HW01 Distributed Systems - Part C

1 Question 1.A: Fragmentation Decisions

Let's denote $Q = \{q_1, q_2, q_3\}$ as our query set from Part C.

Motivation

The queries being asked in our distributed system require access to certain attributes in the given Schema. Therefore, it would be beneficial to divide up the schema into fragments given the Query set Q. We solve this problem by using Horizontal and Vertical fragmentation methods. Although traditional horizontal fragmentation, where rows are divided based on certain criteria, may not be sufficient when specific queries frequently access a subset of attributes in the given schema.

Vertical fragmentation addresses this further by the following:

Reducing Storage Requirements:
By storing only the necessary attributes for a specific query in a fragment, vertical fragmentation minimizes the storage space required for the relation in relation to the server.

Improving Query Performance: Retrieving data from smaller fragments containing only relevant attributes is faster than accessing the entire relation, leading to faster query execution times.

Maximizing Processor Usage further: When querying a specific fragment, only certain required data is needing, thus further fragmentation can make more efficient usage of all the processors in the different servers.

Enhancing Data Security: Storing fragments with potentially sensitive attributes on different servers can improve data security since the queries only access fragments with data relevant to that query.

Therefore, using Horizontal and Vertical fragmentation in the correct manner can help optimize the points mentioned above and thus help with performance in regards to the queries and other measures in a distributed system.

1.1 Fragment Design

The design of fragments using vertical fragmentation involves grouping attributes based on their relevance to specific queries. Here are some key considerations:

Attribute Affinity Matrix: This matrix represents the relationship between attributes, indicating which attributes frequently appear together in the same queries.

Query Frequency: The frequency of a query's usage plays a crucial role in determining the importance of the attributes it accesses.

Data Preservation and Redundancy: While grouping attributes based on query access, it's important to include the primary key in each fragment to ensure data integrity. In some cases, certain attributes might be needed in multiple fragments, leading to data redundancy that needs to be carefully managed during updates.

One common algorithm used for vertical fragmentation is the BEA (Benefit-Cost Analysis) algorithm. Given an input of the affinity attribute measure matrix we can run the algorithm to find the matrix CA to help us divide the fragments up. This requires calculations and measures between attributes which isn't given in the question and thus without this data given it would be difficult to find the right assumptions to measure the relationships and use the algorithm.

Vertical Fragmentation

Vertical fragmentation is a database design technique that splits a relation (table) into multiple smaller relations, each containing a subset of the original attributes. This approach aims to improve storage utilization, query performance, and network traffic by storing only the attributes relevant to specific queries.

we observe that not all attributes are required in the given queries set Q. To avoid using all the relations and to save memory and computation time, we split each relation into a fragment containing the attributes required for the queries (along with the keys) and a seperate fragment containing the attributes that are not needed for the queries (along with the keys, for data preservation and enabling future access if needed). We send the fragments with the unnecessary attributes to a server in a region that does not perform queries involving these attributes. This will help prevent server overloading in certain regions, where we place the fragments with the necessary attributes and information.

Let's define the attribute affinity matrix with relevant attributes to our query set Q:

Query	NetWorth	DeviceID	Genre	AirTime	Duration	GreenLiving
q_1	1	1	0	0	0	0
q_2	0	1	1	1	0	0
q_3	0	0	1	0	1	1

We will first set most of the attributes which aren't needed for the results of the query as it's own fragment so that we don't lose any data (partially these are attributes not shown in the AA matrix as all their entries would be zeros).

vertical fragmentation process:

T_4 :HHID, DeviceID, DMA, ZipCode, householdSize, NumOfAdults, event_Date, eventTime, Title where HHID, DeviceID are the keys

Now we will divide the queries into three fragments using the relevant attributes needed. Furthermore, we do not want to lose previous relationships from the original schema and thus keeping the keys together as they were originally is essential for correctness.

T_1 : HHID, DeviceID, NetWorth where HHID, DeviceID are keys

T_2 : HHID, DeviceID, Genre, Prog_code, Air_date, Air_time where the keys are all the attributes but Genre

T_3 : HHID, DeviceID, Genre, GreenLiving, Duration where HHID, DeviceID are keys

Choice for keys are due to them being keys in the original relations giving thus it makes sense to keep the same keys where needed in order to keep correctness and the ability to reproduce the original relations if required.

Horizontal Fragmentation:

Based on the predicates such as **Genre=news** and **GreenLiving=true** (and their complementing predicates) in our queries we will divide our fragments further. Therefore, we will split the schema given query 3 into three fragments - one where **GreenLiving** is True and the other when False (usage of **DHF** by its definition) and **Genre="news"**. We also keep a third as to not lose information even if the queries don't use it.

$T_{3.1}$: HHID, DeviceID, Genre="news", GreenLiving=True, Duration

$T_{3.2}$: HHID, DeviceID, Genre="news", GreenLiving=False, Duration

$T_{3.3}$: HHID, DeviceID, Genre≠"news", GreenLiving, Duration

We will also split T_2 into 2 fragments since we are checking for Genre="*news*". Therefore, we get:

T_2 : HHID, DeviceID, Genre="news", Prog_code

T_5 : HHID, DeviceID, Genre $\neq$ "news", Prog_code

Thus resulting in 6 fragments which we can put on five servers given.

Reasoning:
As for not using some algorithms learned in class: Looking at the queries and their structure and seeing that they are relatively simple, we were able to methodically cut vertically and horizontally in such a way that we get relatively compact fragments as requested in the question without over complicating the division/splitting into fragments.

Furthermore, it seems that for some of the algorithms learned in class require some real data on the distributed system/database in order to do calculations to split the schema/data into fragments in a precise and optimal manner. As we do not have real-time data to do calculation initially as to how to split up the schema, it seems most relevant to use some of the algorithms learned in class.

We can also see that HHID, DeviceID are keys that are in the fragments in order to have correctness, in case we want to reconstruct the fragments to their previous relations. Therefore, even though this adds more storage usage it's mandatory for correctness. As well as prog_code, air_time, air_date where needed.

Reason for using (or not using) Horizontal algorithms:

- **PHF-**
  if we want to use this method then we would have to create a group of simple minterm predicates group given R where $Pr = \{Genre = news, Genre \neq news, , GreenLiving = True, GreenLiving = False\}$. We can't add other simple predicates since they wouldn't cover all the cases between the different queries in our Q set. Furthermore, we lack data to do calculations for access frequencies and minterm selectivities which is required to help find a division that is minimal and therefore this method wouldn't help us find a good division of fragments. Therefore, it would be less suitable for us to use this method for the fragmentation division.
- **DHF-**
  We used DHF when splitting the tuples horizontally. We split to certain fragments by $Genre =$"$news$" and another pair by $GreenLiving = True$. Which we can reverse with semijoins thus is a DHF division by definition.

Hybrid Fragmentation

Overall, we performed two-stage fragmentation: vertical, and then horizontal.

Another approach could have been used for fragments 3.1 and 3.2, 3.3: first using horizontal fragmentation to create the fragment for the first predicate ("Genre=News"), place it in Texas, then replicate and add the attribute of "Environmentally Friendly Housing" using vertical fragmentation, and then again using horizontal fragmentation to split into two fragments for the values "True" and "False" of the added attribute. However, replication could have brought many unnecessary attributes to the last two fragments. Given that the frequency of query usage varies and Query 2 is read seven times more often than Query 3, there could be significant and unnecessary overhead on fragments 3.1 and 3.2 in this case. Query 2 is asked most frequently, so the attributes it needs are the most significant. Query 3 is read less frequently (with lower priority attributes), and Query 4 is read the least frequently, with the least significant attributes. On the other hand, we saw in the lecture that in terms of efficiency, it is better to replicate fragments rather than split into many attributes. Therefore, if fragments 2 and 3 have mostly common attributes and one does not have very few attributes while the other is very large, it is better to perform the replication and follow the described process.

1.2 Question 1.B: Fragmentation Division based on Predicates and Locations

We will utilize the predicates that we found based on different locations to decide on the fragmentation placement strategy for our data segments. Here's an overview of the fragmentation division:

1. T_1 **- California Server:**
   In order to ensure quick access to query results from q_1 every **Month** within the relevant location. This minimizes cost and optimizes performance.
2. T_2 **- Texas Server:**
   To enable swift access to query results from q_2 daily (at 8pm) in Texas. This approach minimizes cost and optimizes performance.
3. $T_{3.1}, T_{3.2}$ **- NJ Server:**
   To facilitate rapid retrieval of query results from q_3 where **Genre=news** on a weekly basis. This approach minimizes cost and optimizes performance due to NJ's proximity to Washington and the benefit of keeping required data close by for storage efficiency.
4. $T_{3.3}$ **- NY Server:**
   To allow for quick access to query results from q_3 where **Genre≠"news"** on a weekly basis. This approach minimizes cost and optimizes performance due to NY's proximity to Washington and the benefit of keeping required data close by for storage efficiency. Additionally, it offers differentiation from the slightly different q_3 query served by NJ. In addition, queries can get update and so we can move it to the NY server if the Genre is changed/updated to news in addition to not losing data.
5. T_4, T_5 **- Oregon Server:**
   All remaining data (including keys) are stored on this server (as fragments) to prevent data loss and enable potential utilization in future applications. Since T_4 and T_5 share the same keys, it's more efficient to save them on a single server rather than splitting them up. When presented with query set Q alone, there's no need to access data on the Oregon server as it isn't required. This approach minimizes cost and optimizes performance.
6. **Oklahoma Server:** We were able to accommodate all other fragments on servers closer to locations required by query set Q. Therefore, it's more effective to leave the Oklahoma server empty for future data storage needs and server usages.

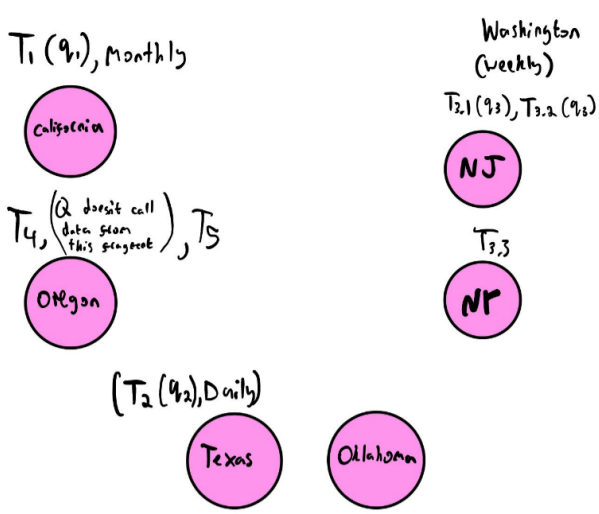

Figure 1: Illustration of fragment placement

2 Question 2

2.1 Question 2.A:

The fragmentation division will change. Firstly, we have to go from 6 servers to 2. Since servers can fit more than one fragment, it makes sense to place the fragment set that gets called frequently on a single server and the other fragment set unto the second server.

Secondly, given $Q = \{q_1, q_2, q_3\}$ queries such that $\forall i, row : r_i \in Q$ and $r_i \notin Q$ we may think that it makes most sense to divide the fragments into these two fragment sets given that we only have two servers and thus two fragments sets that maintain the equivalence relation that we set. Where we have one with fragments where there doesn't exist $i : r_i \in Q$ and will be saved on a server for purposes of correctness that we learned in class, the second server with the data that the queries need overall.
We do take this into consideration as shown in Q2.B but we also need to account for the processor limitation. Therefore, our division of fragments will be affected by these two considerations.

Lastly, we won't do a horizontal fragmentation since we only have two servers and thus dividing up by certain predicates as we did in the previous question may be inefficient and performance may suffer by having multiple fragments with the same attributes where we need to find the right fragment to place the new data.

2.2 Question 2.B:

New Division of fragments (*):

$T_1' := \{T_2, T_4\}$ where $|T_1'| = 2$
$T_2' := \{T_1, T_3\}$ where $|T_2'| = 2$

where T_1, T_2, T_3, T_4 are defined in the vertical fragmentation in question 1.
We could divide T_1' such that it includes all the fragments where for some $i : r_i \in Q$. Also, we also include other data that may not be in a result from the query set Q. For example. for q_2 a result where the $Genre \neq "news"$ since we believe that it's not worth the processing/movement/memory cost for the data to keep it in the second server. Same if not similar reasoning for other data $r_i \notin Q$.
thus it may be intuitive to make the following division:

$T_1' := \{T_1, T_2, T_3\}$ where $|T_1'| = 3$
$T_2' := \{T_4\}$ where $|T_2'| = 1$

Which would be memory efficient.

On the other hand, we want to make the most use out of the processors given to us on each server. Since each server only has one processor it may overload or cause trouble with the performance of the server in answering queries and thus it may be better to split the fragments in such a way that would be more use out of the two servers including the processor usage.
In addition, It would be more beneficial for memory allocation and usage of the processor since the server has only one so it would be more difficult to multitask over multiple fragments as well as in relation to other performance measurements. Thus, we decided that (*) would be the most efficient division of fragments. In addition, we reduce

from the original total of 7 fragments (from q1) to 4 fragments. Reasoning being that the sub-fragments of T_3 will be on the same server and it would be less heavy when updating the fragment with new data rather than 3 different fragments with the same attributes as T_3 as well making processor performance more efficient.
We divided T_2' such that together with the **keys** (for reconstruction purposes) and **attributes** that are not needed for the results of our query set Q.

2.3 Question 2.c:

fragment set T_1'– **Texas** server
fragment set T_2'– **NY** server

Reason:
Texas server (with fragment set T_1')- Since the results from the query (q_2) that is called from Texas is needed daily, it makes the most sense to place the fragments that contains all the results to our query set Q which the other query results can be sent to Washington (weekly) and California (monthly) when asked for since these query results are needed less frequently. Thus we will place our fragment set T_1' in the Texas server.

NY server (with fragment set T_2')- We could put the second fragment T_2' in any server, we prefer NY.

Although we would have to take into account that each server has a single processor and this may cause limitations to multitasking and performance of retrieving for some $i : R_i \in Q$.

If memory allocation isn't an issue, then it may be better to take the fragments relevant to q_1, q_3 which aren't asked for too frequently (weekly/monthly) and place them in the T_2' fragment set for the second server in order to be efficient with processor usage since multitasking may suffer if they are all in the same set and being called by the relevant queries. hence why we use NY as the second server since q_3 is asked weekly.