CS44800 Project 1 (Feedback)
Disk vs In-Memory vs Database
Spring 2023

Overall Feedback

(a) What have you observed to be the fastest Load operation?

[X] Main-memory Load. [ ] MapDB Load

(b) Why?

MapDB uses a disk file to store the loaded data on disk to provide persistence and durability. The disk access is much slower than main-memory access, which contributes negatively to the performance of MapDB.

Even though that there is an overhead due to MapDB internal call, this is cost negligible compared to the overhead of performing I/O operations. Recall that MapDB uses the file system to store the database (data.db). MapDB uses most of the loading process to set the required metadata in the data.db file.

(c) What have you observed to be the fastest Select operation?

[ ] File Select **[X] Main-memory Select.** [ ] MapDB Select

For this question, we consider the time taken by the **select** operation only (and disregard the load times).

(d) Why?

For the most part, it is the same reason as (b). Since a disk file is used to store the actual data in MapDB, it can be slower than the main-memory data structure in many cases, but it can have many benefits as the answer to (e) illustrates. The File Select method operates directly on the input data file. While MapDB Select and File Select both access the disk to return the results, MapDB utilizes internal indexes that are built during the load operation to speed up data retrievals. Besides, the performance of the File Select is affected by the location of the target tuple in the data set (e.g., the desired tuple could be near the end of the file) – since we are doing a sequential search under the assumption that the file is not sorted.

Furthermore, Main-memory Select does not have any I/O overhead due to disk access. Memory access is orders of magnitude faster than disk access, as we saw in Chapter 16. Besides, the hash map data structure is particularly optimized to guarantee constant time on reads in contrast to linear time performance bounded on the File Select.

Finally, as you saw in Project 2 where you implemented the BufferManager, if the data being queried is already in the buffer pool, the DB system can avoid costly disk I/O operations.

(e) What are the benefits of using the embedded database as backend compare to the in-memory storage?

There are many benefits out of the box for database-backed applications, including**:**

1. **More powerful querying capabilities (e.g., SQL)**. Although, MapDB did not provide such capabilities other embedded databases such as SQLite provide this feature.
2. **Flexibility for handling and retrieving data.** This benefit was not observed in this project, but Database in general can provided a lot of flexibility while handling the data. For this project, all the select operation were by using the key of the record. However, the database gives us the flexibility to do requests using other fields or criteria. If we wanted to have a similar support in In-Memory or File System-based backends, it will require a more elaborated implementation.
3. **Persistence and Durability**. Since embedded databases use secondary storage, the application can terminate safely without losing the stored data. MapDB also supports main-memory storage, and we encourage you to explore that and compare performance results with the main-memory solution you examined in this project.
4. **Overcoming the main-memory restrictions**. Main-memory tend to have much smaller capacities than secondary storage (Hard Disks and SSDs). Although current main-memory can go up to 100s of gigabytes of RAM, storing terabytes of data with today's storage technologies still require secondary storage.

Note that the loading of the 30MB file, exhausted the heap space because of the runtime parameter (`-Xmx128m`) used limits the amount of main memory that is available for running the Java application. In contrast, using MapDB to store your data allows utilizing the disk which has a much larger capacity (but it is still finite).