7 Conclusion

We've proposed the conceptual architecture followed by design of a lightweight database manager that uses the Cloud environment to deal with unpredictable demand or unexpected shortage of computer storage, by exploring the concept of data *Elasticity*. We've also provided implementations (the code base) that exhibit examples of technical solutions useful on the materialization of such database architecture.

To conclude this work, we'll present a brief discussion about lessons learned during this project realization and limitations of this current implementation. We then finish proposing a few topics as future work.

7.1.Lessons Learned

7.1.1.Desktop Implementation

This thesis deals with the issue of data bursting; however, the architecture proposed contains basic CRUD operations, additional features of *Controlled File Migration* to the Cloud, as well as *File Reclaim* from the Cloud, which could be used to build a solution for local cache for Cloud data without much additional effort.

In this scenario, data is wholly stored in the Cloud, but with small local caches to speed up queries for most used files. It would require very few modifications in the CDB design to allow for such use, most notably allowing a file to physically coexist both locally and in the Cloud. This possibility will be left for future work.

7.1.2.Windows Phone Implementation

Programming for Mobile Devices required a whole new set of skills and knowledge, which included the Silverlight for Windows Phone framework [29]

and the Asynchronous Programming Model [28], which considerably slowed down development. Moreover, by the time CDB for Windows Phone was being written, Microsoft tools for Windows Phone with Windows Azure were not as comprehensive and stable as those for Desktop computers. These facts led to a path where additional time was spent in learning Windows Phone Programming skills. The combination of all resulted in further delays in the work schedule, culminating in the removal of a previously scheduled feature, the implementation of CRUD atomicity.

7.2.Limitations

The abstraction proposed in this work is highly dependent on Internet bandwidth, as its main goal is to achieve a balance between its two key components: a local storage area and a Cloud storage area. Overflow files go to the Cloud storage area via HTTP transmissions, causing the cost of storing a file in the Cloud to be orders of magnitude higher than storing them locally.

For a brief analysis on how Internet bandwidth impacts the overall flow of work of a CDB instance, we have created an experiment where combined operations of insertion/delete were applied to files of different sizes – 64KB, 1MB, 4MB and 16MB. This experiment was conducted on a desktop PC on a 10Mbps network. Table 8 shows figures for local-only operations, i.e., no Cloud operation involved. Table 9 shows figures for Cloud operations; the first two columns shows Cloud-only operations and the last two columns are intended to estimate a possibly common usage scenario, where most operations are done locally, but 20% are overflown to the Cloud storage area.

	1 File - Local	10 Files - Local	100 Files - Local
64KB	94 ms	687 ms	7.8 s
1MB	453 ms	2.5 s	25 s
4MB	655 ms	1.5 s	19 s
16MB	2.1 s	12 s	1 min 58s

Table 8: Time spent to add+delete files in the local storage

	1 File - Cloud	10 Files - Cloud	10 Files, 20% Cloud	100 Files, 20% Cloud
64KB	2 s	9 s	2 s	24 s
1MB	4 s	38 s	7 s	2 s
4MB	19 s	2 min 15 s	27 s	4 min 32 s
16MB	1 min 04 s	7 min 04 s	1 min 39 s	18 min 42 s

Table 9: Time spent to add+delete files to the Cloud storage area

This experiment is far from being a rigorous standardized benchmark test, but serves as an indication to some interesting observations, summarized below:

- As of today, we cannot expect operation response times similar to those offered to databases without built-in Internet usage. Therefore, client applications that make use of mixed Local-Cloud storage must be designed in a way that network latency times are expected and can be dealt with. For instance, it took less than one second to store and then to delete a 4MB file in the local storage area, whereas the same operations took 19 seconds to complete in the Cloud storage area.
- If we consider scenarios where the Cloud is used as an occasional overflow protection scheme, then the average completion time for operations decreases significantly and becomes much more bearable. For instance, it took 7:04s to insert+delete 10 files of 16MB to the Cloud storage area. However, if we consider a scenario where, on the average, only 20% of files go to the Cloud, this figure drops to 1:39s.

7.3.Future Work

In order to turn these proof-of-concept APIs into ready-for-consume products, the following features need to be further developed:

- For the Desktop API, Isolation/concurrency.
- For the Windows Mobile API, atomicity to CRUD operations as well as Isolation/concurrency.

- For both APIs, a file versioning system.

Moreover, we can go beyond the idea of Cloud-aware Database to the full concept of **Cloud-covered Computing**.

If we look at the CDB model from the perspective that we are basically giving entity classes the ability to serialize and de-serialize themselves, no matter if there are or if there are not local resources – in this case, storage – we can think of extending the model to use the Cloud to also give these entity classes the ability to execute methods no matter if there are or if there are not local resources – this time, processing power. We should be able to create Cloud-covered entity classes which, from the point-of-view of persistence, behave like the blob-derived classes we have introduced on this work (see chapter 3, in particular the overflow, migration and claim back features), and more: if one of its instance's methods cannot or should not run locally, it is redirected to the Cloud.

Similarly to the CDB model, we would need one coordinator running on the local machine to redirect the requested work to the Cloud. Differently from the CDB model, however, a coordinator on the Cloud side would be needed too, to catch the method request from the local machine coordinator along with its parameters and send back the results to it. See Figure 9 below for an initial sketch of the Cloud-covered class model, where the local machine coordinator is referred to as the CloudCoveredBroker and the Cloud Coordinator is referred to as the CloudPowerHouse. Similarly to Migration Policies of the CDB model, we could think of "Execution Policies" for Cloud-covered entities. For instance, we could allow methods to run in the Cloud in multiple instances.

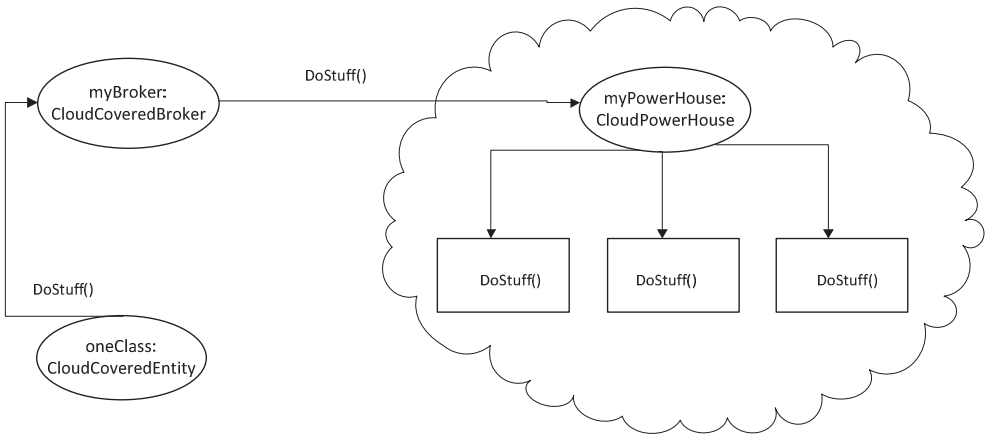

Figure 9: Cloud-covered Computing abstractions