~~Placeholder, feel free to add your solutions!~~

**The following answer cTan be wrong, unclear or incomplete, feel free to point out and discuss**

1. a. I. Resource domains include cooling, power, and bandwidth. Only one entry in a domain can be allocated with the resource at the same time. Because the goal of this type of system is to store infrequently accessed data and save energy.

II. Assuming the resource domains are in two-dimension, can assign the disk groups such that they lay onto the diagonal of the domain space or its shifting version, to locate each entry in the group to be in unique column and row. So that they can be access at the same time. If the one diagonal offset is used up, allocate another. It however leads to some conflicts.

b. I. Because the data accesses of those thread are not bound, which means that they potentially compete for the locks synchronising the data. And some accesses might be remote (e.g. or other cpu cores).Those factors increase the access latency.

II. The general idea is to reduce thread contention. One of the examples is to reserve a particular portion of the data for specific CPU core/thread. Therefore, they tend to only access the subsets of the data, which are disjoint.

III. The memory access time is not uniform, which means that accesses to data in the memory associated with different cores or even packages can lead to higher latency. Randomly distributing data among them leads to suboptimal performance.

c. I. Re-introduce buffer pool to hold contiguous writes to avoid in-place update and random writes, updating them in batches.

II. The quicker the access to storage media, the less extra components (e.g. buffer) need to be added to hide the access latency. Thus, the effect of scaling out the hardware would be more direct without those overhead. || From the cw 2022, it doesn’t effect horizontal scalability so this is actually a trick question, it may have performance impacts tho

III. Since SSD is non-volatile, the transactions do not need to be explicitly persisted. However, since the higher latency, individual transactions are executed slower. You have to insure ACID properties are met.

d. I. Model mapping is more suitable for semi/unstructured data that have no fixed scheme and objected to change over time. Structure-mapping is suitable for ones with pre-defined data format that is not intended to be manipulated.

II. Document databases are unstructured, which is even more flexible than some XML use cases, and it preserves some desirable properties of RDBMS, e.g. primary key and index. It is served with programming-like API.

III. (Really not sure about this one, please replace the following text if anyone gets better ans)

MongoDB uses embedding or linking. The former may be cumbersome and inflexible to deal with. While the latter suffers from access performance overhead due to pointers.

One challenge of the document-oriented mapping approach is that it can be difficult to enforce data integrity in the database. Because the structure of the documents can vary, it can be difficult to ensure that all documents in the database have a consistent structure and contain the necessary data. This can make it difficult to write queries that access and manipulate the data in the database and can lead to inconsistencies and errors in the data. --- From Teacher G.

1. a. The tail latencies refer to the small portion of latencies that are relatively higher among the sample. They are crucial to be optimised since they imply some "worse cases" access scenario. Fast access on the client side would be treated as normal. But occasionally slow ones might result in losing market share.

b.

1. It handles conflicts on reads instead of writes, where write actions are usually triggered by customers.

2. It uses sloppy Quorum, where executions are permitted by a subset of the preference list or can be external nodes if not available. This increases the availability of the system.

3. Uses hinted handoff for temporary failures, where another node serves on behalf of the failed one.

4. Eventual consistency and CAP theorem means availability is favoured over consistency hence we reach our SLAs

c. I. (A, 1) -> (A, 1) -> (A, 2) || (B, 2) -> (A, 3) || (C, 3) -> (E, 4)

II. Both write requests at t=7 and the one at t=8, since they would all see conflicts on the immediately previous vector timestamp and start handling them (e.g. merging).

d. (open question, the following is a poor attempt)

Introduce queues to store the batch update tasks associated with each server nodes. Once a server receive the batch task, it saves a partition and forwards the remainders to others in the nearby servers but not execute them immediately. Whenever it is considered not overloaded, small minibatch would be taken from the queue and processed. It would still be available for the online processing at any time.

One possible issue with bulk inserting very many keys in a similar limited range is that we would end up overloading a small portion of the node ring in the DHT constructing the Dynamo database, since key hashes form a continuous ring correspondence to the nodes in the DHT. One adaptation we can make is to reserve a few extra nodes in the Dynamo system to handle bulk uploads. These nodes can immediately accept the input bulk of keys without requiring the client to block or stop. This extra, temporary node can then be inserted in one go into the ring topology along only the key range specified by the insert key range. Akin to hinted handoff, we can then start accepting requests for keys in this range while simultaneously asynchronously transferring data across to other nodes in the ring that will be responsible for the keys neighbouring this new node which will simultaneously replicate all operations. Once this transfer is complete, we can inform the neighbouring nodes and detach this temporary node from the ring. This should have minimal impact on the tail latencies as we gradually loaded in the vast number of keys and began accepting updates immediately after the bulk loading almost. w