3.THE CONTAINER DATABASE ABSTRACTION

In this chapter we propose an abstraction, the Container Database (CDB), suited for storage of files of any type, and information items to describe them – their metadata. We claim this abstraction to be a lightweight database management system because it provides persistent storage for the files and their metadata, a Programming Interface that allows for client applications to store and to retrieve files based on their filenames, and restricted Transaction Management – a level of compliance with ACID (Atomicity, Consistency, Isolation, Durability) [20] transaction properties, whose limitations are discussed later on this chapter.

3.1.Container Database High-level Architecture

The proposed abstraction is comprised of two main areas for storage: a Local Container, which acts as the main database storage area, and a Cloud Container, which resides on a Cloud and will store files in case of overflow of the Local Container, as we can see in Figure 4.

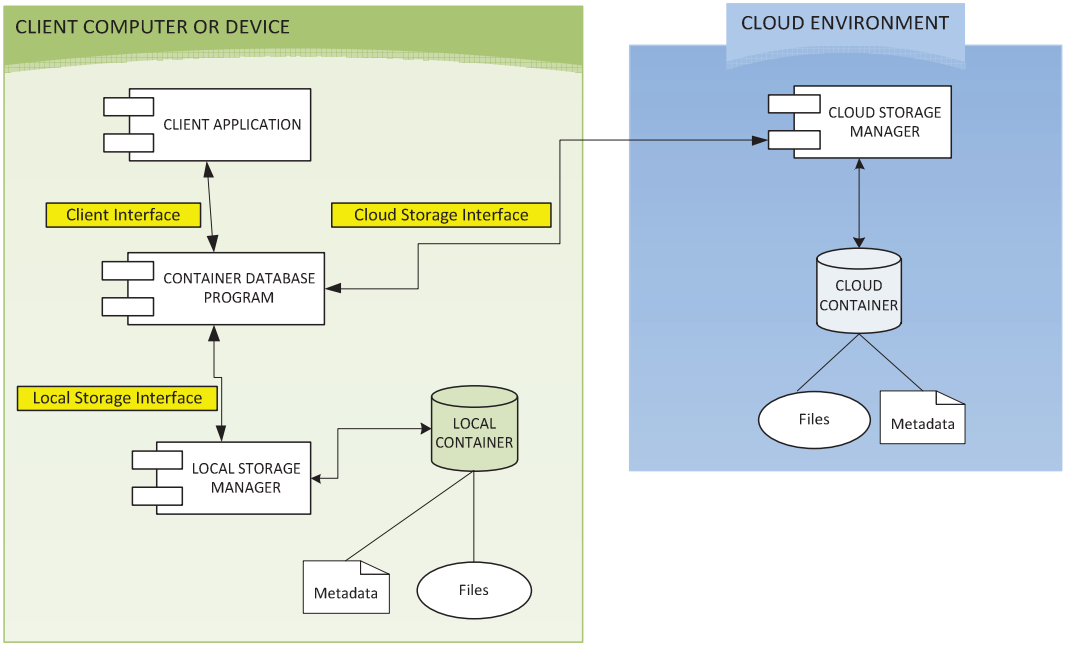

Figure 4: The Container Database High level architecture

The Container Database Program decides where to execute operations requested by Client Applications:

- If an operation is designated to be executed on the local machine or device, it will call the Local Storage Manager**,** via a Local Storage Interface, to manipulate files and their metadata) in the Local Container. This Local Storage Manager is in fact the component that physically manipulates the Local Container, and it could be the File/Directory subsystem of the Operating System running on the Local machine.
- If an operation is designated to be executed on the Cloud, it will call the Cloud Storage Manager, using the Cloud Storage Interface, to manipulate files and their metadata in the Cloud Container. The Cloud Storage Manager which will itself perform the operation on the Cloud Container.

Client Applications work with the Container Database Program via a Client Interface. This interface allows the Client Application to execute CRUD (Create, Read, Update and Delete) operations involving files and their metadata.

3.2.Basic Operations and Mechanisms

3.2.1.System initialization and Storage settings

Defining storage to be used

A Client application initializes the CDB program once through its life time. During initialization, it needs to specify both Local Container and Cloud Container to be used. If necessary, it should also provide the CDB program access permission to the containers.

File Types specification

It is required that the Client Application programmatically define what are the types of files it is dealing with as well as which metadata is associated with them. This information is passed to the CDB program initializer; for it will be used by the CDB in order to correctly store and retrieve file Metadata. For instance, the Client Application could define a type "Analysis Spreadsheet", and each file of this type should be stored with description info such "Project", "Date", and "Supervisor". This step resembles the schema definition, a step in the design of relational database systems [21].

CDB initialization process

The CDB program will check if the designated containers exist and whether it has permissions to manipulate them. Once it passes this verification step, it will perform a couple of checks:

- For each file "type" defined by the Client Application, it verifies if these types already have data and associated Metadata stored in the designated containers or if they are new types to be manipulated. The CDB program uses this info to initialize its internal in-memory structures to allow for client application requests.
- It will try to find a transaction log that reports unfinished operations, detailed in section 3.3.1 below. If there is one, it will read it and undo the unfinished transactions so the database goes back to a consistent state.

Once these three steps are complete, the client application can start to use the CDB program to manipulate files by calling its CRUD operations.

3.2.2.Adding files to the CDB

To add a file to the CDB, the client user must provide a filename, its location and its metadata, if any. The CDB program will store it in a Container and store its metadata too. The file/metadata maybe stored either locally or in the Cloud, according to the CDB overflow mechanism or migration policies (see below). When a file is stored locally, its metadata also is stored locally. When a file is stored in the Cloud, its metadata is also stored in the Cloud. One file is never stored in two places – either it is stored locally or it is stored in the Cloud.

The CDB program keeps an in-memory record of where it stores each file/metadata, in order to be able to retrieve, update or delete them.

Overflow Mechanism and Migration Policies

Client Applications use the Client Interface to manipulate files in the CDB, referred here as to blobs (from Binary Large Objects). When a Client application tries to insert a file in the CDB, these steps are followed: first an attempt is made to insert it in the local container. If there is space available locally, the file and metadata are stored locally. Otherwise, the CDB program checks the file size. If it it's smaller than 64MB, it's uploaded to the Cloud Container in one single request. If it's bigger than 64MB, the file is partitioned and sent to the Cloud in a series of 4MB blocks. The upload process is controlled internally and it is transparent to the client application.

Moreover, users can decide upfront which kind of files they want to be sent straight to the Cloud. In this case no attempt is made to store these files locally. To accomplish this behavior, the Client Interface allows the Client Application to define a list of ***Migration Policies*** for each file type. Each ***Policy*** is a user-defined predicate, i.e., a function that has no parameters and returns either true or false. For instance, an application may define a Policy stating that files of type "X", smaller than 48MB, should be always stored in the Cloud. At runtime, a file Insertion operation will check the type and size of the file being added. If the type and size matches, it will redirect the insertion operation to the Cloud Storage Manager.

It is important to notice that a Policy should be any valid code fragment and it can use any resource at its runtime scope. This approach gives great flexibility

for users that want to use a more directed storage strategy, instead of just using the CDB to overcome storage shortage problems.

3.2.3.Searching for a File

As mentioned on the previous section, the CDB program keeps a live record of each file/metadata it stores. The Client Application can search for a specific file by providing its filename. As a consequence, file names must be unique.

On a successful file search operation, the CDB program will return the file location in an URL form as well as its metadata, packed in an in-memory structure. The Client Application can use this structure to get the needed info only, or she can use it also to:

- Retrieve the file to a local destination;
- Update the file metadata;
- Delete the file.

3.2.4.Retrieving Files locally

Once a Client Application has a file location URL, it is able to retrieve the physical file associated to it by providing a local destination path. When it asks the CDB to retrieve a file, the latter determines if it was stored on the Local Container or on the Cloud Container.

- If it was stored locally, the CDB program uses the Local Storage Manager to copy the file to the destination path;
- If it was stored in the Cloud, the CDB program uses the Cloud Storage Manager to download the file and the Local Storage Manager to copy the file to the destination path.

3.2.5.Updating File Metadata

A Client Application can update the metadata of a file, regardless of where it was stored.

- If it was stored locally, the CDB program uses the Local Storage Manager to update its metadata;

- If it was stored in the Cloud, the CDB program uses the Cloud Storage Manager to update its metadata.

3.2.6. Deleting a File

When a Client Application requests the CDB to delete a file, it will check whether the file is stored locally or in the Cloud:

- If the file is in the Cloud, the file and its metadata are physically removed.
- If the file is stored locally, the file is physically removed but the metadata is NOT altered.
- The CDB updates its in-memory record of files to exclude this file entry in both cases (Local or Cloud Container).

Deletion and Metadata Purge

To keep the model simple, on a file deletion operation for locally stored files the CDB program merely accesses the in-memory file record it maintains and deletes the entry for the file that was physically. This operation will render the metadata for that file inaccessible to the system, and it's enough to keep the system consistent. Successive record insertions with or without deletions in local storage will cause local storage allocated to metadata to increase continually, since deletion operations do not remove metadata.

To physically delete metadata from the Local Container, a **Purge Metadata** operation is needed. This operation is decoupled from the Delete operation, and it is the Client Application that decides when to run it. It will scan the unused metadata only – all info related to deleted files – and purge them from the Metadata storage.

3.2.7. Reclaiming Files Stored in the Cloud

After some usage time, the CDB may come to a state where some data is stored locally and some data is stored in the Cloud. If there is space, the Client Application can request the CDB program to reclaim files from the Cloud back to

the Local Container. It is up to the Client Application to decide when or whether to reclaim files or not. The CDB should provide two services:

- **Single reclaim** – To Bring back to local storage one file currently in the Cloud, if possible, reporting whether the operation was successful or not.
- **Multiple reclaim** – To bring back all files stored in the Cloud, in ascendant size order (for this CDB version, this priority model is fixed and cannot be customized). The operation will stop at the first unsuccessful attempt and generate a log containing the files that were successfully claimed back.

3.2.8.Closing the CDB

As mentioned before, the CDB program keeps a record of all files contained in both Local and Cloud Containers. Before the Client Application finishes, it must explicitly call the Client Interface to close the CDB. The CDB program will then perform two actions:

- Serialize its in-memory structure that tracks file locations to the Local Container;
- Check if there are unfinished transactions in the Rollback log. If there aren't any, the log is unnecessary and it is deleted.

3.3.CDB Operations and the ACID Transaction Properties

We define Transactions as operations that must be executed atomically and in isolation from one another. More specifically, it is desirable that they meet the ACID properties [22]:

- Atomicity – Operations that change a database state should either complete all the changes they are supposed to or not change anything at all, avoiding leaving the database in a state of error.
- Consistency – Expectations about relationships among data elements should be met all the time.

- Isolation – Operations should appear to be executing as if no other operation is executing at the same time.
- Durability – Once the operation on a database is completed, its effect should never be lost.

We shall delineate how the CDB abstraction works in relation to these properties.

3.3.1.CDB Atomicity

We claim that the Client Interface CRUD operations ARE atomic.

As an example of what atomicity should stand for, let us consider the whole operation to add a file and its metadata to the CDB. As a matter of fact, this operation comprises three steps:

- Adding the file to a Container;
- Add the Metadata file to a Container;
- CDB updates its record track of which files are being stored and where they are.

If the system halts after the first step is complete but before completion of the third one, the whole system enters a state of error, because there will be a file (and possibly its metadata) occupying space in a Container that will be impossible to retrieve.

A similar situation occurs if we consider file deletion. If the file is physically removed but the CDB program does not remove its entry on its record track, a further attempt to retrieve the file will incur in system error.

For the CDB to exhibit atomicity, the strategy adopted is to create a special file log, called the Rollback Log. At the beginning of each CRUD operation defined in the Client Interface, an entry for this operation is written to the Rollback Log. If the Operation completes successfully, it is removed. If the application halts in the middle of the operation, the next time the CDB program is initialized it will read the Rollback Log and take the necessary steps to bring the system back to a consistent state.

3.3.2.CDB Consistency

The idea of consistency is strongly related to full-fledged relational databases, where one can establish relations between types of info. From this viewpoint, this concept doesn't make much sense in this context, for the CDB system does not contain any means of defining rules between types. It is up to the Client Application to create these relations, should needed be. Moreover, the CDB doesn't allow any type of metadata constraint. This is also up to the Client Application to define, if needed.

3.3.3.CDB Durability

Data and metadata are stored locally or in the Cloud; therefore, the CDB abstraction obviously exhibits this property. Nevertheless it is important to remark that there is no replication strategy [23], common in many database management systems. A Local Container and a Cloud Container, once defined, will be the only storage used by the CDB program.

3.3.4.CDB Isolation

We claim that the CDB abstraction DOES NOT provide for Isolation and we shall give a brief explanation on this matter.

Isolation boils down to assuring that concurrently executing transactions preserve correctness of the database state, i.e., one transaction will not interfere on the results of another transaction happening at the same time. There are several known techniques to reinforce it, like locking, timestamping or validation [24]. However, the CDB abstraction doesn't provide any structure or mechanism to implement any of these techniques, and concurrent call of Public Interface CRUD operations can potentially leave the database in an inconsistent state.

For instance, there's nothing preventing the Client Application to create two instances of a CDB program and initialize them with the same containers. Let's consider the case where one CDB program, say program A, tries **to insert** a file and, roughly at the same time, another program, say program B, tries **to delete** the same file. One possible outcome of these two calls is the following:

A checks if file exists and finds it doesn't exist yet;
A copies the file to the container;
B checks if file exists;
B deletes the file;
A writes metadata to the Container;
B deletes metadata;
B tries to delete file from index and fails;
A adds an entry to its record track for the file and finishes with no error.

This is not the only possible outcome of these two operations together, because it's hard (or sometimes impossible) to predict the exact duration of each program instruction. We remark that it is a veritable case where, at the end of both operation calls, both A and B will be in an error state. Program A will contain a record saying that a file exists where it has already been deleted. A subsequent search operation for this file will result in error. Program B won't be able to complete the deletion, and the operation will result in error, too.

Even though we do consider isolation to be a highly desirable aspect of database systems, incorporation such a feature in this project would itself be a project of the same scope of this project, so it was left as a requirement for future work. Moreover, this requirement is less important if we consider mobile phone applications where, in most cases, there's only one application running at a time, and it doesn't make sense to initialize more than one CDB program per application.