2. CURRENT TECHNOLOGIES

Two APIs were created as proof-of-concept for the proposed architecture in this work: one API for Windows desktop computers and another API for mobile computers, both using Microsoft technology.

This chapter lists some key technologies used to build these APIs, and it can be overall skipped if the reader is familiar with Microsoft latest technologies for Cloud and Mobile Computing.

2.1.Cloud Services

2.1.1.Microsoft Windows Azure Storage Services

There are a few commercial companies, e.g., Amazon, Google, HP, IBM, that offer Cloud Computing services to the public in general [7].

Microsoft currently offers several Cloud Computing Services, and this work uses its Windows Azure Storage Services [10]. In particular, we will use the Windows Azure Blob [11] solution to store our overflown data.

The Windows Azure Blob (Blob: Binary Large Object) is a data type provided by the Windows Azure Storage Services that provides scalable and safe storage for large unstructured data items in the Windows Azure Cloud, like text, media or binary files. To be able to store files in the Azure Cloud, users need to create a **Storage Account**, using the Windows Azure Management Portal [12].

Once a Storage account is created, the user receives a secret access key which is used to create Blob **Containers** in the Azure Cloud space.

Each Container can be assigned permission policies, and each container can store multiple files, which are called **Blobs**. Each Blob may have a size up to 1 TB and can be assigned {key, value} pairs up to 16KB, called the **Metadata**. Communication between any client software and the Storage Service, like requests/responses for creating or deleting containers, downloading or uploading data files, or inspecting the stored files metadata, is done via REST [13] protocol and all protocol commands are documented in the MSDN site [14].

This abstraction model is well-suited for a plethora of applications in the Internet, where storing and retrieving unrelated data and their description are central to its business model.

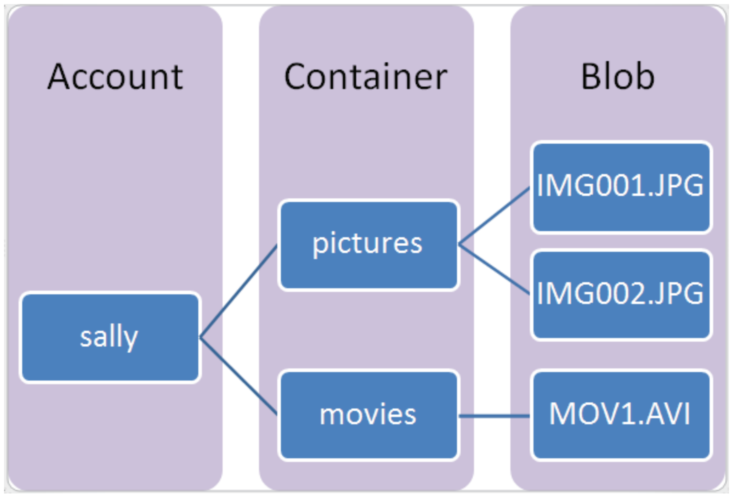

Figure 2: Data example showing MS Azure CloudBlob hierarchy

2.1.2.The Windows® Azure™ SDK and Windows Azure Toolkit for Windows Phone

Microsoft also provides APIs to model the above cited concepts – **Account**, **Container**, **Blob**, **Metadata** – along with other classes, to generate and manage the REST-like requests and responses on a simpler, safer way:

For **Desktop** Applications, the Windows® Azure™ SDK provides both native and managed storage APIs. In this work we are using the managed API, contained in the Microsoft.WindowsAzure.StorageClient namespace [15]. Classes in this namespace provide a client library for working with the Windows Azure storage services.

For **Windows Phone applications**, the Windows Azure Toolkit for Windows Phone provides a managed code API, used in this work, under namespace Microsoft.Samples.WindowsPhoneCloud.StorageClient [16]. This API is not as comprehensive as its desktop counterpart, but still wraps some REST-like requests to the Cloud, like uploading or deleting a file, for instance.

2.2.The Windows NT File System and Memory-mapped files

For the Desktop API implementation specifically, metadata files are manipulated as Memory-mapped files.

32-bit Operation Systems cannot hold in memory files larger than 2GB. To overcome this limitation, so as to allow the proposed Container Database (CDB) to work with large databases, operations that read from or write to metadata files internally opens them as Memory Mapped Files[17]. With Memory Mapped Files one can define portions of a file, named *views*, which are kept in memory and are easily manipulated with view pointers. We can use these view pointers to read from/write to views directly in memory, and the Operation System will use its improved caching techniques to transfer bytes to and from the hardware device, physically updating files stored in the disk. Memory-mapped files can be shared across multiple processes. Processes can map to the same memory-mapped file (see Figure 3).

Following we see a quick pseudocode example on how to write to a memory-mapped file. On this example we use method CreateFromFile() to open file <dataFilePath> as a memory-mapped file. We then use method CreateViewAccessor() to create a view for it that starts at position <writePositionData> with view size of <viewSize>. Then we use method WriteArray() to write the content of <elemBuffer> :

```
using (var mmf = MemoryMappedFile.CreateFromFile(dataFilePath,
                 FileMode.Open, DataFileName, newFileSize))
      {
        using (var accessor =
               mmf.CreateViewAccessor(writePosition, viewSize))
        {
          accessor.WriteArray(0,elemBuffer,0,elemBuffer.Length);
          accessor.Flush();
        }
      }
```

The immediate benefit of using memory-mapped files is the ability of working with very large data files efficiently. Moreover, the ability of defining views that can be concurrently accessed would allow for a fine-grained concurrency mechanism, not implemented on these current Container Database (CDB) versions, but a topic to be explored as a future work.

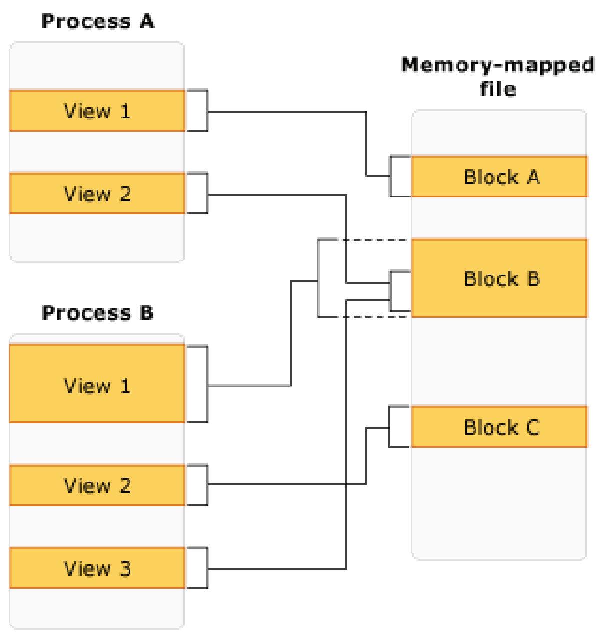

Figure 3: Memory-mapped file and views

2.3. System.Generics.Collections.Dictionary Class

Container Database (CDB) instances keep a Main Index in memory to track files type and location. The Main Index is implemented as an instance of a System.Generics.Collections.Dictionary class, internally implemented using a Hashing scheme and an array store Key/Value pairs, for fast access.

The Dictionary constructor uses a default capacity of 3 elements and keeps the number of empty buckets available for use. If an element is added that exceeds its current capacity, it its current capacity value and creates a new array where the new capacity value will be the next prime closest to the calculated value, copying the contents of the old array to the new one. To provide a fast insert operation, a helper class maintains a static table of the next prime to use, up to 20 doublings, which guarantees a real fast operation for approximately five million entries. The

Dictionary has overall average performance of O(1) for insertion, deletion and retrieval, which guarantees fast access operations.

2.4.LINQ to XML

The CDB system maintains a Rollback log file to fix the database in case Insert, Delete or Update operations are interrupted and the system is left inconsistent. This Rollback log file was implemented using Linq 2 XML Classes [18]. LINQ to XML is an in-memory XML programming interface that enables users to load XML code from files or streams, to serialize XML code to files or streams, and to query XML trees using LINQqueries. It offers similar functionality as the Document Object Model [19].