$	8 MB

$$S\left(projCols\right) = Proj\left(T\right) \cdot \frac{\left|T\right|}{UsedRowGroups\left(Hyb\right)} \cdot S\left(cell\right) = 2\frac{64}{38.21}1$$

$$\begin{aligned} S\left(Project_{Hyb}\right) = & S\left(Header_{Hyb}\right) + S\left(Footer_{Hyb}\right) + \\ & \left\lceil UsedRowGroups\left(Hyb\right)\right\rceil \cdot \left(S\left(metaRowGroup\right) + proj\left(T\right) + S\left(metaCol\right)\right) \\ & + UsedRowGroups\left(Hyb\right) \cdot S\left(projCols\right) \\ = & 0 + 0 + 39 \cdot \left(0.5 + 2 \cdot 0.2\right) + 38.21 \cdot \left[2\frac{64}{38.21}1\right] = 163.1\ MB. \end{aligned}$$

4. **Given the configuration parameters, how much data would you need to retrieve it in a hybrid layout (i.e. Parquet) to select one row (e.g. given its key) if the table is not sorted?**

$size\left(MetaCol\right)$	0.2 MB
$size\left(MetaRowGroup\right)$	0.5 MB
$size\left(RowGroup\right)$	8 MB

$$P\left(RGSelected\right) = 1 - \left(1 - SF\right)^{UsedRows\left(RG\right)} = 1 - \left(1 - \frac{1}{64}\right)^{\frac{64}{38.21}} = 0.026$$

$$UsedRowGroups\left(Select_{Hyb}\right) = UsedRowGroups\left(Hyb\right) \cdot P\left(RGSelected\right) = 0.993$$

$$\begin{aligned} S\left(Select_{Hyb}\right) = & S\left(Header_{Hyb}\right) + S\left(Footer_{Hyb}\right) + UsedRowGroups\left(Select_{Hyb}\right) \cdot S\left(RowGroup\right) \\ = & 0 + 0 + 0.993 \cdot 8 = 7.94\ MB. \end{aligned}$$

3.2.3 Given a file with C chunks and N nodes in the cluster, find the probability that HDFS uses all the nodes in case $C \geq N$.

There are

$$N^C$$

ways that the chunks can be distributed into the nodes.

Among this, the Stirling numbers of the second kind[2] give us the number of possibilities in which all nodes contain at least one chunk, therefore, it is

$$P = \frac{S\left(N, C\right) N!}{N^C},$$

here, the $N!$ comes because the stirling number does not take into account that the nodes are different, i.e., for the Stirling number, it is the same to store two chunks in node 1 and one in node 2, as to store one chunk in node 1 and two in node 2. Therefore, we need to take all possible arrangements into consideration. More explicitly, this formula ends up being

$$P = \frac{N! \sum_{i=1}^{C} \frac{(-1)^{C-i} i^n}{(C-i)!i!}}{N^C}.$$

[2] Visit https://en.wikipedia.org/wiki/Stirling_numbers_of_the_second_kindStirling Numbers of the second kind in Wikipedia for more information.

3.2.4 Consider an HDFS cluster with 100 data nodes, without replication. If I upload a file with 10 chunks and 10 disk blocks each, answer the following questions and briefly justify your answer.

1. **Which is the maximum number of machines actually contain data?**

   Since there is no replication, and the unit of distribution is the chunk, then at most 10 nodes can contain data.

2. **Which is the probability of the maximum number of machines actually contain data?**

   The formula in the previous exercise does not work in this case, because $C < N$. In this case the probability that all chunks end up in different machines is

$$P = \frac{100}{100} \times \frac{99}{100} \times \frac{98}{100} \times \frac{97}{100} \times \frac{96}{100} \times \frac{95}{100} \times \frac{94}{100} \times \frac{92}{100} \times \frac{91}{100} = 0.6282.$$

3.2.5 Given a file of 3.2 GB stored in an HDFS cluster of 50 machines, and containing 16×10^5 key-value pairs in a SequenceFile, estimate the execution time of a Spark job containing a single map transformation and an action storing the result in a file. Explicit any assumption you make and consider also the following parameters:

- **Chunk size: 128 MB (default)**
- **Replication factor: 3 (default)**
- **Map function (i.e. the parameter of the transformation) execution time: 10^{-3} sec/call (this is the only cost you have to consider)**
- **Save action execution time: 0 sec (do not consider this cost)**

There are

$$\frac{3.2GB}{128MB} = 25\ chunks$$

in the cluster. These chunks are replicated three times each, accounting for a total of 75 chunks in the system. However, replicated chunks are not processed unless some error occurs. Therefore, we consider only 25 chunks for execution.

There are

$$\frac{16 \cdot 10^5}{25} = 64000\ kv/chunk$$

and since there are more machines than chunks, we can execute all chunks in parallel. Therefore, since the time to execute one chunk is

$$T = T_{call} \cdot N_{kv} = 0.001 \cdot 64000 = 64s,$$

this is also the time to execute the whole job.

3.2.6 Given a file with 3.2 GB of raw data stored in an HDFS cluster of 50 machines, and containing 16×10^5 rows in a Parquet file; consider you have a query over an attribute 'A=constant' and this attribute contains only 100 different and equiprobable values. Assuming any kind of compression has been disabled, explicit any assumption you need to make and give the amount of raw data (i.e. do not count metadata) it would need to fetch from disk.

- **Replication factor: 3 (default)**
- **Chunk size: 128 MB (default)**
- **RowGroup size: 32 MB**

There are

$$\frac{3.2GB}{32MB} = 100\ RowGroups$$

in the cluster. Since the file contains 16×10^5 rows, each rowgroup contains

$$\frac{16 \times 10^5}{100} = 1600\ rows.$$

Since attribute A contains 100 different and equiprobable values, then we can consider that 16 rows will be fetched from each rowgroup.

In a chunk, there are 4 rowgroups

3.2.7 Consider a cluster of ten worker machines and a single coordinator, which contains a sequence file of 128 MB stored in HDFS with an OS block size of 32 KB in all machines and a chunk size of 64 MB. On the event of a client retrieving that file, give the number of control messages (do not consider data messages) that will travel the network in the case of (a) a client cache miss and in the case of (b) a client cache hit. Briefly justify both numbers.

- Client cache miss: In this case, when a client requests a file, the client first contacts the NameNode to get the location of the first block. The NameNode contacts the nodes in which each chunk is located, and then they answer the client. Therefore, there are 3 control messages, one from client to namenode, and two from namenode to datanodes.

- Client cache hit: In this case, the client knows the locations of the servers, so it asks directly them, accounting for just two control messages.

4 Key-Value Stores

4.1 Theoretical questions

4.1.1 Which is the main difference between the hash functions used in the linear hash and consistent hash algorithms?*

Linear hash adapts as the system grows, and there are two functions at once.

Consistent hash consists of a predefined static hash function.

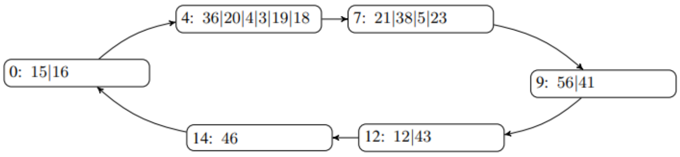
4.1.2 With respect to distributed systems, explain what is a distributed hash table (DHT), and provide a brief description of how consistent hashing guarantees balancing keys when adding new servers. *

A DHT is a hash table that is divided into buckets, having these buckets distributed.

Consistent hashing guarantees balancing by redistributing the keys at the moment of adding new servers, and relies on the big numbers and the assumption of well-constructed hash functions.

4.2 Exercises

4.2.1 Let's assume we have a Consistent hash with D = 16, and the hash function is simply the module of the IP address or the key, and suppose the current state of the consistent hash is (position in the ring:key|key|...):



1. **What happens when we insert objects 30 and 58? Draw the result.** *

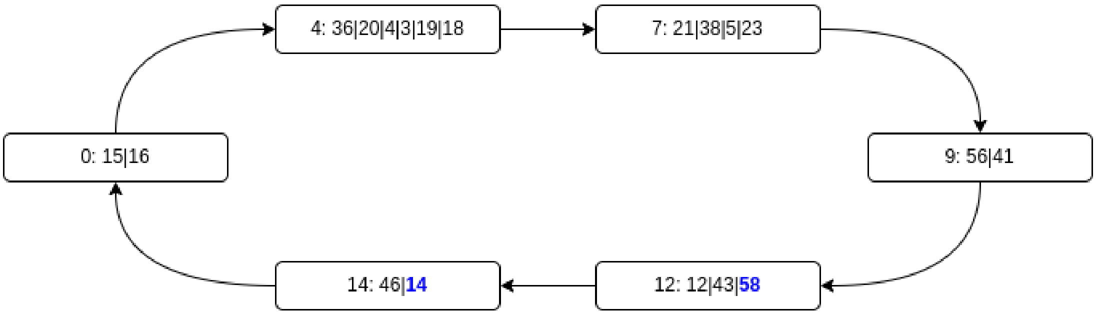
We have

$$30 \mod 16 = 14,$$

so 30 goes to server with IP 14. Also,

$$58 \mod 16 = 10,$$

so 58 goes to server with IP 12:



2. **What happens in the structure when we register a new server with IP address 37? Draw the result.** *

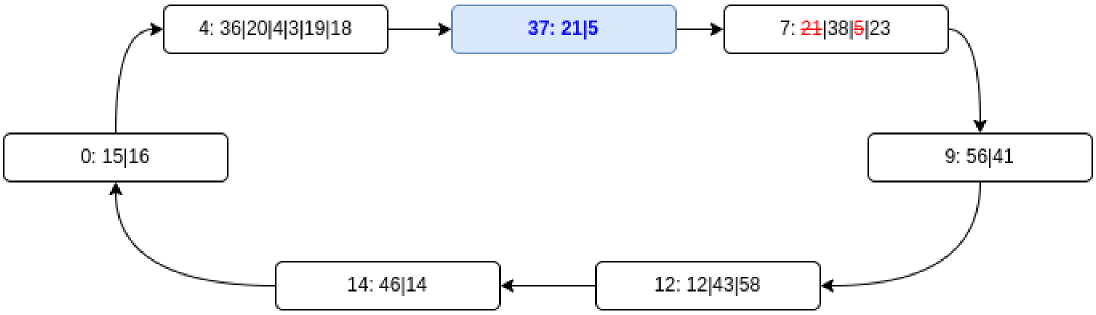
It is

$$37 \mod 16 = 5,$$

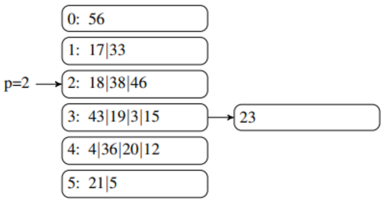
so the server is placed between servers 4 and 7. Now, we have to replace some elements from server 7:

$$\begin{aligned} 21 \mod 16 = 5 &\implies Replaced \\ 38 \mod 16 = 6 &\implies NotReplaced \\ 5 \mod 16 = 5 &\implies Replaced \\ 23 \mod 16 = 7 &\implies NotReplaced \end{aligned}$$

Therefore, the final result is



4.2.2 Let's suppose we have a Linear Hash and the hash function is simply the module of the key, the capacity of a bucket is only four entries, and curent state of the linear hash is (bucketID: key|key|...):

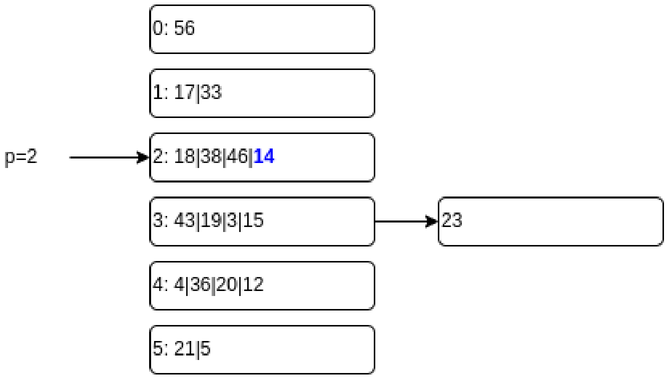


1. **What happens in the structure when we insert keys 14, 27, 37 and 44? Draw the result.** *

We have $n = 2$, so

$$14 \mod 2^2 = 2,$$

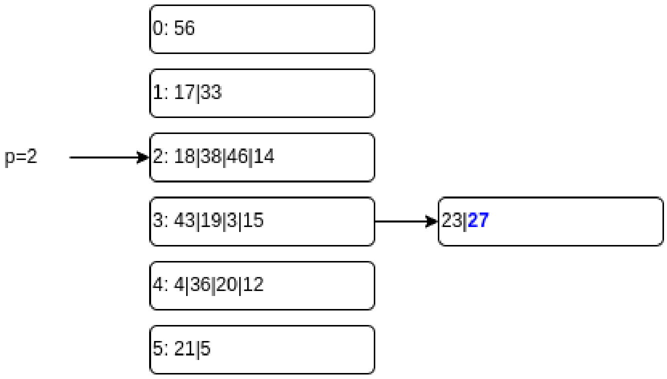
and the first insertion ends up as



Then,

$$27 \mod 2^2 = 3,$$

so



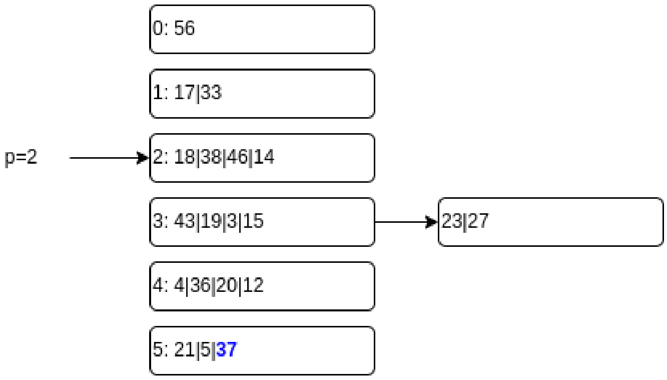
Then,

$$37 \mod 2^2 = 1,$$

since this is before the pointer, we compute also the next hash

$$37 \mod 2^3 = 5,$$

therefore



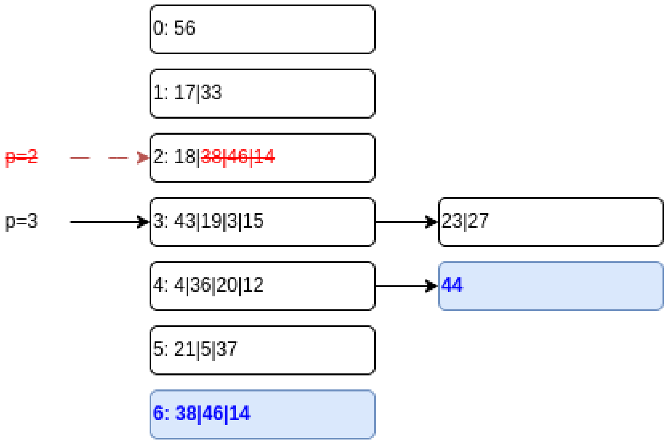
Then,

$$44 \mod 2^2 = 0,$$

since this is before the pointer, we compute also the next hash

$$44 \mod 2^3 = 4,$$

therefore it goes to 4. Since 4 is full, there is an overflow. We have to increase the pointer and recompute the values from the affected bucket (2):



4.2.3 Let's suppose that we have an LSM Tree that reached the threshold to consider the MemStore is full, and it contains four entries with format $[key, value, timestamp]$ needing 10 characters each. the content of the different structures is:

- **MemStore:** $[1, v, t50], [15, v, t49], [17, v, t47], [29, v, t48]$
- **Commit Log:** $[17, v, t47], [29, v, t48], [15, v, t49], [1, v, t50]$
- **SSTable:** $[13, v, t23], [25, v, t17], [35, v, t40], [59, v, t38]$
- **Index:** $[13, 0], [25, 30], [35, 60], [59, 90]$

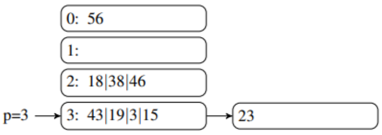
Assuming that the minimum size of an SSTable is 120 characters and on having two SSTables, a minor compactation is automatically triggered. Explicit the content of all structures once the compactation is done. *

- **MemStore:** $\emptyset$
- **Commit Log:** $\emptyset$
- **SSTable:** $[13, v, t23], [25, v, t17], [35, v, t40], [59, v, t38],[1, v, t50], [15, v, t49], [17, v, t47], [29, v, t48]$
- **Index:** $[13, 0], [25, 30], [35, 60], [59, 90],[1, 0], [15, 30], [17, 60], [29, 90]$

Then, they are merged and ordered by key:

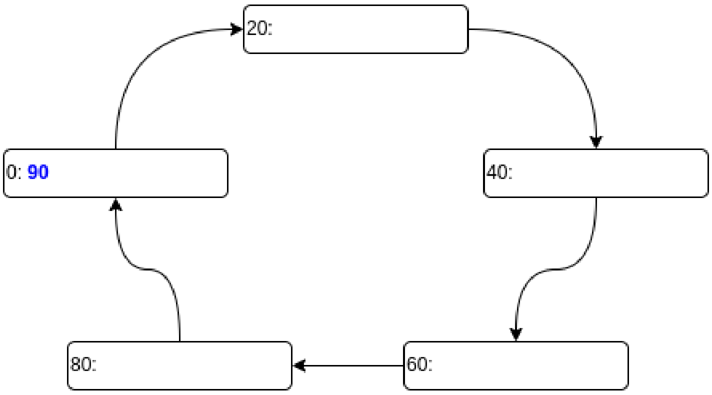
- **MemStore:** $\emptyset$
- **Commit Log:** $\emptyset$
- **SSTable:** $[1, v, t50], [13, v, t23], [15, v, t49], [17, v, t47], [25, v, t17], [29, v, t48], [35, v, t40], [59, v, t38]$
- **Index:** $[1, 0], [13, 30], [15, 60], [17, 90], [25, 120], [29, 150], [35, 180], [59, 210]$

4.2.4 Briefly explain what is wrong in this linear hash structure, or if you think it is right, explicitly say so:

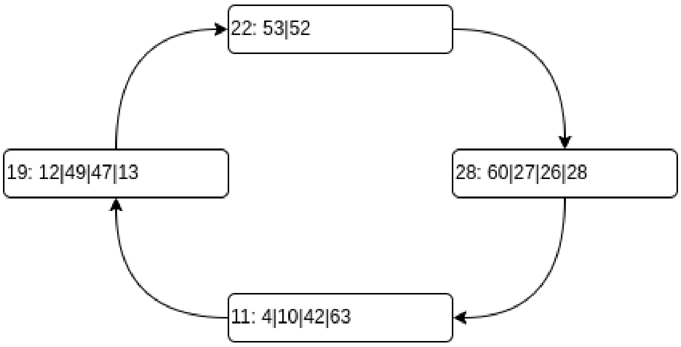


In this case, $n = 2$, and it is not possible to have the pointer pointing to a bucket between 2^n and 2^{n+1}, because the design of the algorithm makes this impossible. When $2^n + 1$ is reached, the pointer goes back to the first bucket and the process starts all over.

4.2.5 Suppose you have a hash function whose range has size 100 (i.e., D=100), and a Consistent Hash structure with 5 machines (M1,...,M5) whose identifiers map to values $h(M1) = 0, h(M2) = 20, h(M3) = 40, h(M4) = 60$ and $h(M5) = 80$. What happens if you have an object mapped to value $h(O) = 90$?



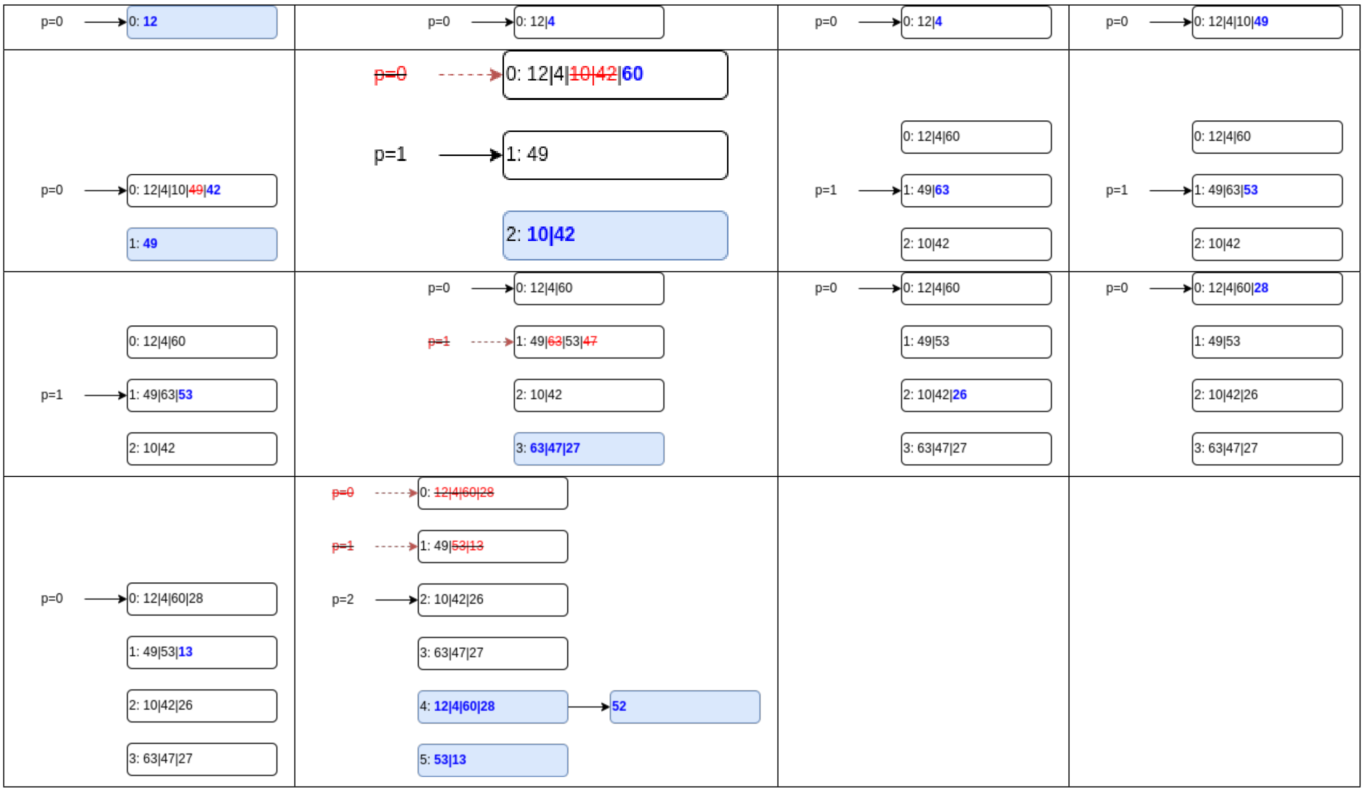
4.2.6 Given an empty consistent hash with $h(x) = x\%32$ (i.e., we directly take module 32 to both the keys and the bucket IDs), and unlimited capacity in each bucket, consider you have a cluster of four machines with IDs 19,22,75,92 and draw the result of inserting the following keys in the given order: 12, 4, 10, 49, 42, 60, 53, 47, 27, 26, 28, 13, 52.



4.2.7 Suppose you implement a system to store images in hundreds of machines with thousands of users using HBase with a single column-family. These images taken at time VT belong to a person P who tags each with a single subject S (e.g. family, friends, etc.) and are concurrently uploaded into the system at time TT in personal batches containing multiple pictures of different subjects taken at different times. Each person can then retrieve all his/her pictures of one single subject taken that were taken after a given time. Precisely define the key you would use if you exclusively prioritize (i.e., do not consider any other criteria): *

- **Load balancing on ingestion:** VT, S, P
  - **Assumptions made:** We want to distribute the data as much as possible.
- **Load balancing on querying:** TT, S, VT
  - **Assumptions made:** We want to distribute the queries as much as possible.
- **I/O cost (i.e., minimum blocks retrieved) on ingestion:** P, S, TT, VT
  - **Assumptions made:** We want to centralize data as much as possible
- **I/O cost (i.e., minimum blocks retrieved) on querying:** P, S, VT
  - **Assumptions made:** We want to centralize the queries as much as possible.

4.2.8 Given an empty linear hash with $f(x) = x$ (i.e., we directly apply the module to the keys), and a capacity of four keys per bucket, draw the result of inserting the following keys in the given order: 12, 4, 10, 49, 42, 60, 63, 53, 47, 27, 26, 28, 13, 52.



Notice that in the last step, when moving objects from 0 to 4, 4 becomes full, and it also overflows, so another bucket is created and the pertinent moves are done.

5 Document Stores

5.1 Problems

5.1.1 Optimize the performance of the following document design:

```
{"_id": 123,
"name": "Alberto"
"address": {
"street": "Jordi Girona",
"city": "Barcelona",
"zip_code": "08034"},
"telephone1": "93 4137889",
"telephone2": "93 4137840",
"telephone3": "123456789"
}
```

5.1.2 Optimize the performance of the following pipe:

```
db.partsupp.aggregate([
{"$sort": {"n_name": 1, "s_name": 1, "p_partkey": 1}},
{"$match": {"part.p_size": 5},
{"$project": {
"supplier.s_name": 1,
"supplier.s_phone": 1,
"supplier.s_comment": 1,
"_id": 0
}}
])
```

5.1.3 Assume you have a MongoDB collection which occupies 6 chunks evenly distributed in 3 shards (i.e., 2 chunks per shard). Being the document ID also the shard key, the chunk of a document is determined by means of a hash function. Assuming that accessing one document takes one time unit (existing indexes are used at no cost) and we have 6000 documents in the collection, k of which have value 'YYY' for attribute 'other', how many units would take the following operations[3]:

1. $\mathit{FindOne}(\{\_id : "XXX"\})$
2. $\mathit{Find}(\{\_id : \{\$in : [1, ..., 3000]\}\})$, being $[1, 6000]$ the range of existing IDs.
3. $\mathit{Find}(\{other : "YYY"\})$, being the attribute indexed.
4. $\mathit{Find}(\{other : "YYY\})$, being the attribute not indexed.

5.1.4 Assume you have a MongoDB collection which occupies 6 chunks unevenly distributed in 3 shards (i.e., 1, 2 and 3 chunks per shard, respectively). Being the document Id also the shard key, the chunk of a document is determined by means of a hash function. Assuming that accessing one document takes one time unit (existing indexes are used at no cost) and we have 6000 documents in the collection, k of which have value 'YYY' for attribute 'other', how many times units would take the following operations[4]:

1. $\mathit{FindOne}(\{\_id : "XXX"\})$
2. $\mathit{Find}(\{\_id : \{\$in : [1, ..., 3000]\}\})$, being $[1, 6000]$ the range of existing IDs.
3. $\mathit{Find}(\{other : "YYY"\})$, being the attribute indexed.
4. $\mathit{Find}(\{other : "YYY\})$, being the attribute not indexed.

[3] As typically in RDBMS optimizers, assume uniform distribution of values and statistical independence between pairs of attributes.

[4] As typically in RDBMS optimizers, assume uniform distribution of values and statistical independence between pairs of attributes.

5.1.5 Assume you have a MongoDB collection which occupies 6 chunks evenly distributed in 3 shards (i.e., 2 chunks per shard). Being the document Id also the shard key, the chunk of a document is determined by range. Assuming that accessing one document takes one time unit (existing indexes are used at no cost) and we have 6000 documents in the collection, k of which have value 'YYY' for attribute 'other', how many times units would take the following operations[5]:

1. $FindOne(\{\_id : "XXX"\})$
2. $Find(\{\_id : \{\$in : [1, ..., 3000]\}\})$, being $[1, 6000]$ the range of existing IDs.
3. $Find(\{other : "YYY"\})$, being the attribute indexed.
4. $Find(\{other : "YYY\})$, being the attribute not indexed.

5.1.6 Assume a MongoDB collection of JSON documents following the structure shown below. Indicate a sequence of stage operators that can be used in a pipeline to compute the top five departments in terms of number of employees such that its location has at least one performance_review greater than 7. In the example below, since both locations have performance reviewss greater than 7, manufacturing has a total of 90 employees, while billing and sales have 80 and 40 employees, respectively. You need only to provide the names of the stage operators and the sequence in which they are applied. You should not use the mapreduce feature of MongoDB, but rather the specific stage operators provided by MongoDB. You do not need to explicitly handle the case of ties (i.e., in the case of ties, you may return any of the tied documents to complete the top five).

```
{ "location": "Barcelona",
"budget": 3.000.000,
"departments": [ { "name": "sales", "employees": 40 }, { "name": "manufacturing",
"employees": 75 } ],
"performance_reviews": [ 7, 6, 3 ]
},
{ "location": "Brussels",
"budget": 5.000.000,
"departments": [ { "name": "manufacturing", "employees": 15 }, { "name": "billing
", "employees": 80 } ],
"performance_reviews": [ 6, 8, 7 ]
}
```

5.1.7 Imagine you want to design a system to maintain data about citizens and doubt whether to use HBase or MongoDB. Precisely, for each citizen, it will store: their personal data (with pID), their city data (with cID) and their employment data. We also know the workload (i.e., queries and frequency of execution) is as follows:

- **Q_1: average salary in Barcelona (50% frequency) - information obtained from the set of city and employment data.**
- **Q_2: average weight for you people (less than 18 years old) (45% frequency) - information obtained from the set of personal data.**
- **Q_3: number of VIPs (5% frequency) - information obtained from the set of personal data.**

Discuss your choice of technology and data model, without pre-computing the results of the queries. Clearly specify the structure of the data (i.e., tables/collections, keys, values, etc.), trade-offs, and assumptions made.

Given the queries and frequency of execution, both MongoDB and HBase can be used effectively, but each has its strengths and trade-offs.

[5] As typically in RDBMS optimizers, assume uniform distribution of values and statistical independence between pairs of attributes.

MongoDB is a document-oriented database that supports rich queries and secondary indexes. On the other hand, HBase is a wide-column store that scales horizontally and is designed for large tables with millions of columns, and it shines with range queries and scans.

Let's analyze the workload:

- Q1: average salary in Barcelona (50% frequency): This query needs to combine city and employment data. As MongoDB supports joining data from different collections, it might have an advantage here. In HBase, you may need to denormalize the data or do client-side joining, which can be less efficient.
- Q2: average weight for young people (less than 18 years old) (45% frequency): This query requires a secondary index on age to perform efficiently. MongoDB, with its built-in support for secondary indexes, may have an advantage. However, HBase can also support secondary indexes with some additional setup and design considerations.
- Q3: number of VIPs (5% frequency): This query requires searching personal data based on a specific attribute, which is doable efficiently in both MongoDB and HBase.

Given the workload and the need for complex queries, MongoDB might be a slightly better choice due to its rich querying capabilities and built-in support for secondary indexes. However, if the volume of data is enormous and you need extreme scalability, HBase might be a better choice due to its distributed nature and high write throughput.

- Data model for MongoDB:
  - Personal data collection: { "_id" : pID, "name": "", "age": , "weight": , "VIP_status": }
  - City data collection: { "_id" : cID, "city_name": "", "citizen_ids": [list of pID] }
  - Employment data collection: { "_id": uniqueID, "pID": , "salary": , "job_title": }
- Data model for HBase:
  - Personal data table: Row key: pID, Column Family: Personal_Info: {"name", "age", "weight", "VIP_status"}
  - City data table: Row key: cID, Column Family: City_Info: {"city_name", "citizen_ids"}
  - Employment data table: Row key: uniqueID, Column Family: Employment_Info: {"pID", "salary", "job_title"}

5.1.8 Analyze (i.e., briefly give pros and cons) the following JSON design compared to other equivalent JSON designs from the three perspectives:

```
"BID-PRODUCT":{
"B_ID": int(4), "B_PRICE": int(4), "U_ID": int(4),
"PRODUCT": {
"P_ID": int(4), "P_INFO": varchar(100)
}}
"PRODUCT-SELLER-REGION": {
"P_ID": int(4), "P_INFO": varchar(100),
"USER": {
"U_ID": int(4), "U_F_NAME": varchar(20),
"REGION": {
"R_ID": int(4), "R_NAME": varchar(10)
}}}
"PRODUCT-COMMENTS": {
"P_ID": int(4), "P_INFO": varchar(100),
"COMMENTS": [{
"C_ID": int(4), "C_TITLE": varchar(20), "U_ID": int(4)
}]}
```

1. **Read (a.k.a Query)**
   (a) **Pros:**

   (b) **Cons:**

2. **Update**

   (a) **Pros:**

   (b) **Cons:**

3. **Memory (a.k.a. Space)**

   (a) **Pros:**

   (b) **Cons:**

6 New Relational Architecture

6.1 Theoretical questions

6.1.1 Briefly explain the concept of associativity in the context of memory usage. Identify the benefit of high (respectively low) associativity.

6.2 Problems

6.2.1 Assume you have a SanssouciuDB table T stored row-wise, which ocupies 300 blocks and contains tuples with three variable length attributes [$\underline{\text{A}}$,B,C] (underlined attribute is declared to be the primary key of the table). Assume there is no index and give the average cost of each query assuming accessing one block is one second. *

- **SELECT A,B,C FROM T WHERE A=x; (being x a constant)**

  The probability of $A = x$ in block k is $\frac{1}{300}$. Therefore, the average cost is

$$C = \sum_{k=1}^{300}\left(\sum_{j=1}^{k} c(j) \cdot P(A = x|j)\right) = \sum_{k=1}^{300}\left(\sum_{j=1}^{k} 1\frac{1}{300}\right) = \frac{1}{300}\sum_{k=1}^{300} k = \frac{1}{300}\frac{300 \cdot 301}{2} = 150.5s.$$

- **SELECT SUM(B) FROM T;**

  In this case, we have to access all of them:

$$C = 300s.$$

6.2.2 Assume you have a SanssouciuDB table T stored column-wise, which ocupies 300 blocks and contains tuples with three variable length attributes [$\underline{\text{A}}$,B,C] (underlined attribute is declared to be the primary key of the table). Assume there is no index, storage of each attribute uses exactly the same space after compression (i.e., 100 blocks), and run length encoding has been applied for non-key attributes storing ending row position per run. Give the average cost of each query assuming accessing one block is one second and explicit any other assumption you make. *

- **SELECT A,B,C FROM T WHERE A=x; (being x a constant)**

  Find x in A is

$$C_A = \frac{101}{2} = 50.5s.$$

  Now, we know the position, and since the encoding is done with ending row position, we can access it directly. Therefore, $C_B = C_C = 1s$ and the total cost is

$$C = 52.5s.$$

- **SELECT SUM(B) FROM T;**

  We need to traverse all of B, so $C = 100s$.

6.2.3 Assume that there is a table T with attributes [A,B,C] (the underlined attribute is the primary key of the table), which occupies 300 disk blocks of 8 KB each, with 100 B per row. All three attributes require the same space (even in case of compression). Supposing that there is not any index, which would be the minimum amount of memory required to process (do not consider the memory required to store the final output of the query) the following query in either row storage or column storage (briefly justify your answer and explicit any assumption you make).

SELECT A FROM T WHERE B='x' and C='y'; *

- **Row storage:** 1 Block. We can have just one block in memory at a time, an check the conditions inside each block, storing the wanted results accordingly.
- **Column storage:** 3 Blocks. In this case, we need to store a block in memory for each column, to be able to make the three comparisons for each row.

6.2.4 Without considering compression and assuming a disk block size of 8 KB, is there any case where a query retrieving all tuples, but only half of the equally-sized attributes of a relaitonal table performs better in row storage without any kind of vertical partitioning than in columnar storage? Numerically justify your answer.

6.2.5 Represent the given column with dictionary and run-length encoding storing and row position.

Table
A
A
B
B
B
C
A
A

Dictionary
A
B
C

End Row	**Code**
2	A
5	B
6	C
8	A

6.2.6 Given the two columns of a table represented with run-length encoding using dictionary, give the position of the row(s) that fulfill the predicate "Charlie AND Beta". Briefly explain how a column store would obtain them.

Dictionary
Bravo
Charlie

#Values	Dictionary Positions
2	1
3	0
3	2

Dictionary
Alpha
Beta

#Values	Dictionary Positions
1	1
5	0
2	2

The first two values in the first column are Charlie, and no more Charlie's are encoutered.

In the second column, then, we have to check these two positions. Since Beta is in the first position, and is encountered only once, then this one is the only solution: row 0.

7 Distributed Processing Frameworks

7.1 MapReduce in use: Theoretical questions

7.1.1 Classify a MapReduce system as either query-shipping (i.e., the evaluation of the query is delegated to the site where the data is stored), data-shipping (i.e., the data is moved from the stored site to the site executing the query) or hybrid (i.e., both query and data shipping). Clearly justify your answer.

A MapReduce system can be classified as a hybrid system, as it involves both data-shipping and query-shipping. In the Map phase, the query (map function) is shipped to the data nodes (query-shipping), where it is executed on the local data. Then, during the shuffle and sort phase, the intermediate data output by the map function is moved to the reduce nodes (data-shipping), where it is further processed by the reduce function.

7.2 MapReduce in use: Problems

7.2.1 Consider the following implementation of the Cartesian Product with MapReduce ($\oplus$ stands for concatenation) and answer the questions accordingly:

$$T \times S \implies \begin{cases} map\left(key\ k, value\ v\right) \mapsto & \\ \left[\left(h_T\left(k\right) \mod D, k \oplus v\right)\right] & if\ input\left(k \oplus v\right) = T \\ \left[\left(0, k \oplus v\right), ..., \left(D-1, k \oplus v\right)\right] & if\ input\left(k \oplus v\right) = S \\ reduce\left(key\ ik, vset\ ivs\right) \mapsto & \\ \left[\begin{array}{c} crossproduct\left(T_{ik}, S\right) \mid \\ T_{ik} = \left\{iv \mid iv \in ivs \wedge input\left(iv\right) T\right\}, \\ S = \left\{iv \mid iv \in ivs \wedge input\left(iv\right) S\right\} \end{array}\right] & \end{cases}.$$

1. **Which is the relationship of D with parallelism and scalability, if any?**

   D is directly proportional to the level of parallelism in the MapReduce computation. D stands for the number of reducers, and increasing D allows for more tasks to be executed in parallel, enhancing the computation's scalability.

2. **Which is the optimal value for D?**

   The optimal value of D depends on the specific use case, the size and characteristics of the data, and the resources available in the cluster. In general, it's best to have D set so that each reducer has a manageable amount of data to process and the workload is evenly distributed across all reducers. A reasonable value for D could be $2 \times N_{machines}$, so that every machine can be activated, and if some are faster, they can still keep processing.

7.2.2 Provide the MapReduce pseudo-code implementation of the following relational operators. You can use "$\oplus$" symbol for concatenation, "$prj_{a_{i_1}...a_{i_n}}(t)$" to get attributes "$a_{i_1}...a_{i_n}$" in t, "$crossproduct\left(S_1, S_2\right)$" to perform the cross product of two sets of rows, and assume the "key" parameter contains the PL of the table and the "value" one all the others.

- **Aggregation ($\gamma_{A,f(B)}\left(T\right)$)**

$$\gamma_{A,f(B)}\left(T\right) \implies \begin{cases} map\left(key\ k, value\ v\right) \mapsto \\ \left(v\left(A\right), v\left(B\right)\right) \\ reduce\left(key\ ik, vset\ ivs\right) \mapsto \\ \left(ik, f\left[ivs\right]\right) \end{cases}$$

- **Selection ($\sigma_P\left(T\right)$)**

$$\sigma_P\left(T\right) \implies \begin{cases} maps\left(key\ k, value\ v\right) \mapsto \\ \left(k, v\left(P\right)\right) \\ reduce\left(key\ ik, vset\ ivs\right) \mapsto \\ \left(ik, ivs\right) \end{cases}$$

Note that in this case the reduce function does not do anything, so in fact it is not needed.

- **Join** ($T \bowtie_{T.A=S.B} S$)

$$T \bowtie_{T.A=S.B} S \implies \begin{cases} map\left(key\ k, value\ v\right) \mapsto & \\ \left[\left(v\left(A\right), k \oplus v\right)\right] & if\ input\left(k \oplus v\right) = T \\ \left[\left(v\left(B\right), k \oplus v\right)\right] & if\ input\left(k \oplus v\right) = S \\ reduce\left(key\ ik, vset\ ivs\right) \mapsto & \\ \begin{bmatrix} join\left(T_i, S\right) | \\ T_{ik} = \{iv|iv \in ivs \wedge input\left(iv\right) T\}, \\ S = \{iv|iv \in ivs \wedge input\left(iv\right) S\} \end{bmatrix} & \end{cases}$$

- **Union** ($T \cup S$)

$$T \cup S \implies \begin{cases} map\left(key\ k, value\ v\right) \mapsto \\ \left(k, v\right) \end{cases}$$

There is no need for reduce, since the results will be united.

- **Difference** ($T \setminus S$)

$$T \setminus S \implies \begin{cases} map\left(key\ k, value\ v\right) \mapsto & \\ \left[\left(v, T\right)\right] & if\ input\left(k \oplus v\right) = T \\ \left[\left(v, S\right)\right] & if\ input\left(k \oplus v\right) = S \\ reduce\left(key\ ik, vset\ ivs\right) \mapsto & \\ \emptyset & if\ size\left(ivs\right) > 1 \\ ik & if\ ivs = \left[T\right] \end{cases}$$

- **Intersection** ($T \cap S$)

$$T \cap S \implies \begin{cases} map\left(key\ k, value\ v\right) \mapsto & \\ \left[\left(v, T\right)\right] & if\ input\left(k \oplus v\right) = T \\ \left[\left(v, S\right)\right] & if\ input\left(k \oplus v\right) = S \\ reduce\left(key\ ik, vset\ ivs\right) \mapsto & \\ ik & if\ \left[T, S\right] \subset ivs \\ \emptyset & else \end{cases}$$

7.2.3 In relational algebra, the antijoin operator ($\triangleright$) is defined as the complement of the semijoin on the primary keys (PKs). Formally, assuming A and B are the PLs of R and S, respectively, then

$$R \triangleright S = R \setminus R \ltimes_{A=B} S$$

Provide the MapReduce pseudo-code implementation of the antijoin operator. Assume the existence of the operator $\oplus$ to concatenate strings, $prj_{att}\left(s\right)$ to project attribute att from the tuple s, and $input\left(s\right)$ to decide the origin (i.e., R or S) from a tuple s.

$$R \triangleright S \implies \begin{cases} map\left(key\ k, value\ v\right) \mapsto & \\ \left[\left(v\left(A\right), k \oplus v \oplus R\right)\right] & if\ input\left(k \oplus v\right) = R \\ \left[\left(v\left(B\right), k \oplus v \oplus S\right)\right] & if\ input\left(k \oplus v\right) = S \\ reduce\left(key\ ik, vset\ ivs\right) \mapsto & \\ ivs\left(v\right) & if\ \left[ivs\left(-1\right)\right].unique == R \\ \emptyset & else \end{cases}$$

7.3 MapReduce internals: Problems

7.3.1 Assume the following MapReduce program:

```java
public void map(LongWritable key, Text value) {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
write(new Text(tokenizer.nextToken()), new IntWritable(1));
}
}
public void reduce(Text key, Iterable<IntWritable> values) {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
write(key, new IntWritable(sum));
}
```

Consider the following dataset:

- Block0: “a b b a c | c d c e a”
- Block1: “a b d d a | b b c c d”

Simulate the execution of the MapReduce code given the following configuration:

- The map and reduce functions are those of the wordcount. The combine function shares the implementation of the reduce.
- One Split is one block.
- The “|” divides the records inside each block. We have two records per block.
- Hadoop is configured with the parameter $dfs.replication = 1$.
- We can keep four pairs [key, value] per spill
- We have two mappers and two reducers:
  - $Machine0$ contains block0, runs mapper0 and reducer0
  - $Machine1$ contains block1, runs mapper1 and reducer1
- The hash functions used to shuffle data to the reducers uses the correspondence:
  - $\{b, d, f\} \to 0$
  - $\{a, c, e\} \to 1$

Fill the gaps in each step (numbers correspond to the phase in the MapReduce algorithm): *

1. $Machine0$ contains 1 block(s). $Machine1$ contains 1 block(s).
2. We keep 1 replica(s) (including the master copy) per block.
3. We have 1 split(s) per machine.
4. Mapper0 reads 2 records. Mapper1 reads 2 records.
5. Memory content during each spill in $Machine0$:

Spill1	Spill2	Spill3	Spill4
[a,1] [b,1] [b,1] [a,1]	[c,1] [c,1] [d,1] [c,1]	[e,1][a,1][][]	[][][][]

Spills in $Machine1$:

Spill1	Spill2	Spill3	Spill4
[a,1] [b,1] [d,1] [d,1]	[a,1] [b,1] [b,1] [c,1]	[c,1][f,1][][]	[][][][]

6. **Three substeps:**

   (a) **Partitions in** *Machine*0**:**

Spill1		Spill2		Spill3	
Partition0	Partition1	Partition0	Partition1	Partition0	Partition1
[b,1] [b,1] [] []	[a,1] [a,1] [] []	[d,1] [] [] []	[c,1] [c,1] [c,1] []	[] [] [] []	[e,1] [a,1] [] []

   Partitions in *Machine*1**:**

Spill1		Spill2		Spill3	
Partition0	Partition1	Partition0	Partition1	Partition0	Partition1
[b,1] [d,1] [d,1] []	[a,1] [] [] []	[b,1] [b,1] [] []	[a,1] [c,1] [] []	[f,1] [] [] []	[c,1] [] [] []

   (b) **Partitions in** *Machine*0**:**

Spill1		Spill2		Spill3	
Partition0	Partition1	Partition0	Partition1	Partition0	Partition1
[b,2] [] [] []	[a,2] [] [] []	[d,1] [] [] []	[c,3] [] [] []	[] [] [] []	[a,1] [e,1] [] []

   Partitions in *Machine*1**:**

Spill1		Spill2		Spill3	
Partition0	Partition1	Partition0	Partition1	Partition0	Partition1
[b,1] [d,2] [] []	[a,1] [] [] []	[b,2] [] [] []	[a,1] [c,1] [] []	[f,1] [] [] []	[c,1] [] [] []

   (c) **Files in** *Machine*0**:**

File0.0	File0.1	File0.2
[b,2] [] [] []	[a,2] [] [] []	[d,1] [] [] []

File0.3	File0.4	File0.5
[c,3] [] [] []	[] [] [] []	[a,1] [e,1] [] []

   Files in *Machine*1**:**

File1.0	File1.1	File1.2
[b,1] [d,2] [] []	[a,1] [] [] []	[b,2] [] [] []

File1.3	File1.4	File1.5
[a,1] [c,1] [] []	[f,1] [] [] []	[c,1] [] [] []

7. **Merges in** *Machine*0**:**

Merge0	Merge1
[b,2] [d,1] [] [] []	[a,3] [c,3] [e,1] [] []

   Merges in *Machine*1**:**

Merge0	Merge1
[b,3] [d,2] [f,1] [] []	[a,2] [c,2] [] [] []

8. **Files in** *Machine*0**:**

File0.0	File0.1
[b,2] [d,1] [] [] []	[a,3] [c,3] [e,1] [] []

   Files in *Machine*1**:**

File1.0	File1.1
[b,3] [d,2] [f,1] [] []	[a,2] [c,2] [] [] []

9. **Reducer0 reads file0.0 from** *Machine*0 **and file1.0 from** *Machine*1**. Reducer1 reads file0.1 from** *Machine*0 **and file1.1 from** *Machine*1**.**

10. **Merges in $Machine0$:**

Merge0	Merge1
[b,{2,3}] [d,{1,2}] [f,{1}] [] []	[] [] [] [] []

    Merges in $Machine1$:

Merge0	Merge1
[a,{3,2}] [c,{3,2}] [e,{1}] [] []	[] [] [] [] []

11. **Reduce function is executed 3 times in $Machine0$. Reduce function is executed 3 times in $Machine1$.**

12. **Files in $Machine0$:**

Output0
[b,5] [d,3] [f,1] [] []

    Files in $Machine1$:

Output1
[a,5] [c,5] [e,1] [] []

7.3.2 Let's suppose that we have a cluster of 100 machines and a MapReduce job with 1 000 000 key-value pairs in the input that generate 100 000 pairs in the output. Assume that both map and reduce functions generate one pair in the output per call. Assuming the reduce function is commutative and associative, is it worth to use the combine function? Briefly justify your answer.

The setup implies that on average each key generated in the map phase, appears 10 times. Therefore, using the combine function locally before combining all the results is probably not worthy, specially if we have a larger number of workers. For example, if we have 50 workers, the probability that several values for the same key are in the same machine is pretty low, so it would just be an overhead on the computations. Maybe, if we are sure that the generated keys will be in the same machine for some reason (for instance if the distribution is based in this key), then it might be good to consider introducing the combining function.

Notice also that the reduce function being associative and commutative ensures that the combine function can be trivially implemented.

7.4 Spark in use: Problems

7.4.1 Consider a file (*wines.txt*) containing the following data:

```
wines.txt
type 1,2.064254
type 3,2.925376
type 2,2.683955
type 1,2.991452
type 2,2.861996
type 1,2.727688
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve the minimum value per type. Do not use SQL and minimize the use of other Python libraries or code. Save the results in *output.txt*. *

```
#RDD-based
wines = sc.textFile("/content/drive/My Drive/BDM_Spark/sources/wines.txt")
pairs = wines.map(lambda t: (t.split(",")[0],t.split(",")[1]))
result = pairs.reduceByKey(lambda a,b: min(a,b))
result.collect()

```

```
#Dataframe-based
import pyspark.sql.functions as fn
source = sqlContext.read.format("csv").load("/content/drive/My Drive/BDM_Spark/sources/wines.txt", header="false", inferSchema="true")
result = source.groupBy("_c0").agg(fn.min("_c1").alias("min"))
result.show()
```

7.4.2 Consider two files containing the following data:

```
Employees.txt
EMP1;CARME;400000;MATARO;DPT1
EMP2;EUGENIA;350000;TOLEDO;DPT2
EMP3;JOSEP;250000;SITGES;DPT3
EMP4;RICARDO;250000;MADRID;DPT4
EMP5;EULALIA;150000;BARCELONA;DPT5
EMP6;MIQUEL;125000;BADALONA;DPT5
EMP7;MARIA;175000;MADRID;DPT6
EMP8;ESTEBAN;150000;MADRID;DPT6

Departments.txt
DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
DPT3;MARKETING;1;PAU CLARIS;BARCELONA
DPT4;MARKETING;3;RIOS ROSAS;MADRID
DPT5;VENDES;1;MUNTANER;BARCELONA
DPT6;VENDES;1;CASTELLANA;MADRID
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve for each employee his/her department information. Do not use SQL and minimize the use of other Python libraries or code. Save the results in ***output.txt*****.** *

```
#RDD-based
emp = sc.textFile("/content/drive/My Drive/BDM_Spark/sources/employees.txt")
dpt = sc.textFile("/content/drive/My Drive/BDM_Spark/sources/departments.txt")
emp = emp.map(lambda t: (t.split(";")[4],t))
dpt = dpt.map(lambda t: (t.split(";")[0],t))
dpt_emp = emp.join(dpt)

dpt_emp.saveAsTextFile("/content/drive/My Drive/BDM_Spark/sources/output.txt")

# DF
sc1 = spark.read.format("csv").option("delimiter",";").load("/content/drive/My Drive/BDM_Spark/sources/employees.txt", header = "false", inferSchema = "true")
sc2 = spark.read.format("csv").option("delimiter", ";").load("/content/drive/My Drive/BDM_Spark/sources/departments.txt", header = "false", inferSchema = "true")
emp = sc1.toDF("eID", "eName", "eSalary", "eCity", "eDpt","eProj")
emp.show()
dpt = sc2.toDF("dID", "dArea", "dNumber", "dStreet", "dCity")
dpt.show()
result = emp.join(dpt, emp.eDpt == dpt.dID, "inner")
result.show()
result.write.save(path="/content/drive/My Drive/BDM_Spark/sources/output", format="csv", header=False)
```

7.4.3 Consider an error log file (*log.txt*) like the one below:

```
log.txt
20150323;0833;ERROR;Oracle
20150323;0835;WARNING;MySQL
20150323;0839;WARNING;MySQL
20150323;0900;WARNING;Oracle
20150323;0905;ERROR;MySQL
20150323;1013;OK;Oracle
20150323;1014;OK;MySQL
20150323;1055;ERROR;Oracle
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, just the kind of operation and main parameters) you'd need to retrieve the lines corresponding to both *errors* **and** *warnings*, **but adding** *Important* : **at the begginning of those of** *errors* **(i.e., only** *errors*). **Do not use SQL and minimize the use of other Python libraries or code. Save the results in** *output.txt*. *

```
#RDD
log = sc.textFile("/content/drive/My Drive/BDM_Spark/sources/log.txt").persist()
error = log.filter(lambda t: t.split(";")[2] in ("ERROR"))
warning = log.filter(lambda t: t.split(";")[2] in ("WARNING"))
important = error.map(lambda t: t + ";IMPORTANT")
result = warning.union(important)
result.saveAsTextFile("/content/drive/My Drive/BDM_Spark/sources/ex313_output.txt")

#DF
source = spark.read.format("csv").option("delimiter",";").load("/content/drive/My Drive/BDM_Spark/sources/log.txt", header = "false", inferSchema = "true")
errors = source.where(fn.locate("ERROR", source._c2)!=0)
warnings = source.where(fn.locate("WARNING",source._c2)!=0)
redef = errors.select(fn.concat(fn.lit("important: "),fn.col("_c0")),"_c1","_c2","_c3")
result = warnings.unionAll(redef)
result.coalesce(1).write.save(path="/content/drive/My Drive/BDM_Spark/sources/ex313_output_df", format="csv", header=False)
result.coalesce(1).show()
```

7.4.4 Given two files containing the following kinds of data:

```
Employees.txt with fields:
  EmployeeID; EmployeeName; YearlySalary; CityOfResidence; SiteOfWork

  EMP4;RICARDO;250000;MADRID;DPT4
  EMP5;EULALIA;150000;BARCELONA;DPT5
  EMP6;MIQUEL;125000;BADALONA;DPT5
  EMP7;MARIA;175000;MADRID;DPT6
  EMP8;ESTEBAN;150000;MADRID;DPT6
  ...


Departments.txt with fields:
  SiteID; DepartmentName; StreetNumber; StreetName; City

  DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
  DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
  DPT3;MARKETING;1;PAU CLARIS;BARCELONA
  DPT4;MARKETING;3;RIOS ROSAS;MADRID
  ...
```

Consider the following PySpark code and answer the questions below:

```
source1 = spark.read.format("csv").load("employees.txt", header='false', inferSchema='true', sep=";")
source2 = spark.read.format("csv").load("departments.txt", header='false', inferSchema='true', sep=";")
A = source1.toDF("eID","eName","eSalary","eCity","eDpt","eProj")
B = source2.toDF("dID","dArea","dNumber","dStreet","dCity")
C = A.select(A.eCity.alias("city"))
D = B.select("dArea")
E = D.crossJoin(C)
F = B.select("dArea",B.dCity.alias("city"))
G = E.subtract(F)
H = G.select("dArea")
result = D.subtract(H)
```

1. **State in natural language the corresponding query it would answer.** *

   Get all department areas located in a city in which at least one employee works, even in a different department.

2. **Clearly indicate any mistake or improvement you can fix/make in the code. For each of them give (1) the line number, (2) pseudo-code to implement the fix, and (3) brief rationale.**

   (a) There is no action, so we could add it at the end of the computations.

   (b) Line 3 and 4: Instead of reading the CSV file without headers and then renaming the columns, we can directly provide the schema while reading the file. This will make the code cleaner and improve performance as well.

```
schema1 = StructType([
   StructField("eID", StringType()),
   StructField("eName", StringType()),
   StructField("eSalary", FloatType()),
   StructField("eCity", StringType()),
   StructField("eDpt", StringType())
])
schema2 = StructType([
   StructField("dID", StringType()),
   StructField("dArea", StringType()),
   StructField("dNumber", IntegerType()),
   StructField("dStreet", StringType()),
   StructField("dCity", StringType())
])
A = spark.read.format("csv").schema(schema1).load("employees.txt", sep=";")
B = spark.read.format("csv").schema(schema2).load("departments.txt", sep=";")
```

   (c) Line 10: The subtract function is an expensive operation as it requires a full shuffle of the data. Instead, we can perform a left anti-join operation to achieve the same result but in a more efficient way.

```
H = E.join(F, on=[’dArea’, ’city’], how=’left_anti’)
```

   (d) Finally, the use of crossJoin in Line 9 can lead to very large intermediate data and should be avoided if possible. A more effective method would be to join the data on the common attribute ’city’ instead. This would give us the department areas that are located in the cities where at least one employee resides, without generating all possible combinations first.

```
E = D.join(C, D.dArea == C.city)
```

7.4.5 Given two files containing the following kinds of data:

```
Employees.txt with fields:
  EmployeeID; EmployeeName; YearlySalary; CityOfResidence; SiteOfWork

  EMP1;RICARDO;250000;MADRID;DPT1
  EMP2;EULALIA;150000;BARCELONA;DPT2
  EMP3;MIQUEL;125000;BADALONA;DPT3
  EMP4;MARIA;175000;MADRID;DPT4
  EMP5;ESTEBAN;150000;MADRID;DPT3
  ...
Departments.txt with fields:
  SiteID; DepartmentName; StreetNumber; StreetName; City

  DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
  DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
  DPT3;MARKETING;1;PAU CLARIS;BARCELONA
  DPT4;MARKETING;3;RIOS ROSAS;MADRID
  ...
```

Give a sequence of Spark operations in pseudo-code (resembling PySpark) to obtain for each city where employees that work in a site of a department in Barcelona live, the sum of the salaries of those employees. The result for the exemplary data would be:

```
MADRID;400000
BARCELONA;125000
...
```

7.4.6 Consider three files containing the following kinds of data:

```
Employees.txt
  EMP1,CARME,400000,MATARO,DEPT1,PROJ1
  EMP2,EULALIA,150000,BARCELONA,DEPT2,PROJ1
  EMP3,MIQUEL,125000,BADALONA,DEPT1,PROJ3

Projects.txt
  PROJ1,IBDTEL,TV,1000000
  PROJ2,IBDVID,VIDEO,500000
  PROJ3,IBDTEF,TELEPHONE,200000
  PROJ4,IBDCOM,COMMUNICATIONS,2000000

Departments.txt
  DEPT1,MANAGEMENT,10,PAU CLARIS,BARCELONA
  DEPT2,MANAGEMENT,8,RIOS ROSAS,MADRID
  DEPT4,MARKETING,3,RIOS ROSAS,MADRID
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you would need to obtain the departments with all employees assigned to the same project. The result must include department number. Save the results in *output.txt*. In the previous example, the result should be $DEPT2$ and $DEPT4$.

7.4.7 Consider two files containing the following kinds of data:

```
Employees.txt
  EMP4;RICARDO;250000;MADRID;DPT4
  EMP5;EULALIA;150000;BARCELONA;DPT5
  EMP6;MIQUEL;125000;BADALONA;DPT5
  EMP7;MARIA;175000;MADRID;DPT6
  EMP8;ESTEBAN;150000;MADRID;DPT6

Departments.txt
  DPT1;DIRECCIO;10;PAU CLARIS;BARCELONA
  DPT2;DIRECCIO;8;RIOS ROSAS;MADRID
  DPT3;MARKETING;1;PAU CLARIS;BARCELONA
  DPT4;MARKETING;3;RIOS ROSAS;MADRID
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you'd need to retrieve the list of department IDs for those departments with workers from all cities where there are employees. Save the results in *output.txt*.

7.4.8 Consider two files containing the following kinds of data:

```
Employees.txt
  EMP1;RICARDO;250000C;MADRID;SITE2
  EMP2;EULALIA;150000C;BARCELONA;SITE1
  EMP3;MIQUEL;125000C;BADALONA;SITE3
  EMP4;MARIA;175000C;MADRID;SITE2
  EMP5;ESTEBAN;150000C;MADRID;SITE4

Departments.txt
  SITE1;DPT.MANAGEMENT;FLOOR10;ST.PAU CLARIS;BARCELONA
  SITE2;DPT.MANAGEMENT;FLOOR8;ST.RIOS ROSAS;MADRID
  SITE3;DPT.MARKETING;FLOOR1;ST.PAU CLARIS;BARCELONA
  SITE4;DPT.MARKETING;FLOOR1;ST.RIOS ROSAS;MADRID
  SITE5;DPT.MARKETING;FLOOR5;ST.MARTI PUJOL;BADALONA
```

Provide the ordered list of Spark operations (no need to follow the exact syntax, but just the kind of operation and main parameters) you'd need to retrieve the list of department IDs for those departments with sites in all cities where employees live (these employees can be even from other departments). Save the results in *output.txt*. In the previous example, the result should be *DPT.MARKETING*, because it has sites in all the three cities where there are employees (i.e.,

MADRID, BARCELONA and BADALONA). However, *DPT.MANAGEMENT* should not be in the result, because it does not have any site in BADALONA, where EMP3 lives.

7.4.9 Consider three files relating to a bibliographic database: *author.csv* relates authors with papers (you may assume that author names are unique, that authors have one or more papers, and that papers have one or more authors); *title.csv* gives the title of a paper (you may assume a paper has one title, but one title may be shared by many papers); and *citation.csv* indicates which papers cite which other papers (you may assume that each paper cites at least one other paper, that a paper may be cited or more times, and that a paper cannot cite itself).

```
author.csv

AUTHOR      PAPERID
...         ...
C. Gutierrez    GP2014
C. Gutierrez    AGP2013
C. Gutierrez    GZ2011
...       ...
J. Perez      GP2014
J. Perez      AGP2013
J. Perez      P2017
...       ...
R. Angles     AGP2013
R. Angles     AKK2016
...       ...

title.csv

PAPERID     TITLE
...         ...
GP2014      Semantics of SPARQL
AGP2013       Deduction for RDF
GZ2011      Graph databases
...         ...

citation.csv

PAPER       CITES
...       ...
GP2014      AGP2013
AGP2013       GZ2011
P2017       AKK2016
...       ...
```

The count of self-citations for an author A, denoted $self(A)$, is defined as the number of citation pairs (P_1, P_2) where A is the author of both. The count of citations given by an author A, denoted $give(A)$, is the count of citation pairs (P_1, P_2) such that A is an author of P_1. The count of citations received by A, denoted $receive(A)$, is the count of citation pairs (P_1, P_2) where A is an author of P_2. The ratio of self-citations to all citations given and received are then defined, respectively, as $\frac{self(A)}{give(A)}$ and $\frac{self(A)}{receive(A)}$. In case that $receive(A) = 0$, you should omit the author A from the results (note that $give(A)$ cannot be 0, as an author must have at least one paper, and a paper must have at least one citation). We provide an example output for the input data:

AUTHOR	SELFGIVERATIO	SELFRECEIVERATIO
C. Gutierrez	1.000	1.000
J. Perez	0.333	1.000
R. Angles	0.000	0.000
...	...	...

We will use Apache Spark to perform the analysis and compute the output. You should not assume any ordering of the input files. You do not need to order the output file in any particular way.

Given this input and desired output, design a Spark process to complete the required processing. In particular, you should draw the high-level DAG of operations that the Spark process will perform, detailing the sequence of transformations and actions. You should briefly describe what each step does, clearly indicating which steps are transformations and which are actions. You should also indicate which RDDs are cirtual and which will be materialized. You should use caching if appropriate. You should provide details on any functions passed as arguments to the transformations/actions you use.

7.5 Spark internals: Theoretical questions

7.5.1 What indicates to Spark query optimizer the end of a stage and the beginning of the next one?

The end of a Spark stage is indicated when a wide transformation is encountered.

7.6 Spark internals: Problems

7.6.1 Considering the file and result example, briefly indicate the problems you find in the Spark code below (if any), and modify the code to fix them (if needed).

```
Employees.txt

  EMP1;CARME;40000;MATARO;DPT1;PROJ1
  EMP2;EUGENIA;35000;TOLEDO;DPT2;PROJ1
  EMP3;JOSEP;25000;SITGES;DPT3;PROJ2
  EMP4;RICARDO;25000;MADRID;DPT4;PROJ2
  EMP5;EULALIA;15000;BARCELONA;DPT5;PROJ2
  EMP6;MIQUEL;12500;BADALONA;DPT5;PROJ3
  EMP7;MARIA;17500;MADRID;DPT6;PROJ3
  EMP8;ESTEBAN;15000;MADRID;DPT6;PROJ3

Expected result

[ PROJ1,[ EUGENIA, 35000, 37500.0 ] ]
[ PROJ2,[ EULALIA, 15000, 21666.6 ] ]
[ PROJ3,[ MIQUEL, 12500, 15000.0 ] ]
```

Code:

```
rawEmps = sc.textFile(r"Employees.txt")
emps = rawEmps.map(lambda l: tuple(l.split(";"))).cache()
RDD1 = emps.map(lambda t: (t[5], (int(t[2]), 1)))
RDD2 = RDD1.reduceByKey(lambda t1, t2: (t1[0] + t2[0])/(t1[1] + t2[1]))
RDD3 = emps.map(lambda t: (t[5], t))
RDD4 = RDD3.join(RDD2)
RDD5 = RDD4.filter(lambda t: int(t[1][0][2])<t[1][1])
RDD6 = RDD5.map(lambda t: (t[0],(t[1][0][1],t[1][0][2],t[1][1])))
Result = RDD6.sortByKey()
```

The given Spark code seems to calculate the average salary per project and filters employees earning less than this average. It also looks like it's supposed to return the project, the name of the employee who has the lowest salary on the project, their salary, and the average project salary. However, it doesn't perform the operations correctly.

Here are the problems and their corrections:

- The calculation of average salary in RDD2 is incorrect. The average should be computed by summing up the salaries and then dividing by the count. The current code performs division in every step of the reduceByKey operation, which doesn't yield the correct average.
- The filtering operation in RDD5 is not required as per the expected output. It filters employees whose salaries are less than the average salary, but the expected output doesn't mention this.
- The result should include the employee with the lowest salary, not just any employee who earns less than the average. This is not correctly implemented in the code.

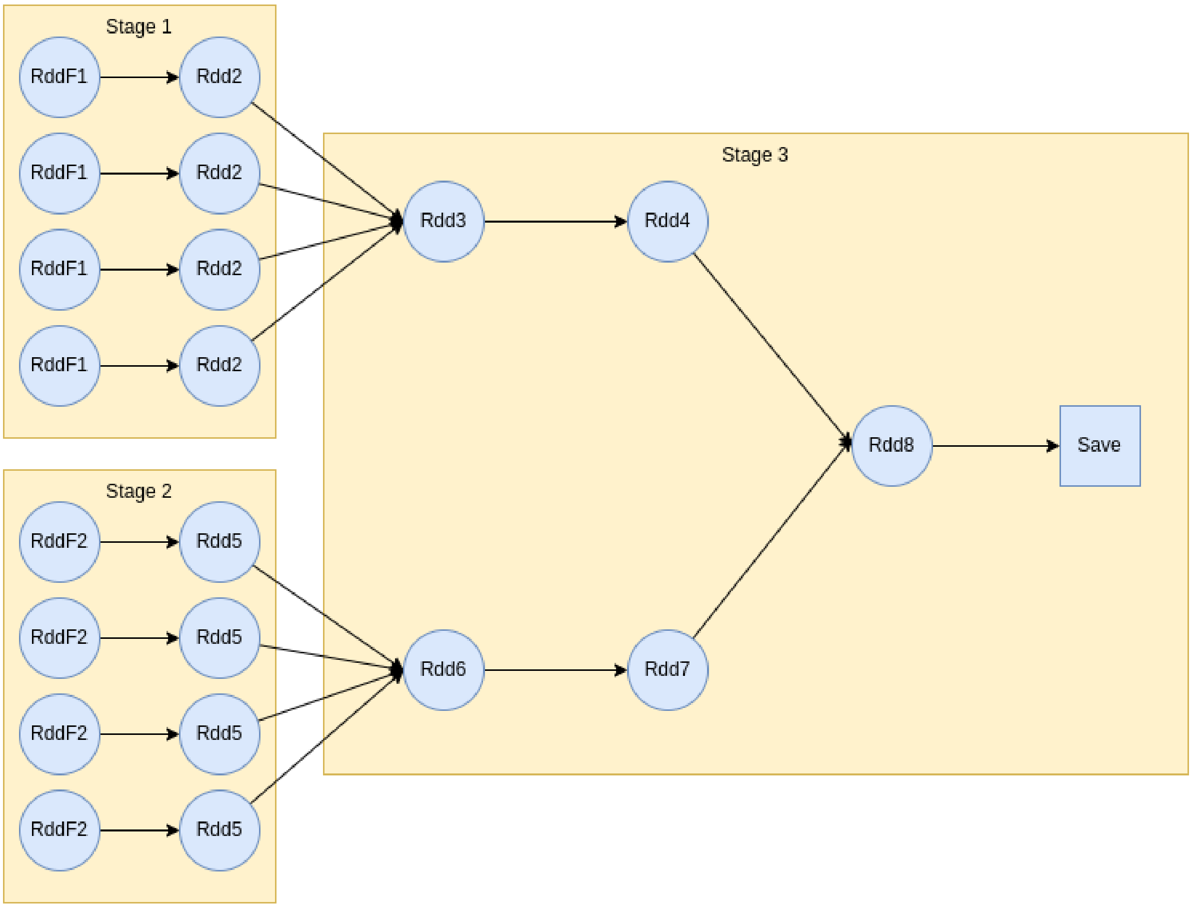
The corrected code could be as follows:

```
rawEmps = sc.textFile(r"Employees.txt")
emps = rawEmps.map(lambda l: tuple(l.split(";"))).cache()

# Mapping the data to ((project), (employee_name, salary, 1)) format
RDD1 = emps.map(lambda t: (t[5], (t[1], int(t[2]), 1)))

# Calculating total salary and count for each project
RDD2 = RDD1.reduceByKey(lambda t1, t2: (t1[0] if t1[1] < t2[1] else t2[0], t1[1] + t2[1], t1[2] + t2[2]))

# Computing average salary for each project and keeping employee with lowest salary
RDD3 = RDD2.map(lambda t: (t[0], (t[1][0], t[1][1]/t[1][2])))

# Including the minimum salary employee with project and average salary
Result = RDD3.sortByKey()
```

7.6.2 Consider the following pipeline. This pipeline runs in a 4-machine cluster with HDFS to store the files and Spark to execute it. *File*1 and *File*2 are distributed in the cluster in 6 chunks each.

```
RddF1 := sc.textFile(''...file1.txt'')
RddF2 := sc.textFile(''...file2.txt'')
Rdd2 := RddF1.mapToPair(s.split('';'')[0],s.split('';'')[1-2])
Rdd3 := Rdd2.GroupByKey()
Rdd4 := Rdd3.MapValues(f1)
Rdd5 := RddF2.mapToPair(s.split('';'')[0],s.split('';'')[1-2])
Rdd6 := Rdd5.GroupByKey()
Rdd7 := Rdd6.MapValues(f2)
Rdd8 := Rdd4.join(Rdd7)
Rdd8.save(''...file3.txt'')
```

1. **How many stages will the scheduler generate for this pipeline? (Justify your answer)** *



2. **How many tasks will be generated within each stage? (Justify your answer)** *

   Stage 1 has 8 tasks. Stage 2 has 8 tasks. Stage 3 has 6 tasks.

7.6.3 **Consider the legacy code written in MapReduce that specifies a $map()$, $combine()$ and $reduce()$ operations. This job reads from a text file f_1. The combine and reduce operations coincide and you can assume all functions are correct. Write a Spark pipeline equivalent to the MapReduce job. Use $fmap$ and $freduce$ to refer to the code executed inside the map and combine/reduce operations. You can parameterize the $fmap$ and $freduce$ functions but resulting in minimal code adaptation.**

7.6.4 **Assume we have a MongoDB collection in a distributed cluster, which contains prices of apartments without any secondary index. Such collection is big enough not to completely fit in memory. We want to use Spark to compute the standard deviation per neighbourhood. Clearly identify the most efficient option and briefly justify the choice (it is not necessary to provide the Spark code).** *

1. **Use Spark only to push the query to MongoDB aggregation framework and simply get the result.**

   No, because we would need the mongo router to perform all the work, losing the power of parallelism.

2. **Push only some of the operations to MongoDB aggregation framework and then run the rest in Spark.**

   Yes, but we need specific drivers for Spark to be able to push operations to the storage engine. For MongoDB these exist.

3. **Load the whole collection to an RDD and perform all computations in Spark.**

   No, because we would need to transfer everything to Spark, implying a huge data transfer overload.

8 Streams

8.1 Problems

8.1.1 Let's suppose an arrival rate $\lambda = 50\ msg/sec$ and a service rate $\mu = 50\ msg/sec$. Given the message arrivals in the table below and a size of $100B$ per message, which is the minimum buffer size needed to guarantee that no message is lost? *

Time	1	2	3	4	5	6	7	8	9	10
#Messages	2	10	100	50	20	75	150	100	40	10

Time	Received	Processed	Buffer
1	2	2	0
2	10	10	0
3	100	50	50
4	50	50	50
5	20	50	20
6	75	30+20	45
7	150	5+45	145
8	100	50	95+100
9	40	50	45+100+40
10	10	45+5	95+40+10
11	0	50	45+40+10
12	0	45+5	35+10
13	0	35+10	0

The maximum amount of messages encountered in the buffer is 195, therefore, its size should be

$$195msg \times 100B/msg = 19500B.$$

8.1.2 Let's suppose a stationary situation in the processing of a stream, where the whole memory available is occupied by messages. Assume also the existence of a separated buffer big enough to allow us the complete processing of these messages already in memory without being concerned with new arrivals. Assume we have $M = 10$ memory pages with $R_S = 10$ messages per page. If the processing of these messages consists only of a lookup and in-memory processing/comparison of a message and a tuple is thousand times smaller than a disk access, give the cost in these two situations considering that every disk block of the table contains also $R_T = 10$ tuples: *

1. **Going through an index for each message with cost $h = 3$ for the index and one more disk access to the lookup table per message.**

   For each of the 10x10=100 messages, we have to access the index, with cost 3, and fetch the data page, so:

   $$100msg \cdot (3 + 1\ IO/msg) = 400IO$$

2. **Bringing each block of the lookup table (whose size is $B = 100$) into memory one by one and checking all messages in memory against all its tuples.**

   In this case, for each block, we need to retrieve it, and compare it (comparison is a thousand times cheaper than a IO), so:

   $$100Blocks \cdot \left(1\ IO/Block + \frac{100comp}{1000} IO/comp\right) = 110\ IO.$$

8.1.3 Let's suppose that setting the execution environment for the processing of messages in a stream is $S = 100$ (e.g., placing & retrieving all the information to/from the stack), independently of their number. Both packing and unpacking the messages in a batch (i.e., a list of elements) have the same cost of $Pk = 1$ per message, and processing each message is $Ps = 10$. *

We can use the formula

$$C = \#batches \times S + \#msg \times P_s + \#packed\_msg \times 2p_k.$$

1. **Which is the cost of truly streaming 10 messages one at a time?**

$$C = 10 \times 100 + 10 \times 10 + 0 \times 2 = 1100.$$

2. **Which is the cost of processing one packed micro-batch of 10 messages at once?**

$$C = 1 \times 100 + 10 \times 10 + 10 \times 2 = 220.$$

8.1.4 Let's suppose we have a log file recording the events coming from different machines. Thus, for each event we have the following information:

$$(logID, traceID, eventID, duration)$$

The *logID* corresponds to the IP of the machine; the *traceID* identifies the transaction inside the machine (i.e., two traceIDs can coincide in different machines); the *eventID* identifies the kind of action performed by the machine; finally, the *duration* is the number of milliseconds taken to implement the action. Assuming that we cannot keep all log entries in memory, and we decide to randomly sample them, give the attributes (up to three) you would use as parameters of the hash function implementing such sampling, so that each of the following queries gives a result as accurate as possible. *

1. **Estimate the fraction of transactions where the same kind of action appears more than once.** $(logID, traceID)$
2. **For each machine, estimate the average number of actions per transaction.** $(logID, traceID)$
3. **Estimate the number of transactions with more than two actions taking more than 100ms.** $(logID, traceID)$
4. **Estimate the average sum of the duration per transaction.** $(logID, traceID)$

8.1.5 Let's suppose we have a black-list of IP addresses (whose packages we do not want to cross our firewall), which it too long to be kept in memory (10^7 elements). Thus, we decide to implement a Bloom filter (to avoid further processing of black-listed addresses), with only 10^8 bits. *

1. **How many hash functions would you use?**

$$k = \frac{n_{bits}}{n_{elem}} \ln 2 = \frac{10^8}{10^7} 0.693 = 6.93,$$

so we would use 7 hash functions[6].

2. **What's the probability of a false positive in that case?**

$$P = \left(1 - e^{-\frac{km}{n}}\right)^k = \left(1 - e^{-\frac{7 \cdot 10^7}{10^8}}\right)^7 = 0.83\%.$$

3. **Briefly explain what is the consequence of a false positive in this concrete case.**

In this case, we would classify a package that comes from a non-black-listed IP as black-listed.

[6] Notice that if the value was more in-between 6 and 7, we would need to substitute 6 and 7 in the false positives function to see which value gives a lower estimate.

8.1.6 Let's suppose we have a log file recording the events coming from different machines. Thus, for each event we have the following information:

$$(logID, traceID, eventID, duration)\,.$$

The $logID$ **corresponds to the IP of the machine; the** $traceID$ **identifies the transaction inside the machine; the** $eventID$ **identifies the kind of action performed by the machine; finally, the** $duration$ **is the number of milliseconds taken to implement the action. Consider the following table and assume that each machine generates the same number of events and at the same pace, and use an exponentially decaying window model with a constant** $c = 0.5$**.**

Time	logID	traceID	eventID	duration
1	1	1	A	1
2	2	1	B	100
3	1	1	C	10
4	2	1	D	10
5	1	1	A	100
6	2	1	C	1

1. **Give the milliseconds (and details of calculation) used per machine (i.e. for Machine 1 and Machine 2) when the last event arrives, provided that the current milliseconds are more valuable than the oldest ones.** *

		Time					
		1	2	3	4	5	6
Machine	1	1	0.5	10.25	5.125	102.5625	51.28125
	2	-	100	50	35	17.5	9.75

8.1.7 Assume we ingest a stream with an event every time a ticket is sold at a theater. Precisely, the stream has the following structure:

$$(movieID, theaterID, timestamps, price)\,.$$

Next, we ingest the following ordered set of events:

- $(m1, t1, 12h, 10€)$
- $(m2, t1, 14h, 12€)$
- $(m2, t2, 15h, 18€)$
- $(m1, t3, 18h, 6€)$
- $(m3, t2, 19:15h, 13€)$
- $(m1, t3, 19:30h, 10€)$
- $(m2, t3, 19:45h, 25€)$
- $(m3, t1, 20h, 17€)$
- $(m1, t3, 20:30h, 10€)$
- $(m2, t2, 21h, 8€)$

Provide a detailed answer (i.e., describe the process) for the following questions:

1. **Which theaters would be considered heavy hitters by the algorithm, given a required frequency of 50%?** *

   In reality there are no heavy hitters with 50% threshold. Let's see with the algorithm:

1)

Struct	
elem	counter
t1	1

2)

Struct	
elem	counter
t1	2

3)

Struct	
elem	counter
t1	2
t2	1

4)

Struct	
elem	counter
t1	1

5)

Struct	
elem	counter
t1	1
t2	1

6)

Struct	
elem	counter

7)

Struct	
elem	counter
t3	1

8)

Struct	
elem	counter
t3	1
t1	1

9)

Struct	
elem	counter
t3	2
t1	1

10)

Struct	
elem	counter
t3	1

   So, t3 would be considered a heavy hitter.

2. **Under the exponentially-decaying window model, using an aging constant of $c = 0.1$ and a purging threshold of 0.6 (i.e., we'll remove movies as soon as their popularity falls strictly below 0.6). What would be the most popular movies at 21h, considering an aging tick every 15 minutes? Give the value corresponding for each of the movies in that moment.**

   Note that with $c = 0.1$, we can obtain the number of ticks that a film stays above the threshold of 0.6 after appearing once by solving

$$(1-c)^t < 0.6 \iff 0.9^t < 0.6 \iff t \log 0.9 < \log 0.6 \iff t > \frac{\log 0.6}{\log 0.9} = 4.84,$$

   therefore, after 5 ticks, i.e., 1:30h, an item would be purged if it only appeared once.

	12:00	14:00	15:00	18:00	19:15	19:30	19:45	20:00	20:30
m1	1	-	-	1	-	1	0.9	0.81	1.6561
m2	-	1	$1 \cdot 0.9^4 + 1 = 1.6561$	$1.6561 \cdot 0.9^{12} = 0.468 \implies$ -	-	-	1	0.9	0.729
m3	-	-	-	-	1	0.9	0.81	1.729	1.40

   Therefore, the most popular movie es m2.

8.1.8 If you have a stream of messages that arrive at a rate of 10 000 per second accounting for 512 MB per hour, how many RDDs will process your Spark Streaming process in one hour? Explicit all the assumptions you make. *

If the RDDs are triggered every t seconds, then there will be $\frac{3600}{t}$ RDDs/h.

8.1.9 Assume we ingest a stream with an event every time a ticket is sold at a theater. Precisely, the stream has the structure $(movieID, theaterID, timestamp, price)$. Nest, we ingest the following ordered set of events:

- $(m3, t4, 12h, 10€)$
- $(m1, t2, 13h, 17€)$
- $(m2, t4, 14h, 11€)$
- $(m4, t1, 15h, 8€)$
- $(m1, t3, 16h, 9€)$

- $(m3, t4, 17h, 5€)$
- $(m6, t1, 18h, 15€)$
- $(m5, t2, 19h, 12€)$
- $(m7, t5, 20h, 17€)$
- $(m1, t1, 21h, 11€)$

Which theaters would be considered heavy hitters (using the approximate method) considering a required frequency of 33%? Provide a detailed answer(i.e., describe the content of the auxiliary structure after processing every message).

9 Architectures

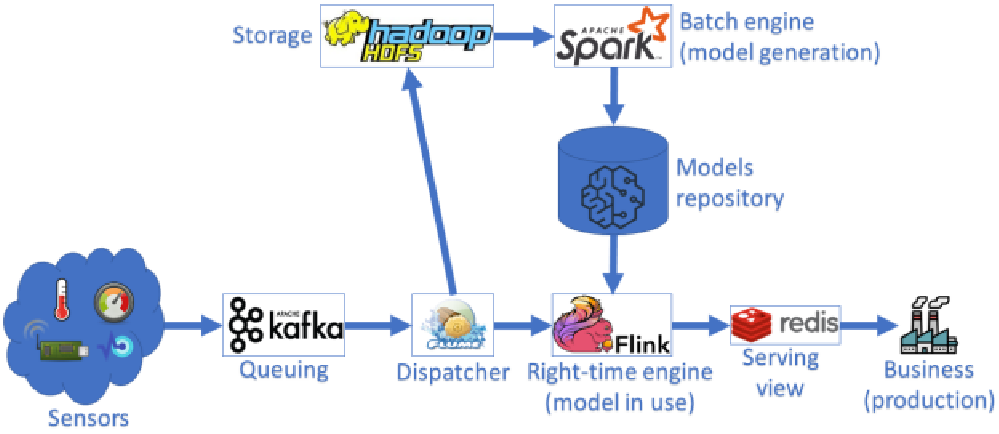
9.1 Theoretical questions

9.1.1 What problems do Data Lakes solve? *

The problem of having a single input/truth, which arises in traditional architectures in which the data model is fixed beforehand for the whole organization.

9.2 Problems

9.2.1 Consider the software architecture below and briefly explain the main management (maintenance) risk it has. *



This is a λ-architecture, so we would need to take care that the transformations done in the batch processing, are also applied in the streaming side. Also, versioning issues can arise between both processing batches.