VFC-BOX: A Multi-user System For Consistent File Sharing

Jean-Pierre Ramos
Advisor: Prof. Dr. Paulo Ferreira

Distributed Systems Group, INESC-ID
Technical University of Lisbon
jean-pierre.ramos@ist.utl.pt

Abstract

Computer assisted collaborative work has been motivating the design and implementation of systems capable of efficiently sharing and storing data. This document provides an overview of the current state-of-the-art technologies to efficiently share and store data. Further, it focuses on the ability of systems to use the interest of collaborators over parts of the shared information in order to reduce the use of network resources and shrink the data's latency seen by users.

Relevant systems are presented and discussed, indicating their main advantages and disadvantages. Finally, this document presents an overview of the preliminary VFC-BOX architecture. VFC-BOX is a multi-user consistent file sharing system, based on deduplication techniques and a consistency model that takes into account the interest management (or locality-awareness) of a user. Through these techniques, this system may achieve higher performances w.r.t. data transfer/synchronization and storage.

Keywords: VFC-BOX, File Sharing, Efficient Storage, Efficient Synchronization, Data Deduplication, Optimistic Replication, Interest Management (locality-awareness).

1 Introduction

The increasing development of the ubiquitous and pervasive computing areas[45] have been creating a requirement for new and more efficient ways of sharing data. Ally to this, there are collaborative tools that foster this sharing of data and its manipulation amongst users. These types of tools are being spread for many environments, in which there are many limitations such as reduced memory and low bandwidth networks. Due to this, efficient information sharing is considered a fundamental aspect to computer supported cooperative work (CSCW)[17].

Consider for instance a team of co-workers that want to share and change

a set of files. To do this, they can use a system that shares data consistently. Furthermore, users can be working with either a desktop that has a high speed connection; a laptop linked to a low bandwidth network, or even with a smartphone with an intermittent connection to the network. Additionally, we may find that many times most of the elements of the team are only interested in some files or even parts of a file. For instance, a team of co-workers writing a document may have elements that are only working in a specific part of the document, not being really interested on the rest of the document. Taking this into account. we may say that it might not be a concern to the elements of a working team, if parts of the sharing data in which they are not interested are not consistent at all times. Nevertheless, if two users are working in the same chapter of a document, they may want to ensure the consistency of that data through time.

The goal of this work is to design and build a system called VFC-BOX that efficiently manages the consistent sharing and storage of data across a multi-user network. The system is required to be efficient, regarding not only the extra overhead required for synchronization, but also the memory usage required for storage and network bandwith. Another requirement is that the system has to be scalable to large networks and to manage large amounts of data. Ideally, the system should also be able to reduce the latency of data which is considered as interest to users and improve concurrency amongst users. To deal with large networks and avoid the problem of having a bottleneck in the network communication, VFC-BOX has to consume the minimum of network bandwith, reducing communication channels where they are not fundamental as well as data to be transfered. To deal with large amounts of data and avoid an overflow of the storage site's disk capacity, this system has to reduce the amount of data stored.

To fulfill the above mentioned requirements, VFC-BOX has to deal with the following challenges: i) reduce the space used to store data through the use of compact forms of representing data; ii) reduce the amount of data to be transfered through compact forms of representing data to be transfered; iii) allow concurrent access to files, while preserving replica consistency; iv) ensure the correct data synchronization, while reducing the use of network resources; v) deal with conflicts, supporting ways of detecting and resolving them through the merge of concurrent data updates; vi) support disconnected work.

Many of the current solutions either to synchronize or store data in distributed systems such as Semantic-chunks[41], BackupChunk [25], Venti[29] and LBFS[24], are based in similarity between data. Data similarity or data redundancy is the concept related with the fact that most of the data that we share and store is similar to other pieces. This redundancy can be detected using **data deduplication**[22] techniques that compare the similarity between portions of data. With this data similarity exploitation, we may achieve higher performance in the storage of data, eliminating redundant data, and substituting it by references to a single instance storage. Efficient synchronization may also be achieved by not sending data that is found as redundant between two sites. Nevertheless, these solutions have the problem of not reducing/postponing com-

munications between sites, due to the inability to adapt consistency guarantees according to the needs.

Other solutions are based in **Optimistic Replication**[33], which enables the bounded divergence of data consistency. These type of solutions try making a trade-off between weaker consistency guarantees and the possibility of a higher performance on availability and access to the replica objects. Nevertheless, most of these solutions do not contemplate an efficient management of updates. Most solutions decide to take an approach to either enforce strong consistency guarantees to all updates, or non at all. Additionally, these solutions also do not cope with the requirements of low memory usage, not being able to sometimes avoid overflowing the disk's storage site.

Some solutions such as Semantic-chunks[41] and VFC for Cooperative Work [11] also take into account the **interest management**[16](or locality-awareness) of a user to filter massive volumes of data in large-scale distributed systems. These tools try to reason about the importance of each update, performing an intelligent management of updates and performing a selective scheduling based on this importance. Many times, users are only concerned with a small subset of the total sharing set of files. Nevertheless, systems waste resources by updating users with updates which are of no interest to them. Thus, the notion of interest management brings benefits by filtering data updates of low interest to users. Systems can enforce higher consistency guarantees over data subsets in which users are most concerned, and enforce low consistency guarantees over the others. This can have great impacts reducing the amount of updates and bandwith usage.

VFC-BOX combines deduplication techniques with an optimistic replication schema that is capable of adapt consistency guarantees according to the users interest. Through information provided by the user, the system is able to identify the user's interest over each data set. The benefits of this interest management are dual. First, data with higher interest can be forwarded to users in advance, reducing its latency and helping users to receive it ahead of data with less interest. Second, it reduces the number of updating messages regarding data of low interest. Additionally, deduplication techniques allow the system to reduce the redundant data stored and transfered between sites, reducing space and bandwith usage. Thus, the system aims to an efficient way of sharing data, reducing its redundancy and the amount of communications taking into account the subsets of data that are more relevant to users.

The remainder of the document is organized as follows. Section 2 describe the related work and briefly present some relevant systems. In section 3 we present the architecture of our solution. Section 4 presents the methodology that will be used to evaluate the system.

2 Related work

This section presents the state-of-the-art of work suitable for the sharing data system proposed in this document. Relevant design choices are studied and existing systems are presented and discussed.

The remainder of the section is organized as follows. Section 2.1 describes the related work on data redundancy, in which several data deduplication techniques used to minimize data storage and data transfer are presented. Section 2.2 addresses the topic of optimistic replication, in which several topics about the consistency of data are presented. Section 2.3 address the topic of adaptive consistency and presents the TACT and the VFC model, two replication models that adapt consistency guarantees according to the required. Section 2.4 describes the inherent problems of storage systems over large-scale networks (e.g. Internet) and addresses the topic of cloud computing, presenting Amazon S3, a scalable simple storage service system for the Internet that provides a highly scalable, reliable, secure, fast and inexpensive infrastructure for storing data. Finally, section 2.5 presents some of the relevant systems.

2.1 Data Redundancy

Nowadays, we may find that most computational systems have a large percentage of redundancy in stored data[22]. This redundancy is related to the fact that there is a high duplication of data parts, or a high similarity between distinct files. For instance, a system that keeps an history of multiple file versions has a substantial redundancy between multiple versions. This is because most of file versions only append a portion of data w.r.t. previous version, or if not, only modify a confined part of it. Apart from this, there are also other situations where we may find a high similarity between data, such as a file heading that several files use or even a piece of code that is auto-generated, and so duplicated across several files. There are two distinct types of redundancy. One is related with the redundancy between versions of the same file and is called *cross-version redundancy*. The other one is related to parts of information within a file that are similar to other parts within another file, and is called *cross-file redundancy*.

Data redundancy can be detected using **deduplication** techniques that compare the similarity between portions of data - called **chunks**. With this data similarity exploitation, higher performance in the storage of data may be achieved, by eliminating redundant data, and replacing it by references to a single instance storage of a chunk. Efficiently synchronization may also be achieved by not sending data that is found as redundant between two sites.

These techniques differ in 3 fundamental dimensions[22]: *algorithm*, *timing*, and *placement*. The choice of the algorithm depends on its efficiency in storage reduction, reconstruction time, network bandwith usage and impact in system's resource consumption. Current *algorithms* are divided in three main families: Delta-Encoding, Compare-by-hash and Version-Based deduplication. W.r.t. *timing*, deduplication can be performed as *Synchronous/In-Band*,

Asynchronous/Out-of-Band, or as *Semi-Synchronous* operation. Finally, regarding the *placement* of deduplication, it can be placed at the client side or at the server side. This placement choice may affect resource utilization, for instance, client-based placement might improve the bandwith utilization.

2.1.1 Delta-Encoding

Delta-Encoding[19, 13] is a deduplication technique that consists in the encoding of a file relatively to another. It is often used between versions of the same file, where the new version may contain a large part of the content of the previous version. This technique compares each byte of the file (binary diff) to be compressed against another one called reference file, and calculates a delta between them. This delta contains the modifications that were made to the reference file. The goal is to transfer and store only the delta files, keeping references to the original data regions, improving the storage space and bandwith usage (by transfering only the deltas).

To take advantage of this technique it is fulcral to detect pairs of files in which there is a high probability of existing data similarity. Applying this technique to two completely different files would end up in the storage of both files, without any storage gains. Thus, it is important to have an heuristic to help in the detection of reference files. Version control systems like CVS[10] and SVN[15] use Delta-Encoding with the naming of files as the heuristic. This heuristic explores most of the *cross-version redundancy*, since a file name is normally maintained through multiple versions.

As this technique needs to compare two files to exploit redundancy, it is only able to explore redundancy locally. This type of redundancy is called *locally trackable redundancy*[7] where data redundancy occurs only locally, and is the contrast to the *locally untrackable redundancy*[7], which exploits redundancy between two sites without the need to have both files in the same site. This is a limitation to systems that may want to explore similarities between data that came from multiple sites (*locally untrackable redundancy*). Another limitation of this technique is the inability to detect redundancy within a set of versions. As the algorithm makes use of the comparison between two files, it is only possible to detect similarities between them, and not within a set of files.

2.1.2 Compare-by-hash

Compare-by-hash[24, 39, 12, 5] is a deduplication technique based on the comparison of hash values of each chunk of data. These hash values have to be collision resistant so that we can assume that a content of a chunk is redundant relatively to another one, only by comparing the hash values[14, 32]. If one hash is equal to another one, we may say that the content of both chunks is identical.

This algorithm is thus capable of identifying either *cross-version* or *cross-file redundancy*, only by comparing hash values. It improves data storage by detecting chunks with the same hash value (same content), eliminating duplicates

and substituting them by references to a single instance storage. Furthermore, it is also capable of exploring *locally untrackable redundancy*, and improve the bandwith usage. This technique is able to achieve this by transferring hash values between two sites. For instance, a site S may have a file to send to site R. Instead of sending the whole file, S could only send the hash values of the data chunks. Then, R would only have to search locally for chunks with the same hash value, and return to S the information about the data chunks that are missing - called literal data. By the end, S would only have to send to R the literal data, avoiding to send chunks that R already has (redundant data). This process is depicted in Figure 1. Efficient synchronization may then be achieved by not sending data that is found as redundant between two sites. However, in terms of data transmission, this technique introduces a new round-trip to the transfer protocol and an increase in the volume of meta-data exchange.

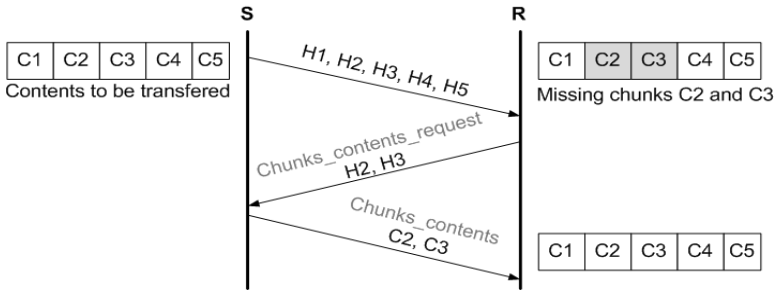

Figure 1: Example of the chunk transference protocol.

In terms of granularity, this technique can be separated in *Whole File Hashing*, *Fixed Block Hashing* or *Variable Block Hashing*.

Whole File Hashing (WFH):
The Whole File Hashing technique used by systems such as Single Instance Storage[8] consists on the hashing of a whole file and its comparison against others. It is the simplest technique of the compare-by-hash algorithms and it is a simple method to detect duplicated files. As it takes into account a whole file, it has not to calculate chunk boundaries, making it easier to implement and more efficient in terms of time processing. A SHA-1[14] or MD5[32] hash of the file can then be computed and compared to the pre-existing hashes in the system, in order to identify duplicates. Nevertheless, it needs an heuristic to detect similar data blocks, for e.g. the name of files (a simple file renaming breaks this heuristic). It is also unable to detect other forms of redundancy, not detecting redundancy between file versions, or between parts of files.

Fixed Block Hashing (FBH):
The Fixed Block Hashing technique consists in detecting data redundancy through the hashing of chunks of the same size. Systems such as Venti[29] and Rsync[39]

use this technique of partitioning files into chunks. When compressing a file, it has to split a file into chunks, calculating its boundaries according to a constant size (chosen *a priori*). Afterwards, it has to calculate their respective hash values (using a SHA-1 or MD5 hash over each block), and then to search for chunks with an equivalent hash. Thus, it is able to detect redundancy in a more fine-grained way, according to the specified size block. This is an improvement when compared to Whole File Hashing, achieving better compression rates. However, this approach is very sensitive to modifications performed on consecutive versions of a file. Since chunk's boundaries are calculated with a fixed size, a simple insertion of data may shift all the boundaries of the blocks from that point until the end of the file. Thus, these blocks will all be considered as new data blocks. Figure 2 illustrates this problem of overlapping chunks. This technique may be appropriate to situations where modifications to files are only appending data. Nevertheless, it is unable to detect a high percentage of redundancy when there is a shifting of chunk's boundaries.

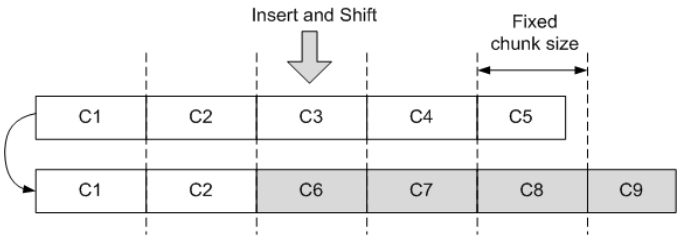

Figure 2: Example of application of FBH. The gray color identifies the chunks that were considered as new chunks.

Other problem inherent to this solution is to find the optimal block size, which is a non-trivial task. Further, it depends on the type of the data. For instance, bigger blocks work better with highly redundant data. However, a smaller block size is able to find more redundancy, especially in less similar data. But then, using smaller blocks requires more meta-data and, at some point, the size of the meta-data generated does not pay the savings in space by using smaller blocks.

Variable Block Hashing (VBH):
The Variable Block Hashing technique consists in splitting data into chunks according to its content. This content-defined chunking is used in many systems such as LBFS[24], Pastiche[12], BackupChunk[25], Haddock-FS[6] and ShiftBack[40]. Instead of calculating boundaries with a fixed sized, it calculates it having the content of a file into account. Thus, it avoids the problem of shifting of boundaries, improving efficiency in data compression.

To divide a file into chunks, the algorithm examines every 48-byte regions of the file and with a probability of 2^{-13} over each region's contents considers it to be the end of a data chunk. It can also be limited with a minimum and a maximum constant size (chosen *a priori*), to restrict very small or very big

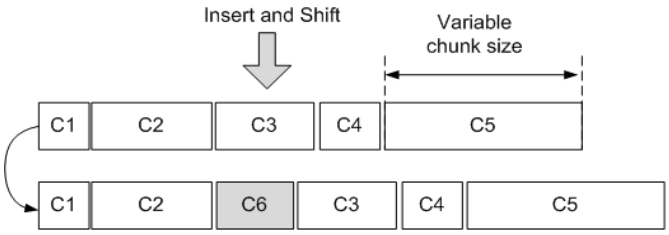

Figure 3: Example of application of VBH. The gray color identifies the chunks that were considered as new chunks.

chunks of data. After calculating chunk's boundaries, this technique only has to calculate a SHA-1 or a MD5 hash over each chunk of data, and to compare them to the pre-existing hashes in the system, in order to identify duplicates.

Current solutions calculate the content-based boundaries using *Rabin Fingerprints*[30]. A fingerprint consists in a polynomial representation of the data modulo. When the low-order 13 bits of a region's fingerprint is equal to a chosen value, that region constitutes a boundary. *Rabin Fingerprints* is normally used because it is efficient to compute it on a file sliding window.

Comparing VBH with FBH, we may say that VBH is a more efficient algorithm to find redundant chunks of data. Nevertheless, it is a more complex and computational expensive algorithm than FBH. When choosing between VBH and FBH, one has to take into account whether the system requires an algorithm more efficient in terms of redundancy detection or in terms of computational consumption. Furthermore, in systems where the problem of sliding boundaries does not often occur, the FBH is able to find redundancy close to levels obtained by VBH.

2.1.3 Version-Based Deduplication

As described before, compare-by-hash algorithms are very powerful to detect either *cross-file* or *cross-version redundancy*. Nevertheless, and concerning data transmission, it adds a significant overhead to the data transfer protocol. It adds to it more round-trips and a substantial volume of exchanged meta-data. When in presence of a low redundancy situation that overhead may not compensate the gains.

The Version-Based Deduplication[4, 7, 40, 25] appears to reduce this overhead introduced by compare-by-hash algorithms. Version-Based Deduplication is a technique used by systems such as dedupFS[4], ShiftBack[40] and BackupChunk[25], that combines versioning information with local similarity detection algorithms. It consists in the knowledge that each site has about the data that is stored in another site. Having that knowledge into account, one can avoid to send redundant data across the network. That knowledge is based in a space-efficient representation, such as version vectors[23]. The algorithm

works in 4 steps:

- A receiver site, R, informs the sender, S, of the version set that R currently stores.
- The sender site compares R's version set with its own, determining the intersection of versions that both contain in common.
- Using some local similarity detection algorithm (e.g. compare-by-hash), S determines which chunks to send to R are redundant with relation to the previous intersection.
- The sender S transfers the contents of the remaining literal chunks to R.

In order to achieve this, Version-Based deduplication imposes the unique identification of each write through the use of monotonically increasing local counters at each site. Thus, each write constitutes a version that is identified by the site's unique identifier and the the local write counter.

Each site maintains two version vectors to identify the knowledge that they have about the state of the other sites. Any site S holds a Knowledge Vector, denoted KV_S, and a Pruned Knowledge Vector, denoted PrV_S. The KV_S is a vector with an entry per known site and contains the last known state of each site, represented by the unique identification of the most recent version written by the other site and obtained by S. The PrV_S is a vector with an entry per known site and contains the last known versions that were pruned and no longer available at the site S. When S receives new data, it updates the KV_S to the value of the most recent version obtained by each site. Thus, S is able to represent the most recent version that it has ever seen from each site. When S removes some files, it updates the PrV_S, updating it to the most recent version that has been deleted. The interval between the KV_S and the PrV_S, $]PrV_S, KV_S]$, represents the set of data that is found at the site S. Nevertheless, S has to guarantee that for every version v for each site i it can respect the clause $PrV_S[i] < v \leq KV_S[i]$. In other words, S must guarantee that the set between the PrV_S and the KV_S has no gaps. To accomplish this, S must receive only consecutive updates and discard non-consecutive updates.

This technique achieves two fundamental properties: i) as the process of similarity detection is performed locally, it can employ more data-intensive techniques; ii) on contrary to compare-by-hash it does not introduce a substantial overhead w.r.t data transmission. Yet, as it only takes into account the knowledge of shared data, it does not detect *locally untrackable redundancy*. Depending on the situation, that issue may or may not be important. In most cases the redundancy comes mainly from *cross-version redundancy*. Thus, the gains may overcome this limitation.

2.1.4 Deduplication Timing

Data Deduplication Timing varies according to the time in which it is employed. It can be performed as *Synchronous/In-Band*, *Asynchronous/Out-of-Band*, or

as *Semi-Synchronous* operation.

***Synchronous/In-Band*:**
Synchronous deduplication[22] consists in performing deduplication operations when data is being consumed by the system. Furthermore, this means that for each write operation there is a deduplication operation associated, before the effective write occurrence.

This type of deduplication timing allows a search for redundancy before the actually occurrence of each write. As it never writes data into the system before compacting it, it allows a better space usage reduction. Nevertheless, this type of deduplication timing needs to process data as it is being written. Thus, a continuous overhead in the usage of computational resources is introduced. This causes some latency on writing operations, reducing by this the throughput of the system.

***Asynchronous/Out-of-Band*:**
On contrary to synchronous deduplication operation, asynchronous deduplication [22] consists in performing deduplication operations only periodically. From time to time, it executes a deduplication operation over the new written data, searching for redundancy and compacting the necessary data. By this, the system can perform its activities without being constraint by deduplication operations. Thus, increasing the consumption of system's data and its throughput. However, as data is written immediately, there is no processing before, allowing the writing of redundant data into the system. This requires a higher storage capacity, since it needs to stage data while uncompressed.

Normally, this type of deduplication is not a good solution when perform together with a client-based placement approach. This is firstly explained by the lack of up-to-date deduplicated meta-data at runtime, unabling to do queries immediately. Secondly, it causes a loss in the network bandwith reduction achieved by the client-based placement.

***Semi-Synchronous*:**
Semi-Synchronous deduplication[22] consists in a combination between synchronous and asynchronous deduplication, performing each technique when more adequate. It chooses the type of deduplication dynamically according to resource availability.

2.1.5 Deduplication Placement

Data Deduplication may be performed on the client side or on the server side according to the intended.

On the client side:
When data deduplication is performed on the client side, the client has the duty of performing deduplication operations. Thus, data is only transfered between sites after redundancy removal. As data is already sent in a compact form, this

can achieve higher performances w.r.t. data transmission. Typically, this approach involves a deduplication client that communicates with a server. Thus, the client processes data before synchronizing it with the server, sending to the server only the respective meta-data. With the received meta-data the server can search for identical chunks of data. Then, it informs back the client about the missing chunks. Therefore, the client only sends to the server the literal data, decreasing costs in the usage of network bandwith.

Despite the gains achieved in data transmission, this type of deduplication implies a higher resource usage on the client side, regarding CPU and IO operations. Taking this into account, the client could be affected in terms of performance of other applications.

On the server side:
Data deduplication performed on the server side consists in having a server appliance that executes deduplication operations on its received data. Venti[29] and Quantum[1] are examples of solutions these appliances. Normally, it is chosen having into account the benefits in the server's storage. On contrary to client-based deduplication, it does not overhead the client with the responsibility of processing data. Nevertheless, as redundant data is sent to the server, it is not so efficient in terms of data transmission.

2.2 Consistency in Distributed Systems

Replication is a fundamental technique in file sharing systems to improve availability, scalability, performance and to support disconnected operations. However, these systems have a difficult task ensuring replica consistency across several users. There is a lack of adaptability and efficiency on these systems regarding replica consistency. Most of them are not capable of scale to large networks, due to the huge bottleneck in data transmission. This section presents some replication models, comparing its advantages and disadvantages.

2.2.1 Limitations of Pessimistic Replication

Pessimistic Replication[33] is a model to ensure strong data consistency guarantees. It tries to guarantee that all the replicas are identical to a single copy. Further, for any sequence of read and write operations on a replicated object, it guarantees that the sequence of values associated are the same for any other replica. Thus, a sequence of reads and writes on a replicated object will produce the same effect as if the object were not replicated. To ensure data consistency at the level of a non replicated schema, this model has to block access to data whenever a replica is not up-to-date or disconnected from network. Before performing any operation request over a replica, the pessimistic replication runs a synchronous coordination protocol to ensure that the requested operation will

[1] Quantum: Data de-duplication overview.
http://www.quantum.com/Solutions/datadeduplication/Index.aspx.

not violate any consistency guarantee. So, it preserves consistency of data preventing conflicts, even at the cost of denying access to replicas.

This model of replication specially fits to systems where stale data cannot be read or data conflicts cannot occur. However, it comes with the cost of reducing data availability, which is a big constraint to file sharing systems. Also, it is difficult to scale systems that use pessimistic replication to larger networks. Its natural frequency of updates causes system's throughput and availability to suffer as the number of sites increases.

2.2.2 Introduction to Optimistic Replication

Optimistic replication[33] is a replication model for sharing data efficiently in wide-area or mobile environments. In opposition to pessimistically replicated systems, its approach is based on the improvement of concurrency.

It consists on the guarantee that object replicas will converge to the same value, within a certain period of time. This convergence is called eventual consistency[33, 43], and assumes that for a long period of time, all updates will eventually propagate through all the replicas. On contrary to Pessimistic Replication, this model does not block access to data even at the cost of suffering some divergence between replicas. Optimistic Replication does not have to run any type of coordination protocol before accepting an operation request. Thus, it overcomes Pessimistic Replication regarding access performance, as the replicated system no longer waits for a synchronization before accepting a request. Concerning scalability, Optimistic solutions are also preferred due to less coordination requirements and due to the possibility of running synchronization protocols in background. Thereby, it trades data consistency for availability and scalability.

With this temporarily relaxed consistency, stale reads and conflicting writes are inherent risks. A classic approach to this problem is to ignore the stale reads and to detect and resolve conflicting writes. Unison[28](file synchronizer) describes 4 types of existing conflicts:

- change the name of the same file to different names on different replicas;
- delete a file on one replica and change its name on another;
- create a file with the same name and different contents on different replicas;
- make different modifications to the same content of a file on different replicas.

The option that Unison makes to resolving conflicts is to detect them and then request users to solve the conflicts. Most of file synchronizers, such as Dropbox[2] and SugarSync[3] take this option to solve conflicts. Others, such as CVS[10] and SVN[15] try to merge different modifications to the content of a file, asking the

[2]Dropbox: Secure backup, sync and sharing made easy. https://www.dropbox.com.
[3]Sugarsync: Backup and file synchronizer. https://www.sugarsync.com/.

help of the user only when this attempt failures.

Optimistic replication normally fits to systems that can tolerate some divergence between replicas and where conflicts are very rare. To these type of systems, optimistic replication can bring several advantages such as availability, scalability and the support of disconnected operations. Many, also indicated this model as the appropriate to some human activities, as it is better to allow collaborators (system users) to update data independently and repair occasional conflicts than to lock data while someone is editing it [10].

Distributed file systems are an example of systems that usually fit to an optimistic replication model, where usually conflicts are very unlikely to happen cite[44].

The goal of any optimistic replication system is to provide consistency while improving availability and scalability. However, there are some design choices according to the requirements of each system. On the following we describe some of those design choices, namely *update submission*, *update propagation*, *update transfer*, *Operations Scheduling* and *Conflict Resolution*.

Update Submission: Single-Master vs. Multi-Master [33]
Update submission regards where an update can be submitted to and how it is propagated. This can be divided in two main submission forms, Single-Master and Multi-Master. Single-Master consists on the submission of updates exclusively to one replica (master), and its propagation from that replica to the others (slaves). Since updates are all submitted to one replica, this type of systems can detect and solve conflicts in a centralized way. Besides its simplicity, they may have some limited availability caused by the bottleneck in the master replica when experiencing frequent updates.

Multi-Master consists on submitting updates to multiple replicas independently and its propagation in the background. Updates can be submitted to any replica, existing by this a decentralized way of detecting and solving conflicts. In comparison to single-master systems, these improve availability with the cost of a significantly more complex system. Nevertheless, and w.r.t. scalability it can be a problem due to the increased conflict rate.

Update Propagation: Push vs. Pull Model [33]
Update propagation regards the model that is used when there is an update to be propagated. There are two main models, namely push and pull model. On the push model used by systems such as Bayou[27] and Roam[31], a replica holding an update is responsible for pushing it to other replicas. On the pull model there is the concept of polling replicas in order to request the new updates. This polling process can be manually triggered or automatically using a periodical signal.

This design choice can have an important impact on systems regarding scalability, due to the overhead associated to periodic polling (pull model) or due to the high frequency of update propagation. Systems like Coda[35] use hybrid

solutions to take both advantages of each model.

Update Transfer: State vs. Operation Transfer [33]
State and Operation Transfer are variants of update transfer and refer to what is transfered between replicas when there is an update to be propagated. On state-transfer systems, replicas are required to read or to overwrite a entire object. When reading/writing an update, a replica has to read/write the whole object to which the update concern. It is a simple model of propagating updates that can easily and transparently be adapted to any solution, since maintaining consistency only involves sending the newest replica contents to other replicas.

On operation-transfer systems, replicas are required to propagate only the operations/modifications related to an update. In comparison to state-transfer systems, they can be more efficient since they do not need to send entire objects. Additionally, this model also improves concurrency and a lower conflict rate, since operations may be commutative. Yet, these systems are more complex as they need to reconstruct an history of operations.

Some systems like LBFS[24], Semantic-chunks[41] and Haddock-FS[6] make use of an efficient representation of updates to achieve a mixing of the two solutions (State and Operation Transfer). They use chunks as a representation for updates. As already mentioned on this document, chunks are portions of data within a file. With this notion, both advantages of the state transfer and operation transfer may be achieved. When a chunk is modified, its update involves the transfer of the whole chunk, making the transfer process easier as achieved by the state-transfer model. Yet, when a file is modified, the whole file does not have to be transferred, since only the affected chunks have to be transferred. Thus, it improves concurrency and decreases the conflict rate as achieved by the operation-transfer model.

Operations Scheduling: Syntactic vs. Semantic [33]
Operations Scheduling regards to the ordering of operations in a way that produces equivalent and expected states across users. There are two policies to produce the ordering of operations, namely Syntactic and Semantic scheduling. Syntactic scheduling is based on the time in which operations happened, preserving an operation ordering according to the relationship *happens-before* defined by Lamport[20]. This scheduling method is simpler than the semantic scheduling due to unnecessary knowledge about the semantic of operations. Nevertheless, as it does not examine the semantics of operations, it is not able to order operations in a way that can cause less conflicts. Semantic scheduling is based on the ordering of operations according to the operation's semantics. Thus, this method is able to reduce the conflict occurrence and increase the merging process of different operations. This policy is more complex than syntactic and is only applicable to operation-transfer systems, since state-transfer systems do not take into account the semantics of operations.

Conflict Resolution: Manual vs. Automatic [33]
A conflict occurs when a precondition of the system's scheduling is violated. The detection of conflicts may be based on a syntactic approach, when the *happens-before*[20] relationship is violated (e.g. when two or more operations are concurrently applied). On the other hand, conflict detection may be based on a semantic approach, which identifies conflicts according to the violation of precondition related with the semantics of the application.

W.r.t. conflict resolution/reconciliation[9], it may be performed manually or automatically. When performed manually, the conflict is detected and then delegated to the user to resolve. When performed automatically such as in Bayou[27], the conflict is resolved according to a set of rules defined by the application. These techniques try to reconcile and merge updates that respect to the same object. Systems like rcsmerge[38] try to merge updates through techniques based on plain text files. Systems like Semantic diff[18] try to merge updates based on the particular context of the application. In certain situations, this attempt of reconciliation may failure for example due to non-commutative actions and on this case the reconciliation has to be delegated to the user.

2.3 Adaptive Data Consistency

Many times, designers of replicated services are forced to choose either to use strong consistency guarantees or none at all, in order to cope with system's requirements. In this section we introduce the topic of adaptive consistency. In systems such as Haddock-FS[6] and IDEA[21], instead of having a static model of ensuring consistency guarantees, they use a model where the system adapts to the environment ensuring strong and weaken consistency guarantees as it is appropriate. With this model, systems may preserve strong consistency guarantees while improving scalability and efficiency.

On the following we present TACT, a middleware layer that enforces arbitrary consistency bounds amongst replicas. Further we introduce *locality-awareness*, a notion that has been widely researched and that can be used to achieve adaptive consistency systems. Finally, we present Vector-Field Consistency, an optimistic replication model that adapts consistency through locality-awareness techniques.

2.3.1 TACT - A Consistency Model for Replicated Services

TACT[46, 47] is an efficient and adaptable consistency model based on optimistic replication. Instead of having a model where either strong or weak consistency guarantees are enforced, TACT permits a consistency enforcement over multiple levels of consistency guarantees. With this concept of multiple consistency levels, it is possible to adapt consistency guarantees according to requirements (availability, performance, probability of inconsistent access). To achieve this consistency multiple level, TACT permits a bounded divergence between object replicas in accordance to a maximum level of inconsistency. This model

proposes three metrics to bound consistency:

- **Numerical Error:** limits the total weight of writes that can be applied across all replicas before being propagated to a given replica.
- **Order Error:** limits the number of tentative writes that can be outstanding at any one replica.
- **Staleness:** limits the time delay of write propagation amongst replicas.

In order to specify consistency levels, applications specify their application-specific consistency semantics using *conits*. A *conit* is a physical or logical unit of consistency where applications quantify consistency continuously along a three-dimensional vector:

Consistency = (Numerical Error, Order Error, Staleness)

According to consistency semantics, TACT is able to efficiently manage the propagation of updates, delaying updates that do not violate consistency bounds. Thus, TACT reduces the use of network resources and masquerades latency, while adjusting consistency guarantees in accordance to the application semantics.

2.3.2 Locality-awareness in Large-scale Systems

To achieve system's scalability over large networks such as the Internet, there is a need to reduce the amount of exchanged data between multiple entities. The notion of Locality-awareness is associated with Interest Management[16, 36, 37] that is a filtering mechanism that aims to solve the scalability problem through techniques that take into account the users' interest. Systems such as MANET(Mobile Ad-hoc Networks)[37] use techniques to detect the users shared interest in some topics in order to filter messages of low interest to them. Therefore, a higher scalability may be achieve by filtering massive volumes of data and thus reducing the volume of exchanged data that would be found as no interesting.

Locality-awareness is also a form of interest management, detecting users interest based on their locality. For instance, in massively multiplayer games, middlewares such as Matrix(Adaptive Middleware for Distributed Multiplayer Games)[2], VFC for Ad-hoc Gaming[34] and Unifying Divergence Bounding and Locality Awareness in Replicated Systems with VFC[42] can track players position. According to it, they can strengthen consistency guarantees around the player position and weaken it as the distance to the player position increases. This model can achieve a higher scalability since it can adapt consistency levels according to the need and to the associated interest.

Current solutions such as Semantic-chunks[41] also use the notion of locality-awareness to efficiently share files between multiple users, taking into account information provided by the user, regarding their interests over each data set.

This can improve these type of systems not only by reducing the overload of the network but also by reducing the latency, helping users to get important data in advance.

2.3.3 Vector-Field Consistency (VFC)

Vector-Field Consistency[42, 34] is an efficient and adaptable consistency model based on optimistic replication. It permits a bounded divergence between object replicas. For this, it takes into account several forms of consistency enforcement and a multi-dimensional criteria (time, sequence and value) to limit replica divergence. These forms of consistency are determined through techniques based on locality-awareness. Thereby, it uses locality-awareness techniques to identify different zones (consider a zone as a subset of a sharing set of data), in which the VFC dynamically strengthens/weakens replica consistency. Different multi-dimensional criterias are then applied to different zones, creating different divergence bounds to each zone.

To identify different zones, VFC uses the concept of *Pivots*. Pivots are based in locality-awareness techniques and they identify points in which consistency around is required to be strong, and weaker as the distance from the pivot increases. Figure 4 illustrates an example with 3 consistency zones, where a concentric circle was chosen as the space delimiter. The object O_3 was chosen as pivot. Therefore, VFC would enforce stronger consistency within Z_1, following with Z_2 and by last Z_3. Each object covered in each zone, would then have applied different divergence constraints, and thereby different consistency guarantees.

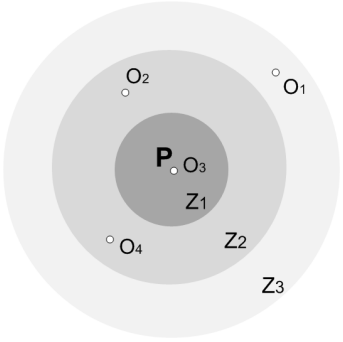

Figure 4: Consistency zones centered on a pivot.

To ensure different forms of consistency, VFC provides a 3-dimensional vector, $\kappa = [\theta, \sigma, \nu]$, to specify the consistency degrees. Each dimension of the vector bounds the maximum objects divergence in a particular view.

Each dimension is a numerical scalar defining the maximum divergence of the constraints time (θ), sequence (σ), and value (ν), respectively.

- **Time:** Specifies the maximum time a replica can be without being re-

freshed with its latest value.

- **Sequence:** Specifies the maximum number of lost replica updates.
- **Value:** Specifies the maximum relative difference between replica contents or against a constant.

For example, consider a consistency vector $\kappa = [0.1, 6, 20]$, that describes the divergence bounds of each constraint (time, sequence, value). It represents that replicas can only be outdated for 0.1 seconds or 6 lost updates or with a 20% variation in the replica content.

Through a selective form of increasing and decreasing consistency enforcement, VFC is able to ensure critical updates to be immediately sent and less critical to be postponed. Thereby, it makes an efficient resource usage, reducing the network bandwith usage and masquerading latency.

2.4 Cloud Computing for Large-Scale Storage

Cloud computing[1] is a modern concept of using software and hardware infrastructures as a service over the Internet. It pretends to provide high performance computing as a cloud, where users may run their programs or store their data without having to concern about the management of the background system/infrastructure. As such, instead of having to manage large infrastructures and develop systems to provide high availability and scalability, cloud users can use the cloud as a service, paying only for what they use.

On contrary to client-server architectures, cloud computing provides an abstraction over the details of individual servers. Instead of performing requests to a server, cloud users may perform requests as a service, without the need of being concerned about the physical location of servers or cloud computing infrastructure. As such, users do not require any knowledge regarding the control and management of the remote services, as they are handled by cloud providers.

The most attractive features of cloud computing are: **i)** *Cost Reductions*: fewer IT skills, fewer implementation requirements and no waste of power and computing resources; **ii)** *Scalability*: mechanisms may auto-scale functionality according to users requirements. Developers do not need to concern about peak loads, as the cloud system scale resources; **iii)** *Availability*: improved if multiple redundant sites are used. Possibility of using multiple cloud providers; and **iv)** *Reliability*: improved if multiple redundant sites are used Possibility of using multiple cloud providers.

There are several types of services that may be provided by cloud computing. Nevertheless, for this particular document we focus on cloud computing as a system to provide large-scale storage. Many systems such as Dropbox and SugarSync use cloud systems in order to provide high available and reliable storage.

2.4.1 Amazon S3 (Simple Storage Service)

Amazon S3[4] (Simple Storage Service)[26] is an Amazon's system based on cloud computing to provide storage for the Internet. It provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. Through Amazon S3, developers may achieve highly scalable, reliable, secure, fast and inexpensive infrastructures to store data, without having to be concerned about any internal issues.
Amazon S3 is supported by a large number of data centers in the United States and Europe and is expected to offer low data access latency, infinite data durability and 99.99% of availability[26].

Data stored in Amazon S3 is organized in a two level namespace: *buckets* and *object names*. *Buckets* are similar to folders and allow users to organize their data. *Object names* correspond to objects that are stored into buckets.

Regarding the data access protocols, Amazon S3 supports 3 main protocols: SOAP[5], REST[6] and BitTorrent[7].

2.5 Relevant Systems

2.5.1 VFC for Cooperative Work

VFC for Cooperative Work[11] is a synchronization tool to efficiently synchronize Latex documents amongst collaborators. It uses the concept of locality-awareness to intelligently manage the transfer of updates. Through techniques that track the editing position of a user within a document, the system is able to reason about the importance of sections of a Latex document. With this information it performs a selective scheduling of update propagation, enforcing strong consistency guarantees to most important updates and weaker consistency guarantees to less important updates. This system divides Latex documents into chunks, which can be considered as sections or paragraphs of a document. Multiple consistency guarantees are then applied to multiple chunks according to clients locality.

To accomplish the dynamically adaptation of consistency guarantees, VFC for Cooperative Work uses the Vector-Field Consistency model. With this model, the system is able to assign multiple consistency guarantees (creating bounds of inconsistency) to multiple sections of a Latex document.

In comparison to systems that ensure total consistency, VFC for Cooperative Work is able to achieve higher performances w.r.t. bandwith usage. Additionally, this system is able to improve concurrency and reduce the conflict rate, since modifications to different chunks of a document (different sections for example) are not considered as conflicts as they can be merged and constitute

[4] Amazon s3. http://aws.amazon.com/s3/.
[5] http://www.w3.org/TR/soap/
[6] http://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm
[7] http://www.bittorrent.com

a single version. Nevertheless, this system does not perform a compression of transfered data, being unable to reduce even more the use of network resources. It also does not contemplate a compression of stored data.

2.5.2 Dropbox

Dropbox[8] is a commercial backup and file synchronizer[3] system that enables users to share data with others across the Internet. It is designed to achieve a high performance regarding the transfer of data between clients and servers. To provide and support storage to large-scale networks, Dropbox uses Amazon S3 (described in 2.4.1) to store files.

Each Dropbox client has in his computer a "Dropbox Folder" where he can modify and upload new files/folders. This folder is managed by a background process that is responsible for the correct synchronization of the folder between clients and servers. This application can also be used by several users as a collaboration tool as a user can allow others to access specific folders inside his "Dropbox folder".

To accomplish the high performance w.r.t. data transfer, Dropbox uses delta-enconding techniques to produce a "binary diff" between new and previous versions of a file. As such, it enables an efficient syncing, only uploading changes made to a file. It also enables a file versioning allowing clients to fetch previous versions. To detect the existence of modified data, Dropbox also uses Compare-by-Hash techniques over folders to find out which folders have been modified. For this, the system exchanges folder hashes in order to find if the contents of a given folder have been modified.

With delta-encoding Dropbox is able to reduce the use of bandwith, by only uploading the changes performed over a file. However, it does not perform deduplication to reduce the amount of stored data. Also, it does not use any kind of technology to specify multiple consistency levels, which could improve even more the efficiency in data transfer. Moreover, Dropbox does not provide any kind of tools to resolve updating conflicts, delegating to clients this task when any concurrent operations to the same files have been performed.

2.5.3 Haddock-FS

Haddock-FS[6] is a peer-to-peer replicated file system. It is designed for mobile ad-hoc environments where constraints of reduced memory and low bandwith are usual. Haddock-FS permits collaborative operations, detecting and solving conflicts by comparing multiple versions. To accomplish this, it uses a consistency protocol that relies on *dynamic version vectors*[31]. Haddock-FS is based on an update log system that organizes operations as tentative or as stable, according to the state of updates. Tentative updates are reversible on contrary to the stable updates. Stable updates are selected by a single replica called

[8] Dropbox: Secure backup, sync and sharing made easy. https://www.dropbox.com.

primary replica.

W.r.t. data redundancy, Haddock-FS makes use of compare-by-hash techniques in order to improve the bandwith usage and to reduce the memory used by peers. To deal with the problem of shifting file offsets and overlapping chunks, this system uses Variable-size Block Hashing, basing chunk boundaries on file contents.

Haddock-FS is based on an adaptable optimistic consistency protocol, providing a highly available access to a weakly consistent view of files, while delivering a strongly consistent view to more demanding applications.

Briefly, Haddock-FS is able to reduce resources consumption regarding network bandwith and memory. It makes use of deduplication techniques to either explore *cross-file* or *cross-version redundancy*. Additionally, it is also able to detect *locally untrackable redundancy*. Regarding the consistency protocol, it makes an efficient use of resources by enforcing either weak or strong consistency guarantees according to the required. Nevertheless, this system is not able to enforce multiple levels of consistency guarantees. By enforcing multiple levels of consistency guarantees, this system could balance resources and requirements, instead of forcing applications to choose between weak or strong consistency guarantees. Further, as Haddock-FS makes use of compare-by-hash techniques, it introduces an overhead regarding the meta-data exchange, which may not compensate the gains over low redundancy situations.

2.5.4 LBFS

LBFS[24] is a network file system designed to perform in low-bandwith networks. The main goal of this system is to avoid the transmission of data that may already be found at the receiver's site. To accomplish this, this system makes use of compare-by-hash techniques in order to improve the bandwith usage. To deal with the problem of shifting file offsets and overlapping chunks, this system uses Variable-size Block Hashing, basing chunk boundaries on file contents.

To make the chunk comparison possible both client and server store chunks in a database indexed by chunks hashes. When reading a file from the server, the client makes a request to the server in order to retrieve the hashes of the chunks to be read. Further, the client compares the received hashes with the already detained, labeling the chunks that were not found as missing. After this, the client requests the missing chunks to the server, receiving by this the missing data. As these operations are all pipelined, downloading a file only incurs in two network round-trips plus the cost of downloading the data.

When writing back a modified file, the opposite of the reading process is done. Firstly, the client sends the hashes and only after the missing data. To avoid dealing with the reordering of writes, LBFS implements atomic updates and a Close-To-Open consistency model. Thus, the commitment of updates is only applied when a file is closed. Additionally, when a file is closed by a client

and another client reads it, it always receives its last content.

To achieve atomic updates, LBFS makes use of temporary files, in which updates are incrementally written. When the file is closed, the temporary file is then committed, overwriting the previous version of the file. First the client sends a "create temporary file" request to the server, to which the client will write the updates. Then, it sends the hashes of the chunks that compose the new version of the file. The server will return with a "missing chunk response" or with an "Ok response", which indicates that the server already detains that chunk. Missing data is pipelined from the client to the server, and at close time the client requests a commitment of the file.

Summing up, LBFS is a system that efficiently synchronize data, saving resources regarding the use of network bandwith. Although it is clear the improvement in the transfer protocol, this system has to exchange sets of hash values between client and server, which in case of low redundancy may not compensate the gains and introduce a substantial overhead. Moreover, it does not explore data redundancy to efficiently store data. This system could use the same deduplication techniques to explore this last issue, making this a system that could efficiently transfer and store data. Additionally, LBFS also does not have into account multiple consistency guarantees, which could guarantee a multi-level consistency according to the bandwith constraints and user needs. This could even improve more the efficiency w.r.t. data transmission.

2.5.5 redFS

redFS[7] is a distributed file system that performs *locally trackable deduplication* in order to achieve an efficient synchronization. The synchronization process of redFS is composed by two main steps: I) version tracking - responsible for detecting the set of versions that two synchronizing sites share in common; II) local redundancy detection - responsible for detecting local redundancy.

To perform the local redundancy detection, redFS makes use of Variable-size Hashing in addition with a byte-by-byte comparison technique. redFS uses simple hash functions over data chunks in order to detect similar chunks. After detecting the similarity it uses a byte-by-byte comparison technique to confirm the chunks similarity, preventing hash collisions. With this techniques, redFS is able to detect the common versions (common data chunks) between two sites and reduce redundancy with local deduplication techniques, avoiding the transfer of local redundant data and data that is already found at the receiver's site. Thus, when transferring data between two sites, redundant data is substituted by references to the actual chunks. Thus, only the non-redundant data (literal data) is sent over the network.

Summing up, redFS may be able to achieve better data transfer performance than systems based on Compare-By-Hash and Delta-Encoding, while maintaining the integrity of data during this operation.

2.5.6 Semantic-chunks

Semantic-chunks[41] is a middleware that aims to efficiently enforce data consistency for cooperative work systems. It uses Compare-by-Hash techniques with Variable-size Block Hashing to exploit data redundancy either locally or between multiple sites. Thus, it is able to detect either *locally trackable* or *locally untrackable redundancy*, being able to improve the efficiency of storing and syncing data.

Additionally, Semantic-chunks have presented a model in which documents are partitioned into chunks, and each chunk is annotated with semantic consistency information. These chunks are called semantic-chunks and consist in semantically-annotated document regions with relevance to applications and users, further annotated with consistency information and enforcement. The consistency information is in part provided by users. With this semantical structure a user is capable of know the structure of a given document, without the need of having to download the whole document content. As such, a user can work over parts of a document without the need of having the whole content of it. Thus, Semantic-chunks can ensure consistency over semantic-chunks of important relevance to a user, and relax consistency over less important chunks. With this concept, this system is able to improve concurrency and reduce update conflicts. Moreover, it reduces bandwith usage, not only by exploiting data redundancy, but also by reducing and postponing the transmission of semantic-chunks with low relevance to a user. As Semantic-chunks uses Compare-by-Hash techniques, sometimes the introduced overhead for hash exchange between multiple sites, may not compensate the gains achieved by deduplication due to low redundancy. To overwhelm this situation, this system could use more efficient ways of representing the chunks knowledge.

2.5.7 ShiftBack

ShiftBack[40] is a backup system based in a client-server architecture, designed to support an efficient storage and network bandwith use. Additionally, this system also supports a task-oriented backup with a time-shifting interface which enables the browsing and retrieval of versions from any point in time. To cope with the requirements it uses Version-Based Deduplication techniques to explore *locally trackable redundancy*.

ShiftBack supports 3 main operations: Data Backup, Data Index and Data Recovery. Data Backup is the most frequent operation and consists in backing up data from the client to the server, storing and transferring only one occurrence of each chunk. This operation starts at the client side, that requests to the server its current state (knowledge vector). After this, it increments the knowledge vector and uses a Variable-size Block Hashing technique to find redundant chunks of data. Finally it returns to the server the literal data and its meta-data.

The local redundancy detection, illustrated in Figure 5, is accomplished within 3 main steps:

- Data partitioning using a rolling hash (variable-size blocks);
- Hash Calculation over the variable-sized blocks;
- Chunk repository lookup.

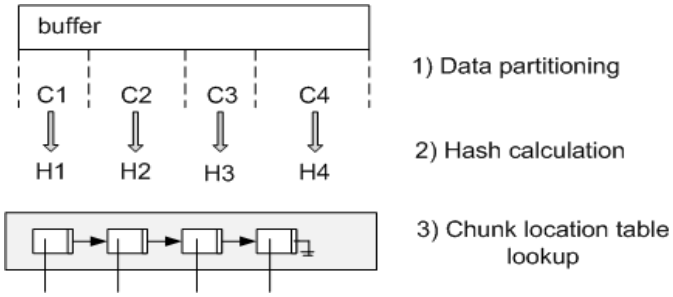

Figure 5: Example of deduplication process.

The chunk location table lookup is the process that involves finding a chunk with the same hash value (similar content). ShiftBack performs a search operation in which it tries to find a hash value. If the process succeeds, the chunk is substituted by a reference link to the single instance chunk. Otherwise, The new hash value is inserted in the chunk location table for further searches.

Data Index is a server-side only operation that consists in indexing the data stored at the server during backup, creating mappings between each backed up chunk's hash value and its location.

Data Recovery consists in the operation that retrieves the backed up data to the client. To do this, the server makes use of the mappings created by the data index operation, in order to find the chunks to be retrieved.

A special feature of ShiftBack is its high level of pipelining, being able to perform deduplication operations over sets of data while others are already being sent. Moreover, ShiftBack uses a pipe and filter architecture which allows each filter to run on a different thread enabling a better use of the CPU at the client.

ShiftBack provides a high efficient protocol to backup data through low-bandwith networks, reducing data transfer and achieving better performances than other solutions based in Delta-Encoding and Compare-by-Hash techniques. This is due to the use of lightweight structures (knowledge vectors) to represent the whole state of a site, reducing the amount of exchanged meta-data. However, this system does not provide any form to reduce/postpone the exchange of data having into account the importance of the backed up data. This system could

use a model to somehow ensure that important data is immediately backed up, and less important data is postponed to moments of high-bandwith connection.

2.5.8 Subversion (SVN)

Subversion (SVN)[15] is a revision control system typically used to synchronize and store multiple versions of source code files. SVN is based on a client-server architecture. It supports disconnected operations and provides to clients tools for handling conflicting updates, since normally there are multiple clients making concurrent changes to the same files.

In order to achieve higher performances w.r.t. data transfer protocol, SVN tries to reduce the use of network resources through the use of delta-encoding techniques. Through this technique it compares file versions with their previous versions, detecting *cross-version redundancy*. This redundancy exploitation is not only used to reduce the use of network bandwith, but also to achieve better performance w.r.t. data storage. As specified, the delta-encoding technique needs one new version and one old version to encode data, which forces the client to use extra space to store old versions. Furthermore, this method also imposes the limitation that each file is encoded only against one other file, which makes SVN unable to exploit *cross-file redundancy*.

In short, SVN is able to perform efficient data transfer, improving concurrency and providing tools to reconcile conflicting updates.

2.6 Summary

The following table2.6 presents a summary of the above mentioned systems w.r.t. deduplication techniques and data synchronization models.

Many systems do not have any mechanisms to adapt consistency guarantees according to the needs/resources. Although some systems provide mechanisms to adapt data consistency, they do not provide multiple levels of consistency. These, either enforce weak or strong consistency guarantees, becoming unable to provide intermediate consistency guarantees.

Further, most systems are not capable of taking into account the interests of users in order to propagate updates of the shared data. Nevertheless, there are some few systems that provide these features. Yet, they lack of efficient mechanisms to explore redundant data in order to either reduce storage space or network bandwith.

System	Description	Deduplication Algorithms	Adaptive Data Consistency Mechanisms
Dropbox	Commercial online backup system	Delta-Encoding	None
Haddock-FS[6]	Distributed file system	Compare-by-hash (Variable-Size Hashing)	Hybrid consistency: weak or strong consistency guarantees according to the resource constraints
LBFS[24]	Network file system	Compare-by-hash (Variable-Size Hashing)	None
redFS[7]	Distributed file system	Version-Based deduplication + Variable-Size Hashing + byte-by-byte comparison	None
Semantic-chunks[41]	Middleware for ubiquitous cooperative work	Compare-by-hash (Variable-Size Hashing)	Several consistency levels according to user provided information
ShiftBack[40]	Backup system	Version-Based deduplication + Variable-Size Hashing	None
SVN[15]	Revision control system	Delta-Encoding	None
VFC for C. W.[11]	Synchronization tool	None	Several consistency levels according to user locality

Table 1: Comparison between the studied systems.

3 Architecture

This section presents the architecture of the proposed file sharing system, named VFC-BOX. In section 3.1 we present the main architectural decisions regarding deduplication and replication model. In section 3.2 we describe a global overview of the system's architecture.

3.1 Architectural Decisions

3.1.1 Data Deduplication

To achieve a high performance both in terms of storage and bandwith reduction, the Version-Based Deduplication technique has been selected. Additionally, a

Variable-size Block Hashing technique has been selected to perform compare-by-hash techniques locally. With these techniques, both storage and bandwith usage may have several gains. Variable-size Block Hashing permits a fine-grained redundancy exploitation, which can reduce space used to store data and bandwith usage since redundant data are no more sent over the network. Version-Based Deduplication permits an efficient way of transmitting the information about the knowledge of each synchronizing site. This feature is an advantage due to the low overhead introduced.

3.1.2 Replication Model

In order to prevail the features of availability, scalability, improved concurrency and to support disconnected operations, we decided to choose an optimistic replication model. W.r.t. conflict resolution we decided to take the approach of trying to merge conflicting document updates. This approach is based on the attempt of merging updates that have application to different chunks. In the case of having conflicting writes on the same chunk, we decided to delegate the reconciliation to the user.

We identified the VFC model as a natural fitting model to this environment, where its locality-awareness techniques can be applied from the most to the less important user's data. By this, VFC can impose strong consistency guarantees to parts of data that are of extreme interest to users. Or relax some consistency guarantees to parts of data of less interest to users. Figure 6 illustrates a file

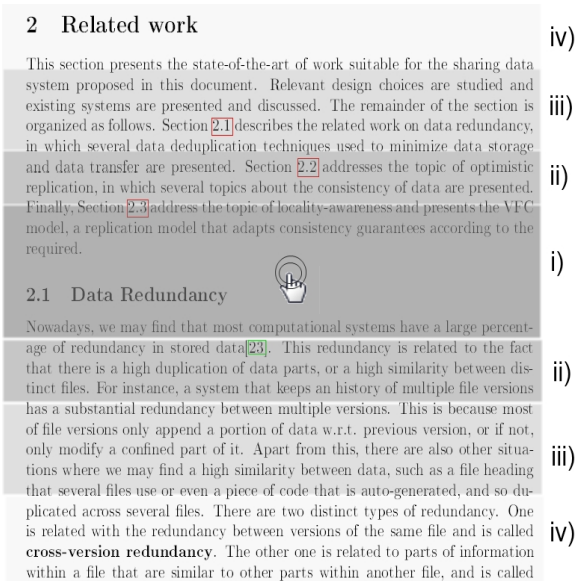

Figure 6: A file partitioned in chunks defining multiple consistency zones. This example illustrates how VFC can be adapted to documents.

can be partitioned into several sections, defining multiple consistency zones. In

this example, the user defined a section that is considered as important in the document, in which is required stronger consistency guarantees. The rest of the sections are then considered as less important, and therefore have applied weaker consistency guarantees. In this example, 4 consistency zones are identified. These zones could be classified as follows and have applied an according consistency guarantee level: **i)** very important section; **ii)** related section; **iii)** section likely to be related; **iv)** unrelated section. These zones could be mapped to chunks (defined by deduplication operations) and therefore the enforcement of consistency guarantees could be applied at the level of data chunks.

3.2 Architecture Global Overview

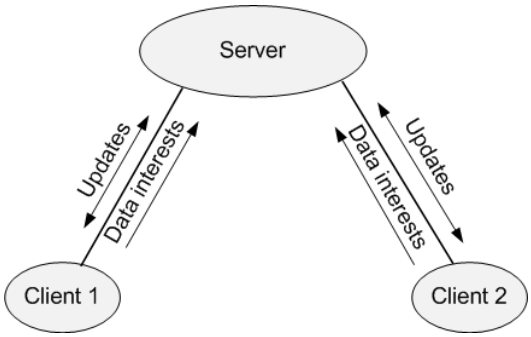

Figure 7: VFC-BOX's main overview.

VFC-BOX is based on a client-server architecture. Figure 7 illustrates the process where clients submit their updates and data's interests to the server. These interests represent parts (or chapters) of files in which clients have special interest. Taking these interests into account, the server is then able to enforce multiple consistency guarantees over multiple data subsets. Thus, the process is based on the exchange of updates between clients and server, and on the submission of client's interests to the server. These interests are then used to make decisions of which updates have to be immediately propagated and which can be stored for a while and only later be sent.

To achieve the above proposed, we suggest an interface that can adapt to different file types and in which clients can specify their interests over parts of data (Figure 6 may represent on scenario for this). This interface is responsible for the integration with different file types (e.g. Office files) and to collect users interests.

3.2.1 Deduplication Process

The deduplication process of VFC-BOX will be inspired on the architecture of the ShiftBack[40] system. In short, the process is composed by three main steps: I) Data partitioning; II) Hash Calculation; III) Lookup in chunk repository.

3.2.2 Data Transfer Protocol

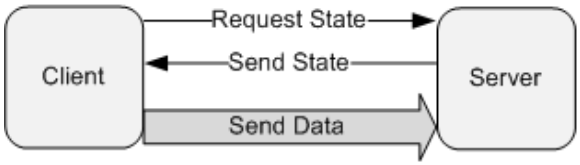

Figure 8: VFC-BOX's data transfer protocol.

The data transfer protocol will be inspired on the architecture of the ShiftBack[40] system. Figure8 illustrates how the system manages to transfer data. First, the client has to make a request to the server in order to obtain server's state. After receiving the state of the server, the client is then able to remove redundancy from the data to be transfered, according to the knowledge of the server. By last, only the literal data is sent over the network, avoiding to send data that is already found at the server.

3.2.3 VFC-BOX Main Components

In this section of the document, we describe an overview of the VFC-BOX main components, namely client and server node. Client nodes correspond to the application and environment that enables clients to asynchronously edit files, while guaranteeing their efficient synchronization and storage. This component has also the duty of informing the server about the current interests that users have over parts of the shared data.

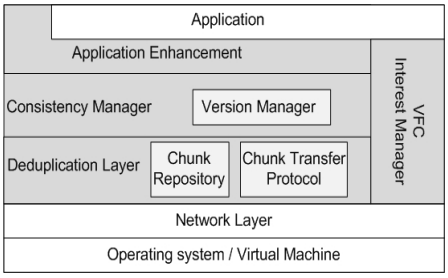

Figure 9: Client Architecture.

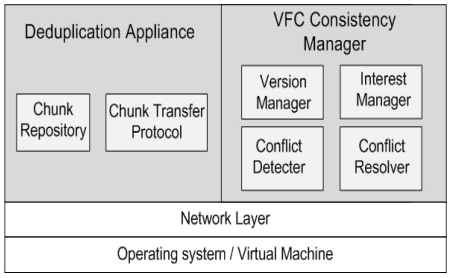

Figure 10: Server Architecture.

We propose an architecture where client nodes are composed by 4 main layers: *Application Enhancement*, *VFC Interest Manager*, *Consistency Manager* and *Deduplication Layer*. On the following we describe each layer in more detail:

- **Application Enhancement**: This layer is responsible for keeping track of any document update performed by the application. It is also responsible to provide support to the application in what concerns to conflict resolution.

- **VFC Interest Manager**: This layer is in charge of collapsing user's interests over data parts and sending them to the server.
- **Consistency Manager**: This layer is responsible to receive and apply incoming updates and to propagate outgoing updates to the server.
- **Deduplication Layer**: This layer performs deduplication operations, either for local storage proposes or for transfer proposes.

The Server node corresponds to the central node of the system. Every updates are either received or sent by the server, being this node responsible to ensure data consistency through client nodes. This component is thus responsible for receiving updates and propagate them according to users interests. Additionally, this component also performs deduplication operations in order to remove redundancy either from data to be stored or transferred.

We propose an architecture where server nodes are composed by 2 main layers: *Deduplication Appliance* and *VFC Consistency Manager*. On the following we describe each layer in more detail:

- **Deduplication Appliance**: This layer is responsible of performing deduplication operations, either for local storage propose or for transfer proposes.
- **VFC Consistency Manager**: This layer is responsible to enforce the VFC model over the synchronization process. Thus, it manages all users interests in order to perform the propagation of updates in a selectively way. This layer has also the task of managing the history of file versions in order to detect and resolve existing conflicts.

4 Methodology and Evaluation

To evaluate the system, four dedicated machines are required. Two of these machines will be used to simulate two different VFC-BOX clients that want to synchronize several data files. Another machine will be used for the VFC-BOX server, which will perform synchronization operations and conflict detection between clients. Finally, one machine will be used as a bandwith regulator, which will simulate different network environments by delaying network packets.

For the evaluation we aim to perform a benchmark to evaluate the performance of deduplication operations and some mini-benchmarks to evaluate the performance of VFC regarding bandwith usage reduction.

These benchmarks will take into account different workloads:

- **emacs-tar**. This workload is comprised of ten snapshots of emacs[9], one from every year from 1999 to 2009, each one packed in one tar file. This workload is 80% redundant.

[9] obtained from http://www.gnu.org/software/emacs/

- **linux kernel**. This workload is composed by several versions of the linux kernel[10] containing high percentages of *cross-version redundancy*.
- **personal workloads**. This workload is composed by several files from Dropbox users. This may simulate the usual workload of the Dropbox system.

To have a reference point for the evaluation of the performance of VFC-BOX, the same environment and testing procedure will be applied to to a set of solutions from the state-of-the-art. The selected systems for this testing procedure are:

- Subversion (SVN), to compare VFC-BOX with a system that performs Delta-encoding deduplication;
- ShiftBack, to compare VFC-BOX with a system that performs Version-Based deduplication;
- VFC for Cooperative Work, to compare VFC-BOX with a system that performs data synchronization based on VFC;
- Dropbox, to compare VFC-BOX with a multi-user synchronizing system.

Three main measures will be taken to evaluate the system, namely Memory Usage, Bandwith Usage and Time. The system must be efficient in terms of memory usage both at the client and server side, in terms of bandwith usage (reducing the amount of transferred data) and in terms of performance, not creating a substantial overhead in which regards to the throughput of the system.

5 Conclusion

Computer assisted collaborative work has been motivating the design and implementation of systems capable of efficiently sharing and storing data. Furthermore, the interest of collaborators over parts of the shared information improve the ability of systems to perform data synchronization while reducing the use of network resources and shrinking the latency of data seen by users. This document provides an overview of the current state-of-the-art technologies to efficiently share and store data.

Relevant systems were presented and discussed, indicating their main advantages and disadvantages. Finally, this document presents an overview of the preliminary VFC-BOX's architecture. VFC-BOX is a multi-user consistent file sharing system based on deduplication techniques and a replication model that takes into account the interest management (or locality-awareness) of a user. Through these techniques, this system may achieve higher performances w.r.t. data transfer/synchronization and storage. We expect this solution to overcome

[10] obtained from http://caixamagica.pt

some limitations of current solutions, improving the capacity of users to receive important data in a faster way through the support of an interest management interface. Additionally, we expect to achieve higher performances than current solutions, due to low-overhead and more efficient data transfer protocols.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, and R. Katz. A view of cloud computing. *In Magazine Communications of the ACM*, Volume 53 Issue 4:50–58, 2010.

[2] R. Balan, M. Ebling, P. Castro, and A. Misra. Matrix: Adaptive middleware for distributed multiplayer games. *In Middleware '05: ACM/IFIP/USENIX 6th International Middleware Conference*, Volume 3790:390–400, 2005.

[3] S. Balasubramaniam and B. Pierce. What is a file synchronizer. *In MobiCom '98 Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, 1998.

[4] J. Barreto. Optimistic replication in weakly connected resource-constrained environments. *PhD Thesis, Instituto Superior Técnico*, 2008.

[5] J. Barreto and P. Ferreira. A replicated file system for resource constrained mobile devices. *In Proceedings of IADIS International Conference on Applied Computing*, 2004.

[6] J. Barreto and P. Ferreira. A highly available replicated file system for resource-constrained windows ce .net devices. *In 3rd International Conference on .NET Technologies*, 2005.

[7] J. Barreto and P. Ferreira. Efficient locally trackable deduplication in replicated systems. *In Middleware'09: Proceedings of the ACM/IFIP/USENIX 10th international conference on Middleware*, 2009.

[8] W. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in windows 2000. *In WSS'00 Proceedings of the 4th conference on USENIX Windows Systems Symposium*, Volume 4, 2000.

[9] M. Cart and J. Ferrie. Asynchronous reconciliation based on operational transformation for p2p collaborative environments. *In COLCOM '07: Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 127–138, 2007.

[10] P. Cederqvist and et al. Version management with cvs. http://www.cvshome.org/docs/manual/. 1993.

[11] J. Costa, L. Veiga, and P. Ferreira. Vector-field consistency for cooperative work. *Msc Thesis, Instituto Superior Técnico*, 2010.

[12] L. Cox, C. Murray, and B. Noble. Pastiche: Making backup cheap and easy. *In OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 285–298, 2002.

[13] F. Douglis and A. Iyengar. Application-specific delta encoding via resemblance detection. *In Proceedings of the 2003 USENIX Annual Technical Conference*, pages 113–126, 2003.

[14] D.E. Eastlake and P.E. Jones. Us secure hash algorithm 1 (sha1). *http://www.ietf.org/rfc/rfc3174.txt?number=3174*, 2001.

[15] Collins-Sussman et al. Version control with subversion. *O'Reilly*, 2004.

[16] K. Morse et al. Interest management in large-scale distributed simulations. *Information and Computer Science, University of California, Irvine*, 1996.

[17] I. Greif. Computer-supported cooperative work: a book of readings. *Morgan Kaufmann Publishers Inc., San Francisco, CA, USA*, 1988.

[18] S. Horwitz, J. Prins, and T. Reps. Integrating noninterfering versions of programs. *In Journal ACM Transactions on Programming Languages and Systems (TOPLAS)*, Volume 11 Issue 3:345–387, 1989.

[19] J. J. Hunt, K.-P. Vo, and W. F. Tichy. An empirical study of delta algorithms. *In ICSE '96: Proceedings of the SCM-6 Workshop on System Configuration Management*, pages 49–66, 1996.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *In Magazine Communications of the ACM*, Volume 21 Issue 7, 1978.

[21] Y. Lu, Y. Lu, and H. Jiang. Adaptive consistency guarantees for large-scale replicated services. *In NAS '08 Proceedings of the 2008 International Conference on Networking, Architecture, and Storage*, 2008.

[22] N. Mandagere, P. Zhou, M. Smith, and S. Uttamchandani. Demystifying data deduplication. *In Companion '08 Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, 2008.

[23] F. Mattern. Virtual time and global states of distributed systems. *In Parallel and Distributed Algorithms: proceedings of the International Workshop on Parallel and Distributed Algorithms*, 1989.

[24] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. *In SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, Volume 35 Issue 4:174–187, 2001.

[25] J. Paiva and P. Ferreira. Backupchunk: A chunk-based backup system. *Msc Thesis, Instituto Superior Técnico*, 2009.

[26] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon s3 for science grids: a viable solution? *In DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, 2008.

[27] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. The bayou architecture: Support for data sharing among mobile users. *In WMCSA '94: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, pages 2–7, 1994.

[28] B. Pierce and J. Vouillon. Whats in unison? a formal specification and reference implementation of a file synchronizer. *Technical Report MS-CIS-03-36, Department of Computer and Information Science University of Pennsylvania*, 2004.

[29] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. *In First USENIX conference on File and Storage Technologies, Monterey,CA*, 2002.

[30] M. Rabin. Fingerprinting by random polynomials. *Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University*, 1981.

[31] D. Ratner. Roam: A scalable replication system for mobile and distributed computing. *PhD Thesis 970044, University of California*, 1998.

[32] R.L. Rivest. The md5 message-digest algorithm (rfc 1321). *http://www.ietf.org/rfc/rfc1321.txt?number=1321*, 1992.

[33] Y. Saito and M. Shapiro. Optimistic replication. *In Journal ACM Computing Surveys (CSUR)*, Volume 37 Issue 1(1):42–81, 2005.

[34] N. Santos, L. Veiga, and P. Ferreira. Vector-field consistency for ad-hoc gaming. *INESC-ID/Technical University of Lisbon, Distributed Systems Group Rua Alves Redol N 9, 1000-029 Lisboa.*

[35] M. Satyanarayanan. The evolution of coda. *In Journal ACM Transactions on Computer Systems (TOCS)*, Volume 20 Issue 2:85–124, 2002.

[36] H. Seunghyun, L. Mingyu, and Dongman L. Scalable interest management using interest group based filtering for large networked virtual environments. *In VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology*, 2000.

[37] A. Shiferaw, V. Scuturici, and L. Brunie. Interest-awareness for information sharing in manets. *In MDM '10: Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, 2010.

[38] W. F. Tichy. Rcs - a system for version control. *Department of Computer Sciences Purdue University West Lafayette, Indiana 47907*, 1991.

[39] A. Tridgell and P. Mackerras. The rsync algorithm. *Australian National University*, 1998.

[40] P. Vala and P. Ferreira. Shiftback: Eficient and time-shifting backup. *Msc Thesis, Instituto Superior Técnico*, 2010.

[41] L. Veiga and P. Ferreira. Semantic-chunks: A middleware for ubiquitous cooperative work. *In ARM '05 Proceedings of the 4th workshop on Reflective and Adaptive Middleware Systems*, 2005.

[42] L. Veiga, A. Negrão, N. Santos, and P. Ferreira. Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency. *INESC-ID/Technical University of Lisbon, Distributed Systems Group Rua Alves Redol N 9, 1000-029 Lisboa*, 2009.

[43] W. Vogels. Eventually consistent. *In Magazine Communications of the ACM Communications of the ACM - Rural engineering development CACM*, Volume 52 Issue 1:40–44, 2009.

[44] A.-I. Wang, P. L. Reiher, and R. Bagrodia. Understanding the conflict rate metric for peer optimistically replicated filing environments. *In DEXA '02: Proceedings of the 13th International Workshop on Database and Expert Systems Applications*, 2002.

[45] M. Weiser. The computer for the twenty-first century. *In ACM SIGMOBILE Mobile Computing and Communications Review*, Volume 3 Issue 3, 1991.

[46] H. Yu and A. Vahdat. Design and evaluation of a confict-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems (TOCS)*, Volume 20 Issue 3:239–282, 2002.

[47] H. Yu and A. Vahdat. The costs and limits of availability for replicated services. *In Journal ACM Transactions on Computer Systems (TOCS)*, Volume 24 Issue 1:70–113, 2006.

Appendix: Work Scheduling

The following table presents the proposed schedule for the Dissertation Course.

Activity	**Date start**(dd/mm/yy)	**Date end**(dd/mm/yy)
Improve architecture design	10/01/11	05/02/11
Selection of tools	05/02/11	25/02/11
Implementation	25/02/11	30/04/11
Evaluation	01/05/11	31/05/11
Evaluate other systems	01/05/11	20/05/11
Evaluate VFC-BOX	20/05/11	31/05/11
Conclusion of Thesis and Article writing	01/06/11	30/06/11
Thesis Review	30/06/11	30/07/11
Thesis Delivery	30/07/11	