The Nottingham Trent University

Department of Computing and Mathematics

Cross Platform Download Delta Updater

Paul Phillips

2003

Project Report in part or fulfilment

of the requirements for the degree of

Bachelor of Science with Honours

In

Computing (Visualisation)

Abstract

Acknowledgements

Table of Contents

List of Figures and Tables

1.0 Introduction

The growth and speed at which new software and other documents are created and modified has escalated beyond all belief in recent years and the need for up to date information and files has also risen in importance. Companies and individuals need the latest patches for their systems to maintain the security of their systems and data. Developers and users download the latest versions of source code from projects regularly to modify and compile at home. All of these tasks involve large amounts of bandwidth on the computer networks involved and thus a price, be it time, personnel performance or financial in nature. This coupled with the general rising size of files and software, results in local area networks and peoples' internet connection bandwidth being stressed for long periods of time as files are copied from one place to another. In the case of copying newly updated files over a network, many data that has already been downloaded, in the original file, is needlessly downloaded again.

This project aims to reduce the bandwidth used in the copying of new versions of files over computer networks by sending only the parts of the files that have changed since the previous version. Therefore a large file that has been changed but only by a few characters would currently have to be sent in its entirety over the network, the proposed system will attempt to only transmit the parts of the file that have changed. These changes can then be applied to the old file, which will result in the creation of new version.

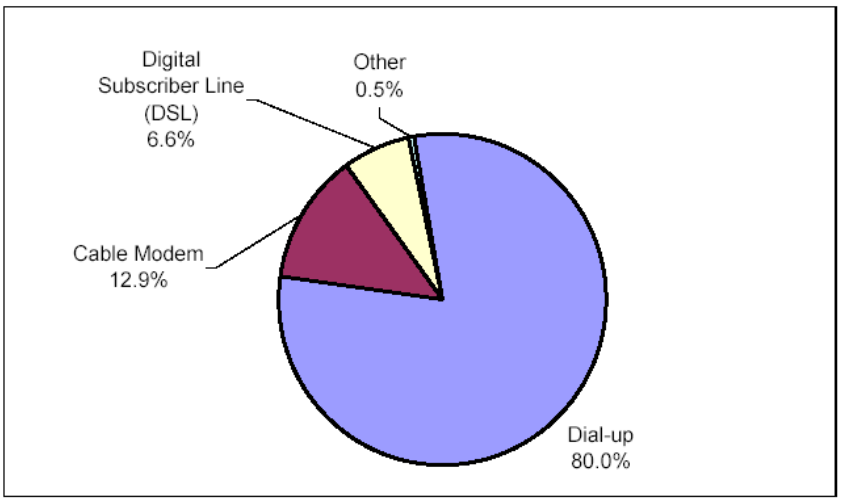
In the end, this could increase productivity of developers or other users of the system by decreasing the time they spend waiting for files to download, over a period of time

this could mount up depending on the size and quantity of files involved in their work. For example, a developer who is performing a bringover of a full source tree from a remote site could have to wait whilst gigabytes of information are transferred. This system could dramatically reduce the amount of data needed that needs to be transferred by only copying over files that have changed since their last bringover, potentially saving megabytes or hours. The majority of users of the internet connect at relatively low speeds, usually between 28 and 56 kilobits a second, this makes moderate to large downloads very tiresome and best avoided unless essential. Attempting to surf the web *and* download an important file will almost certainly suck up all available bandwidth and leave users heading for the kettle whilst they wait. A system that would allow these users only to download the parts of files that they needed would mean only one potentially large initial download and then successive smaller downloads to acquire and apply the updates to the old files.

This could also have applications in other areas of computing where updates are required. For instance, the synchronising of websites could be made simpler and faster by only copying over the parts of files that have changed, or the downloading of software updates through a web portal could be made faster.

1.1 History

It has been apparent for many years that most users of the internet have not been well catered for. The majority of users still access the internet on old slow dial up modems at 14-56k lines, whereas many websites are designed for much faster connections up to 10Mbps LAN connections. Websites often have large graphics to download that take up large amounts of bandwidth whilst adding little to the browsing experience, especially on sites that have banner advertisements. These advertisements are not going away anytime soon because they provide a healthy income for the sites that host them. Many users are turning to solutions that stop banner adverts being shown, either to reduce bandwidth requirements or to stop the annoyance of them. In 1996 Fox and Brewer reported that '[a] study by a popular server of shareware, Jumbo, revealed that about 1 in 5 users were connecting with graphics turned off, to eliminate the annoying latency of loading web pages.' According to a report published in 2002 by classicbranding.com, using information gathered from the NTIA and ESA, US Department of Commerce, using US Census Bureau Current Population Survey Supplements; 80% of US households that use the internet in 2001 accessed the internet using a dialup connection. A mere 18.5% of US household that had an internet connection had a broadband service such as DSL or Cable.



Source: NTIA and ESA, U.S. Department of Commerce, using U.S. Census Bureau Current Population Survey Supplements

Figure 1.0: Distribution of internet connection types used by US households in 2001.

There have been studies on how best to reduce the problem of slow internet connections and the desire for large images in web pages that will be discussed in more detail in the next chapter.

In 1995, a group of loosely knit web developers formed the Bandwidth Conservation Society (BCS); with the intent of teaching people how best to reduce the byte count of images through proper use of the compression algorithms images use. The use of lossy algorithms that reduce image file sizes has been essential to the development of the internet where most users have slow connections. Compressed images allowed images to be used in situations where the use of inherently large bitmap images is infeasible (due to slow connection speeds), the compressed image has allowed for the commercialisation of the interne. The aim of BCS was to promote better use of image compression algorithms and make the web browsing experience a more pleasurable experience to users. They provided tutorials on good image compression for users to

follow and then use in their web pages. It was argued that by reducing the byte count of the images used,

'1. The developer uses less hard disk real estate.

2. The developer uses less cpu overhead to deliver the image.

*3. **The user gets the image quicker!** This is the important one.*

4. Fewer bytes across the net means more room as the popularity of the net grows.

*5. **Courtesy**; As more folks are sharing web-server resources, your electronic neighbo[u]rs will appreciate your intelligence and web-savvy delivery.'*

Back in 1995, these were big problems with the internet and the creation of web pages by amateurs, and big business alike that often tested their websites on the company LAN that runs hundreds of times faster than the connections people use at home.

Although the BCS had good intentions at reducing bandwidth, there has been much research into other forms of compression in order to reduce bandwidth use. Files are often compressed using algorithms such as zip and bzip2 but through using a delta compression algorithm it should be possible to achieve better compression when a user already has a previous version of the file(s). Hunt, Vo and Tichy (1998) describe delta algorithms as 'compress[ing] data by encoding one file in terms of another'; this is discussed in more detail in section 2.2.3. To put this technique into context, Suel and Memon (2002), proposed a number of different applications for delta compression. The one that pioneered the development of delta compression was the use of software revision-control systems that maintain the revision history of software projects and other documents. There has also been work on file system level delta systems such as XDFS created by MacDonald in his 2000 thesis, 'File system support

for delta compression.’ There are also uses for software distribution (see chapter 2.2.8), exploring file differences, improving HTTP performance through using delta compression on html files that are similar, and efficient web page storage.

This project hopes to expand upon and create a usable delta compression system that can be used by novice users for the distribution of their software and documentation over the internet.

1.2 Literature Search

This project is primarily focussed on maximising the efficiency of downloads for users of the internet with low speed dial up connections by eliminating the need to download data the user already has. A study of bandwidth maximising strategies, studies and products is therefore required. Below is a list of the sources of information used in gathering this information and the search terms and engines used in locating the information.

Bandwidth

Search phrase: Bandwidth Conservation Society

Search engine: http://google.co.uk

Results:

- http://www.google.co.uk/search?q=bandwidth+conservation+society&btnG=Google+Search&num=50&hl=en&ie=UTF-8&oe=UTF-8&safe=off

Search phrase: bandwidth

Search Engine: http://google.co.uk & http://citeseer.ist.psu.edu

Results:

- http://www.google.co.uk/search?q=bandwidth&ie=UTF-8&oe=UTF-8&hl=en&btnG=Google+Search
- http://www.google.com/search?q=site:citeseer.nj.nec.com+bandwidth&btnG=Google+Search
- http://citeseer.ist.psu.edu/cis?q=bandwidth&submit=Search+Documents&cs=1

Search phrase: bandwidth optimization

Search Engine: http://google.co.uk & http://citeseer.ist.psu.edu

Results:

- http://www.google.co.uk/search?num=50&hl=en&ie=UTF-8&oe=UTF-8&safe=off&q=bandwidth+optimization&spell=1
- http://citeseer.ist.psu.edu/cis?cs=1&q=bandwidth+optimisation&submit=Documents&co=Citations&cm=50&cf=Any&ao=Citations&am=20&af=Any
- http://www.google.com/search?hl=en&lr=&ie=ISO-8859-1&q=site%3Aciteseer.nj.nec.com+bandwidth+optimisation&btnG=Google+Search

Download Managers

Search Phrase: "download managers"

Search Engine: http://google.co.uk & http://uk.search.yahoo.com

Results:

- http://www.google.co.uk/search?q=%22download+managers%22&ie=UTF-8&oe=UTF-8&hl=en&btnG=Google+Search
- http://uk.search.yahoo.com/search/ukie?fr=fp-top&p=%22download+managers%22&y=y

Data Compression

Search Phrase: "delta compression"

Search Engine: http://google.co.uk & http://citeseer.ist.psu.edu

Results:

- http://citeseer.ist.psu.edu/cis?cs=1&q=delta+compression&submit=Documents&co=Citations&cm=50&cf=Any&ao=Citations&am=20&af=Any
- http://www.google.co.uk/search?q=%22delta+compression%22&ie=UTF-8&oe=UTF-8&hl=en&btnG=Google+Search

1.3 Report Overview

1.3.1 Limitations

In this chapter, the research and products available in the same and similar areas to this project will be compared and contrasted with a view to identifying their limitations and weaknesses. These cover a wide range of products from download managers, file sharing applications, FTP clients and tools for enabling easy uniform installation of operating systems. All these products deal with functionality that involves the copying of files and the use of bandwidth, either over a network or possibly bandwidth internal to a computer. As such, research into them is necessary to identify systems that do the same or similar things and to find where these systems let the user down and where this project might compliment these other programs and provide new ideas and functionality is discussed in the next chapter.

1.3.2 New Ideas

Following on from the limitations discussed above this chapter will set out a new area that this project will address. These ideas will form part of a problem description and requirements specification for a new piece of software to be developed. This chapter also covers the project breakdown, the tasks that have to be achieved, timelines and contingency plans.

1.3.3 Software Development

Once the project has been detailed and an outline of a specification drawn up, this chapter will refine the requirements and discuss the process of the development. The chapter will cover the design of the system and the reasons behind design decisions, to the coding and the progression through the software development process.

1.3.4 Results

Once the development is complete, the system will be tested thoroughly using a range of testing techniques and metrics, including black and white box testing and analysis against the requirements specification.

1.3.5 Conclusions and Future Work

The results of testing will show where the project succeeded and did not; this chapter covers the reasons for these failures and successes and suggests future work in similar areas where this project can be developed further.

2.0 Limitations

2.1 Literature Review

As far as it was known to the author, no such system like this has been developed before and thus a detailed search was required to find systems that did either the same or similar things to those that are being proposed. The subject areas that are of interest to this project were discussed and identified in section 1.1.1 above, namely network protocols, bandwidth, quality of service, bandwidth optimisation, download managers and typical services offered by web hosts and ISPs. During the search, 'delta compression' was also identified as an area of interest that was previously unknown to the author, this is discussed in section 2.2.3.

2.1.1 Distillation and Refinement

Fox and Brewer (1996) discussed a dynamic method for reducing the bandwidth images required. Their approach was a real time lossy compression algorithm that would run on the web server and be user configurable. The system could reduce the physical dimensions of an image and its pixel depth both of which affect the file size before being sent to the user. They called this process *distillation.* Their argument was that, 'the resulting representation, though poorer in colo[u]r and resolution than the original, is nonetheless still recognizable and therefore useful to the user.' The advantage being that the server does not have to store multiple copies of an image for different user preferences because the compression is done in real time by the server on an original image. In other words, 'any desired intermediate representation can be created on demand using an appropriate distiller.' This technique is in frequent use nowadays by websites that utilise libraries such as the GD library in PHP to create dynamic thumbnail images. This does go against one of the principles of the BCS

stated above (see chapter 1.1), that of reducing CPU cycles on the server, it is nonetheless effective at bandwidth reduction and website complexity reduction. Fox and Brewer also discussed a process they called *refinement*, that is a user could request a *part* of an image at increased quality, perhaps a page of a large PDF document or an area of a GIF image. This allows for user selectable bandwidth saving by not download data irrelevant to the user. The example Fox and Brewer use in their report shows an image of a building that has been distilled from 503KB down to 17KB whilst still clearly being of the original image. However, the writing on the building is illegible, so an area is refined to show the area from the original image that shows the text clearly. This part of the original image that was downloaded used a mere 15KB of bandwidth. Altogether the user got the information they needed but used only 32KB (17KB for the full distilled image and 15KB for the area that was refined) of bandwidth as opposed to 503KB for the full image. This is a reduction of the required bandwidth by almost sixteen times.

2.2.2 Web Prefetching

Web Prefetching was developed to reduce the latency experienced by internet users with slow connections, by prefetching between caching proxies and browsers. The technique uses a proxy to predict what the user will download next and use the time in which the user's connection is idle to download those documents to the user's browser. Idle connection use is often found after a user has downloaded a webpage and stops using the connection to read the page. Fan, Cao and Jacobson reported in 1999 that 'prefetching combined with a large browser cache and delta-compression can decrease user-perceived latency up 23.4%.' The results were for an algorithm called Prediction-by-Partial-Matching (PPM), 'whose accuracy in prediction ranged

from 40% to 73% depending on its parameters, and which generated 1% to 15% extra traffic on modem links.' This does not reduce bandwidth at all but it does make the latency experienced by user smaller and is therefore a generally good thing. It is therefore wrong to assume that decreasing the use of bandwidth is always good for the user, especially if the cost is increased waiting times.

2.2.3 Delta Compression

Whilst conducting this literature search and review the area of delta compression was discovered. This is the technique and area that this project aimed to address.

It is suggested by Suel and Memon (2002) that 'in today's network-based environment it is often the case that files and content are widely replicated, frequently modified, and cut and reassembled in different contexts and packagings.' Thus to try to reduce unnecessary bandwidth use they suggest the use of delta compression and give an example of a good use for such a system:

'Consider the case of a server distributing a software package. If the client already has an older version of the software, then an efficient distribution scheme would only send a patch to the client that describes the differences between the old and the new version. In particular, the client would send a request to the server that specifies the version number of the outdated version at the client. The server then looks at the new version of the software, and at the outdated version that we assume is available to the server, and computes and sends out a "patch" that the client can use to update its version. The process of computing such a "patch" of minimal size between two files is called ***delta compression****, or sometimes also* ***delta encoding*** *or* ***differential compression****.'*

Delta compression has bee around for some time and started life in database technology as a method of backing up only the files that have changed since the last backup. This is however pretty coarse granularity and techniques have since been developed that allow not only just the files that have changed to be backed up, but also just the parts of the files that have changed to be backed up. In database systems, a logging algorithm is typically used that records all the writes made to the file, this can then be used to recover a file from a previous state to its current state. As Burns and Long (1997) point out, 'while semantically similar to delta compression, logging does not provide the compressibility guarantees of differencing algorithms. In the database example, a log or journal of changes grows in proportion to the number of writes made.' This technique is clearly not what is required for a system that aims to reduce bandwidth, indeed Burns and Long go onto suggest that on a busy database page, 'the log would likely exceed page size.'

Another early use of delta compression was in revision control systems such as RCS and SCCS, Hunt, Vo and Tichy (1998) state that the classic program for creating delta files was the UNIX *diff* command that both SCCS and RCS used. This works well for text-based files such as source code, but when used with binary files the deltas could often be larger then the new file itself. They go on to say that new tools are available such as *bdiff* and *suff* that do not have this problem. They argue that 'binary differencing capability has become mandatory,' due to the number of files that are binary these days, from word processor documents and presentations, spreadsheet data, CAD data, sound and video etc. It is therefore necessary that revision control systems handle binary data in an efficient way.

A by-product of using a delta algorithm that Hunt, Vo and Tichy mention is one of encryption, 'a delta is effectively an encrypted form of the new version. It can only be decoded if the original is available.' They suggest that this could 'strongly reduce the threat of piracy' when distributing software updates, this is very much dependent however, on how difficult it is to get hold of an original version and the delta. An original version with no delta is just out of date but it will still work, a delta with no original is useless, and therefore if someone can get hold of both, the delta can be applied. This is analogous to suggesting that because software updates are useless on their own with nothing to update, that this will reduce piracy, when piracy comes from the distribution of full software, not the updates for original software (although this may also be pirated).

Suel and Memon (2002) compare three different delta algorithms, *vcdiff*, *xdelta* and *zdelta* against a reference of the popular compression program *gzip*. A graph they produced of their results can be seen below (figure 2.0), it shows that the *zdelta* algorithm is the better algorithm by compressing better over a wide range of file similarities. Suel and Memon state that 'we see that *vcdiff* and *zdelta* give benefits even for only slightly similar files for which *xdelta* does not improve over *gzip*.'

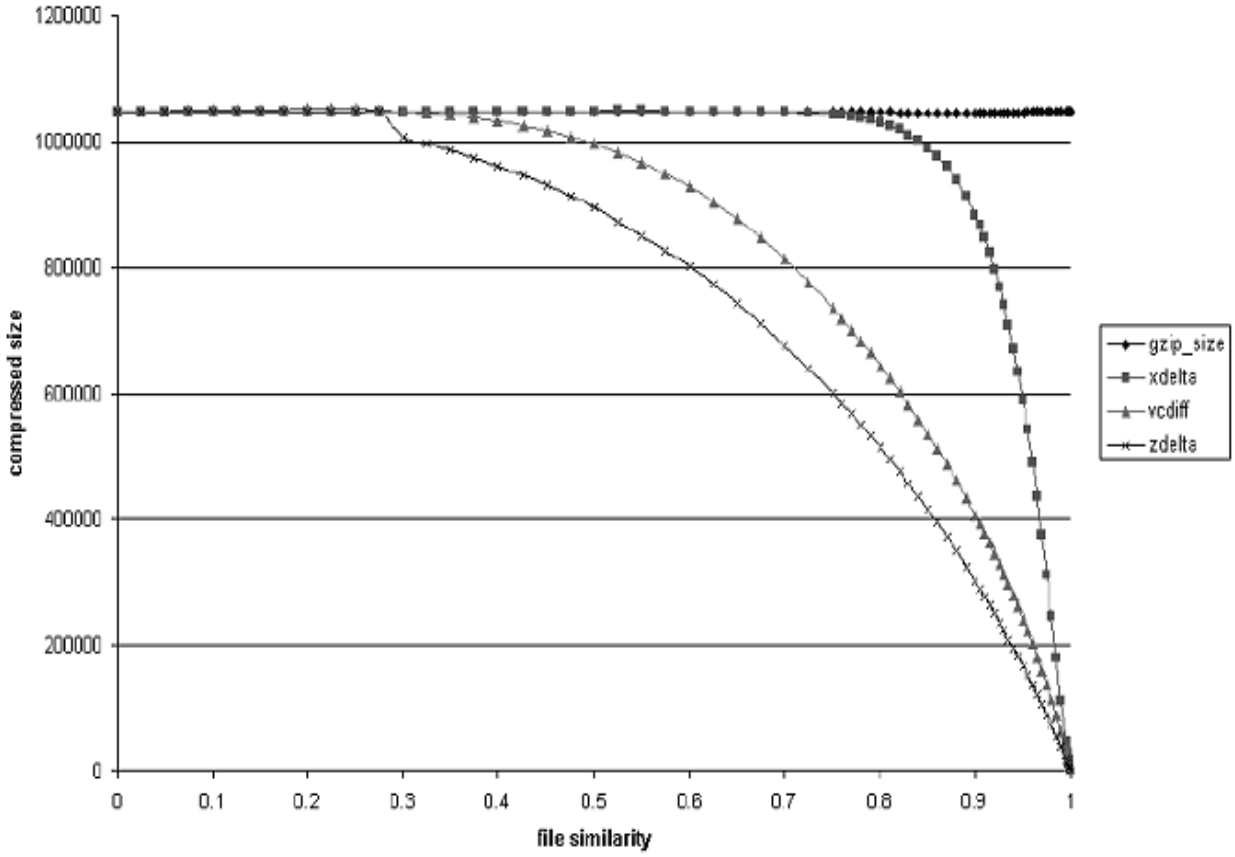


Figure 2.0: Compression versus file similarity

Another type of algorithm is *rsync* developed by Tridgell and MacKerras. This is a slightly different algorithm in that it is used to synchronise data between clients and servers where a reference file is unavailable. This changes the situation, in that data has to be transmitted over the network to compare files, synchronisation techniques tend therefore to be worse than delta compression algorithms for bandwidth use.

The *rsync* algorithm uses checksums to compare blocks of data between two files, the idea being that a checksum provides a way of determining if data are the same. The algorithm compares blocks of data firstly with an unreliable 32bit 'rolling checksum' and then if they match it is verified they are the same with a stronger MD4/5

checksum (128bit). The 32bit “rolling checksum” is used because it is fast to compute and provides evidence, although not proof of, data being the same. According to RFC 1321 (1992), the MD5 checksum ‘produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest.’ This therefore provides proof that the blocks of data are indeed the same. The MD5 checksum is however more computationally expensive and thus is only performed when the quick “rolling checksum” indicates a block might be the same.

The algorithm proceeds by splitting the data into two blocks and compares recursively on blocks that differ until the exact location of the change is located. Suel and Memon acknowledge that this approach does not cope well when a single word is added at the start of the data because it ‘destroy[s] the alignments of all the block boundaries between the two [data].’ They then propose a more refined version of this algorithm that over comes this problem, the results can then be compressed using any compression algorithm such as *gzip*. The sizes of the blocks of data that are compared are important because different block sizes will produce different results depending on how similar data are, so a tailoring of the block sizes can be done manually to achieve the best compression.

The results of this algorithm are not as good as with other methods such as *zdelta* by a factor of three or four, however, this does include data that is transmitted over the network just to compare the files and update them, *not* just the resulting delta data needed to create the new file from the old.

2.2.4 Network Protocols

There are many different network protocols in use today and each has it own different uses and properties that make it useful. There three that of most interest to this project are the user datagram protocol (UDP), internet protocol (IP) and transmission control protocol TCP. TCP and IP are most usually used in conjunction with each other and known as TCP/IP, this binding has some very useful properties as shall be discussed. Each of these protocols are in common use on the internet for varying purposes and all are available to modern network cards and network operating systems, such as UNIX or Microsoft Windows.

In the most common model of network computing a seven-layer system is used and each protocol works at a given level, from the physical layer that sends the individual electrical signals along the network cable to the application on top that initiates action in the layers below. This model is called the Open Systems Interconnection Reference Model (OSI Model or OSI Reference Model for short), and was developed by the Open Systems Interconnect initiative, part of the International Standards Organisation. A table of the different layers and protocols that run on them is shown below.

Layer	Examples
7 – Application	HTTP, SMTP, SNMP, FTP, Telnet, FTAM, APPC, X.400, X.500, NCP, Appletalk, AFP, DAP
6 – Presentation	TDI, XDR, SNMP, FTP, Telnet, SMTP, NCP, AFP
5 – Session	NWLink, NBT, Named Pipes, NetBIOS, ASP, ADSP, ZIP, PAP, DLC
4 – Transport	TCP, UDP, SPX, NetBEUI, ATP, NBP, AEP, RTMP
3 – Network	IP, IPX, NWLink, NetBEUI, DDP
2 – Data link	Ethernet, Token Ring, PPP, ODI, NDIS, LocalTalk, TokenTalk, EtherTalk
1 - Physical	RS-232, ISDN, 10BASE-T, electricity, radio, fibre optics

Figure 2.1: OSI Reference Model

IP

The Internet Protocol runs at level three of the OSI model, source and destination hosts use it to communicate by passing packets of data to each other. Packets are simply parts of larger sets of data; the large data that is to be sent is broken into a series of IP packets that are sent separately and recombined at the destination. The protocol provides unreliable service to the packets that may be received out of order, more than once, corrupt or not at all. The headers of packets contain various pieces of information about the packet, but crucially, it contains a 32-bit source and destination address of the packet. This data is used by routers to deliver the message (or not as the case may be). Currently these 32-bit addresses, used in IP version 4, are being phased out in favour of a new version; IPv6, that uses 128-bit addresses and is designed to fix the problem of running out of available IP addresses on the internet. Because of unreliability and potential for errors, IP is usually used in conjunction with TCP, where it is the most popular protocol on the internet.

UDP

UDP is a minimal message passing system that runs at the transport layer and provides and interface between that application layer and the network layer. Only a

checksum and application multiplexing are added to an IP packet to get an UDP packet. It has several properties that make it unsuitable for many applications but which play to its advantage under certain circumstances. When using this protocol there is no guarantee that messages will arrive at their destination and no way of the sender knowing if the message got there or not. Reliability is not part of this protocol, although some applications such as TFTP may add their own reliability wrappers to the protocol. UDP is useful in situations that are designed to cope with a high rate of errors or where errors are not a huge problem. Real-time games and streaming audio / video are typical uses. Another property lacking from the protocol is congestion avoidance and control mechanisms, that is, it does not know when the network is busy and so will send packets regardless of the state of the network. This means that in a strained networking environment with switches working near their maximum UDP packets will still be sent out, even though the likelihood of them reaching their destination is smaller than most uses require. Having said this there are many applications that use UDP packets that most internet users to use, DNS, SMTP and DHCP all use the UDP protocol.

TCP

The online encyclopaedia, Wikipedia (http://en.wikipedia.org), describes TCP as 'a connection-oriented, reliable delivery byte-stream transport layer protocol.' It sits between the application and network layers and provides connections that stream bytes between hosts. The data is sent as an unstructured stream, but it also sends information that provides a robust connection and presents the stream to the application as free from errors and corruption. This means that an application programmer does not have to worry about corrupted messages or packets being

received out of order because TCP handles these problems. The definition given by Cisco is useful for understanding its other benefits:

'TCP can provide a sending node with delivery information about packets transmitted to a destination node. Where data has been lost in transit from source to destination, TCP can retransmit the data until either a timeout condition is reached or until successful delivery has been achieved. TCP can also recognize duplicate messages and will discard them appropriately. If the sending computer is transmitting too fast for the receiving computer, TCP can employ flow control mechanisms to slow data transfer. TCP can also communicate delivery information to the upper-layer protocols and applications it supports.'

Therefore, if a reliable connection is desirable TCP will provide this at a small overhead in packet size, connection establishment time and timeout time (that is, the time it takes before TCP gives up on a packet and sends it again). For sending large files over a network, this is usually an acceptable trade off because the connection will be used for a period of time. Whereas for sending a one off packet to a computer this trade off might be too great to justify as the sender must wait for a connection to be established.

TCP/IP

The most common protocol on the internet is actually two used together. IP packets are sent through TCP byte streams to provide reliable connections between hosts. This provides the advantages of IP routing and addressing, multicasting (where data is sent to multiple selected hosts), broadcasting etc. with the reliability and benefits of a connection-orientated stream. It is this combination of protocols that applications such as HTTP and FTP use for uploading and downloading files to and from websites.

The combination guarantees packet delivery even on saturated networks, this is important when corruption and packets arriving out of sequence is a problem.

2.2.5 Download Managers

For some time now, there have been so called download managers available to users that allow fast, scheduled and robust methods of downloading lots of files or large files. It has been found that there are in fact many such programs each with similar facilities, but non have been found that address the issues this project aims to tackle, as we shall see.

GetRight

GetRight is one of the most popular download managers and has been around for several years, it offers and impressive line up of features and tools. One of its primary uses by most users is that it enables you to resume broken downloads in case of interruption of a connection or a system failure at either end. This is useful if the user has a slow or unreliable connection, or requires downloading a file over several days due to the size of the file. As many people will know it is very annoying to have a download that has been going for several hours or longer to crash or die in the middle and lose all that data. This system eliminates that problem by saving the file as it is downloaded and then all that is required is the resumption of the download from where it left off. It should be noted however that not all web servers allow the retrieval of files from the middle of the file and force downloads always to start at the beginning of files. These systems are rare however, and it is generally not a problem for most users, GetRight will tell the user when file resuming is not enabled.

Problems with errors within files once they have been downloaded are also of concern, especially when an unreliable connection is used, so GetRight provides error detection mechanisms using cyclic redundancy checks, it cannot however recover from such errors.

GetRight allows the scheduling of downloads at specific times and will even automatically dial up using the users modem and disconnect and shutdown their PC when it has finished. Useful if you want to start a download after you have gone out, for example after 7pm when the phone calls are cheaper, and have it turn off the PC to save electricity.

The most interesting features it has to offer from the point of view of this project are its ability to synchronise files on a user's local computer with those on a web server. This is not as advanced as that of *rsync* in that is does not transfer only the data that needs to be transmitted but the whole file that has changed, no matter how minor the change was. In effect, this is an improved utility of the UNIX and Linux utility *wget,* which is used to download files and directories from web sites. There are also Windows GUI based tools that allow the downloading of entire sites, such as QuadSucker from SB Software. Some of these website downloader's however, do not take into consideration the time the files were created and so they needlessly download files that may have been downloaded before in a previous synchronisation. *wget* can be setup to get around this problem and GetRight provides an easy to use solution and will only download changed files. These systems do not however, get around having to download new files in their entirety and thus re synchronising a website with a local copy or a mirror site wastes bandwidth.

2.2.6 FreeBSD Binary Updater Project (binup)

The FreeBSD organisation that develops its own free UNIX distribution started development of a utility that would allow automatic updating of a FreeBSD installation according to installation profiles. The project has not been completed and the page has not been updated for almost a year (December 2002), as such, it is not a finished product and appears out of development, but is still worth discussing here.

The aim of the project was to 'provide a secure mechanism for the distribution of binary updates for FreeBSD ', the interesting feature that this project has is that of 'profiles'. These profiles describe how to install a system with FreeBSD, such that a profile of a computer can be created and then used to install other systems the same. This does not require a disk image to be taken of the original computer but a list of instructions on what is installed is created as a profile. This enables the profile to be used to install the operating system on computers of differing architectures, e.g. on a SPARC or ALPHA machines from the same profile data.

This project uses client-server architecture with the profiles being stored on the upgrade server; this allows for authentication of users, as there may be situations where company policy states that only certain people are allowed to upgrade or install computers.

To upgrade a client's software it first authenticates with the server and sends over its current profile with version numbers of all the packages installed on it. The server

can then search to see if a new version of each package exists ‘Deltas and/or entire collections are sent to the client for unpacking along with any before/after actions for each delta or collection which should be executed on the client.’ The information about packages is installed in a SQL database that is implemented through an abstraction layer, so that the underlying database that is used can be changed without changing the servers interface to the data. For instance, a MySQL or Postgresql could be interchanged but due to the database abstraction layer the server does not know this.

The client is less well developed and is not currently usable as it states on the binup project website. In addition, ‘at the moment, it consists of some code to perform the actual updates, and some quick code to test the various functions of the updater. Also, the client does not currently handle packages.’ This is a far from finished product but the idea was a good and promising one and is almost an extension to the method commonly used for a ‘net install’ on UNIX systems such as Solaris.

That is, a remote machine acting as a boot/install server holds entire operating system images and when the client issues a ‘boot net – install’ command the client finds the boot server and starts running the installation of the operating system the server has assigned to the client’s IP or MAC address. The author expects that the binup system would have complimented the boot install procedure very well because an entire new installation of the new operating system from scratch would not be required to get the latest version on a computer. Instead, the operating system is upgraded for you by this system installing the latest packages using the data in the profile for a particular setup.

This project is interesting because it held a lot of promise for the easy updating of files in a client-server system. However, more than that, it allowed for the execution of commands and programs after they had been copied without major changes, such as the need for an object request broker or Java Remote Method Invocation functionality.

2.2.7 Quality of Service Solutions

NetEnforcer

NetEnforcer is not a software package as the previous two studies are; instead, it is a hardware device that sits between a company's internet router and switch. Its job is to improve the quality of service (QoS) to mission critical uses of the internet connection by slowing down non-mission critical uses. That is, important VPN or teleconferencing connections get a larger share of the bandwidth than internet radio of file sharing applications. The default for most routed internet connections is that the router evenly distributes the bandwidth between users, or worse, on a first come first served basis; this system however, prioritises bandwidth depending on its determined importance. The system allows for the creation of configurable profiles that can be applied on a per user or application basis, these determine what priority each user or application should receive. In times of high internet usage, the system kicks in and starts to limit bandwidth to those users and applications not defined as critical.

The system has a Java GUI front end that not only allows configuration of the systems itself but also monitoring and report generating to show bandwidth usage over a period of time. Statistics on bandwidth usage for users or applications can be shown in graph form that can be used for deciding on what the most appropriate policies to

use are. It will even show what network protocols are running on the network and show you what ones potentially need limiting.

2.2.8 Typical Web Services

As the internet will be the distribution architecture in this project, it is important to know what is available to web masters when creating their sites that can be used within this project. All websites have some space available to store files but some have differing means of uploading files to the site. Many free or cheap hosts only allow access through a web interface that is limited in the number of files that can be manipulated and the operations that can be performed.

The interface that free web host Tripod provides (Figure 2.2) is typical of such web based systems, it offers limited functionality but with the benefit of being easy for novice users. The number of files that can be uploaded in one go is set at under 10 in most cases. Another example can be seen below from Geocities service (Figure 2.3).

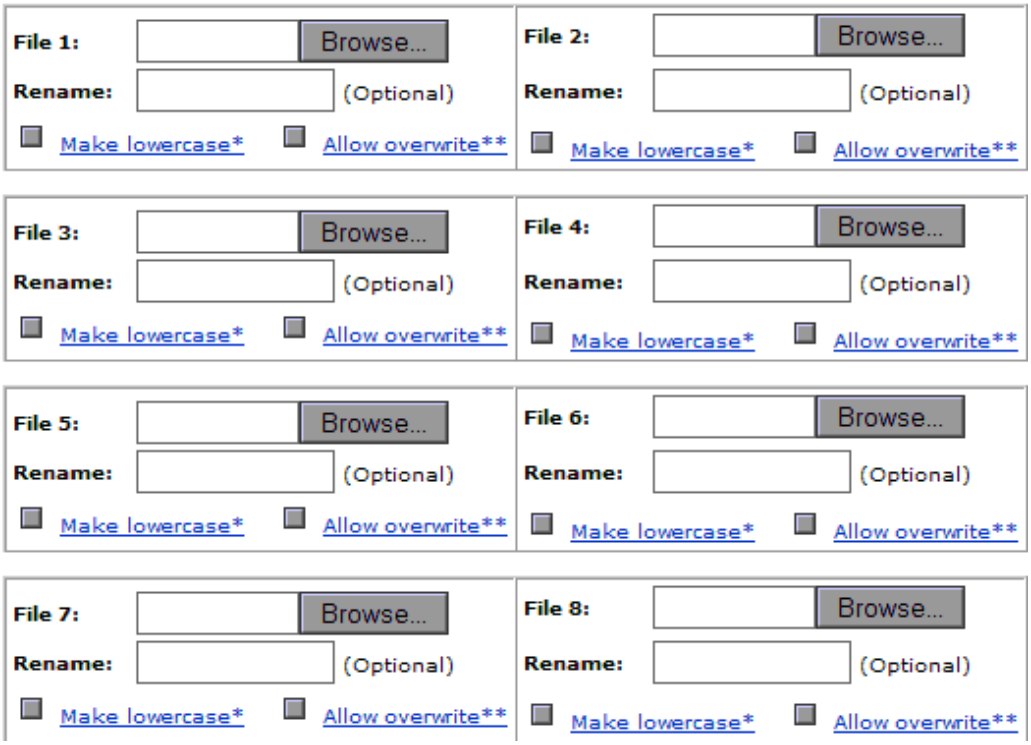


Figure 2.2 Tripod file manager interface

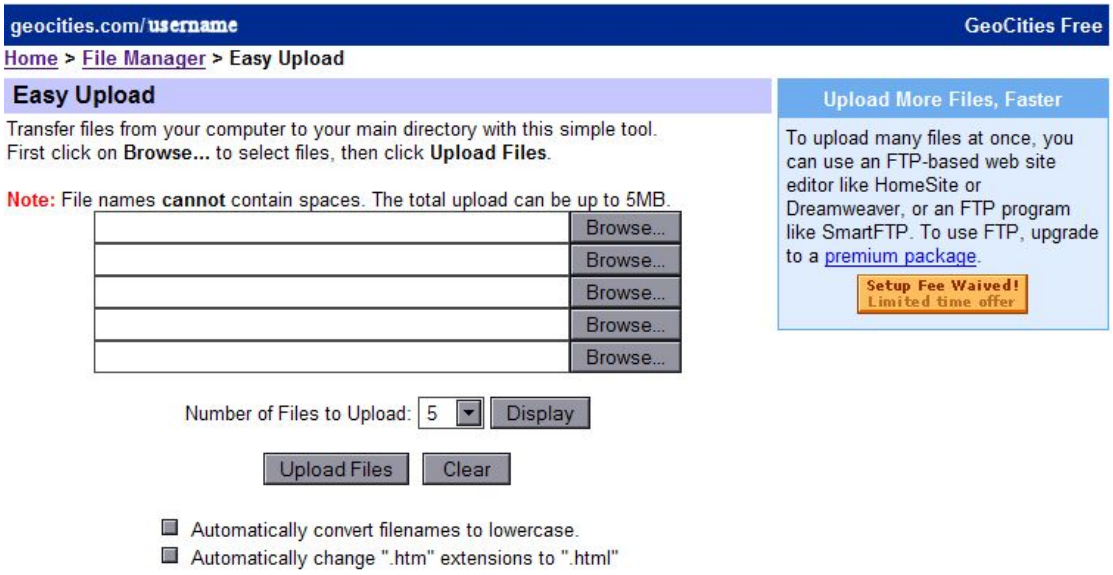


Figure 2.3 Geocities file manager interface

More common for paid for hosting, and some free hosting, is the ability to use FTP to connect to the web server. This provides a lot of functionality that is lacking in web-based systems and can be expanded upon through different FTP clients to offer queuing of transfers, easy ability to change file permissions etc. FTP can be used

through an intuitive explorer interface familiar to almost all users of GUI operating systems, such clients include SmartFTP and FTPvoyager available to Windows users. There are also command line tools that are often used by users of UNIX or Linux systems, for example, *ncftp* is a CLI application that builds some of the functionality of the bash shell into and FTP program making a much more friendly experience than the default ftp program, albeit not as user friendly as a GUI based system.

Many web hosts now offer some form of server side scripting support, be it PHP, Perl, CGI, ASP etc. This allows user created scripts to be run on the server allowing users to create dynamic sites that query data in files and produce real-time content. These are often used to create login in systems, shopping sites, online training web sites etc. They can also be used with a database such as the popular open source database, MySQL (http://www.mysql.com) that many hosts offer to their users. This allows users database functionality on the server, which is ideal for providing dynamic content, and standard API's that plain files alone cannot provide. Permissions are versatile on most databases and MySQL allows flexible access controls from access to a database, its tables and even table columns. That is, a user account can be setup that only has access to specific columns in a table if necessary, allowing different users access to different parts of the system. Therefore, a web master can hide all his private data in some tables and allow other users, using a different account, to use another table freely, without compromising security or privacy.

3.0 New Ideas

From the review of the literature that has been performed, new ideas will be explained and discussed, and the impact and uses towards which the ideas may be put. It was clear from the previous chapter that bandwidth is still a limiting factor on the usability of the internet and other slow network systems. The ability to download up to date data quickly is becoming increasingly difficult for the average internet user as the size of files the user wants increases. Moreover, it is becoming increasingly important that users stay up to date with the patches and updates released for their software in order to stay secure. Worms and viruses spread at alarming rates and often require replacement files from software vendors to plug the holes. The rise in popularity of open source software has meant an increase in home users downloading megabytes of data each time a new version released. This is often all source code that for the most part will be the same as the previous version, moreover, the user probably already has the original so is wasting bandwidth downloading mostly the same files again.

3.1 New Thoughts and Ideas

From what has been discovered and reported above, it is felt that a general easy to use system for the distribution of files from web masters to users, that uses a delta algorithm to reduce bandwidth, will fill a niche in the market. The idea would prove a useful addition to download managers currently available or even as a stand-alone product in its own right. Below is a breakdown of the properties it is felt the system should have, these are not concrete plans and indeed may change for the final design of the program.

3.1.1 Platform

As the internet is such a diverse environment, there are literally dozens of different types of devices and computers attached to it. There are many different computer architectures acting as servers and clients, from Apple Macintosh's and IBM compatible PC's to SPARC, DEC and ALPHA computers. The internet is now a heterogeneous environment in which platform independence or at the least multi-platform software is increasingly important. The rise in popularity of multi-platform web browsers such as Mozilla is just one such example of software doing well on different operating systems and on different computer architecture. The cross platform functionality of Mozilla however, is achieved through a lot of hard work as each platform uses a separate branch of one source tree that is updated and developed separately.

A better way is though platform independent programming languages such as Java that allow write once run anywhere programming. As long as a computer or device has a suitable Java runtime environment, applications written in Java will run the same no matter what the underlying operating system or physical architecture is. A program that fills a niche in the market may do well on one platform but there is certainly no harm in widening the potential market for the program by making it platform independent. Many educational and scientific communities and departments run solely on non-IBM PC compatible architectures such as SGI or SPARC workstations therefore a Java program will be readily available to these users.

3.1.2 Interface to Internet

One of the advantages of this idea is that it should not require changing major parts of the typical web server setup in order for it to be implemented. That is, the current client server architecture of the internet should be preserved and programs should not run on the server, as this is rarely allowed by typical web hosts. Instead, the software should run on users' local computers and interface the server through some standard means.

There are two main ways that the system can interact with and store the data on the web server, either through FTP or through a database connection. As has been reported in section 2.2.8 most web hosts offer FTP access to web masters that they can use to transfer and delete files from their site, so this is a valid interface to the web server for storing data there. The advantage of this is that only the web master needs to know the username and password of the site as clients can access the contents of the site through HTTP, which does not require a login. In addition, this method could prove useful if the web master wants the data kept securely because basic HTTP authentication could easily be provided through the use of .htpasswd and .htaccess files residing in the target directory.

It is also possible to connect to a database running on the server through something like the Java JDBC API. Many web hosts provide a MySQL database with their service so this is also a valid option. Consideration must be taken however, if a password is required to access the database as to how a client may receive and store this information without compromising the web masters security. The use of a database will provide a convenient and tidy way to store the data on the server and is

probably most advantageous. This does mean that the web master will have to setup a new user account for the database, an account that does not have many permissions, probably only the ability to run SQL SELECT statements to get data from the database. If done properly the account can be setup to have only access to specific tables in the database, this means the account does not have any access to the rest of the database. What's more, the account does not need a username or password; a blank user account can be setup so that anyone who accesses the database with no username or password has read only access the specific tables. In some situations, this might not be desirable so a username and password could be set and this data distributed to the intended users by some other means.

3.1.3 System Architecture

Server

The system will require two separate halves of development. The first half will be used by the web master who will use the system to create the delta information from the new and old file versions they have. This delta information can then be uploaded to the web server and stored in some manner, either in a database or in normal files. The system could even present an abstracted view of what is present on the server and allow the web master to upload the new files and delete the old ones if necessary. The view the system displays need not be fragmented, as it may well actually be with lots of delta files present on the server, but could use the information to present a user-friendly interface to the files on the server. For example, for one file using the delta system there may be many files on the server that are all cryptically named and unintuitive to a user. The system could present an overall view of these files and show the file in question and its history of updates without the need to worry how the

system is storing data on the server. This is equally true if the data is stored in a database, the user is presented an abstracted version of what is really there to remove them from the complexities of implementation. This would become more important once the number of files under the system's control increases. If it assumed for now that each modification to a file requires an extra file on the server, ten files in a directory each with six changes each will result in sixty files. To any user viewing this through a standard FTP program this will be a meaningless mess of files. However, through using this system the data can be presented in a meaningful, useful and friendly way. The user will also not need to know what files to delete, for example, when a file is removed from the systems control; an interface to the whole operation will be presented to the user that hides this complexity.

Client

For the other half of the development there will be the program that the client uses to receive and apply these updates. It is not assumed that the client and server functionality will be separate programs; indeed, it is likely they will form two halves of the same program; it is just useful to describe the system in this way.

The client system will be able to connect to different servers, or more likely directories on servers. It will then receive a list of the files available and their latest version numbers, size of the download etc. The system may highlight files on the clients' computer that are un-synchronised with the server and suggest these are updated. On selection of the files to update, the client pulls the updates off the web server, either from normal files or from data in a database, and uses this delta information and the old versions of the files on the clients computer to create the

updated file. The system will know what delta information is required and in what order it is to be applied to the old files. For example, the client may be three updates behind the latest release of a file; the system will know this and know to download all three intermediate versions and to apply them in order. Applying them in the wrong order will result in unpredictable and wrong results. The client could retain the delta information itself for the future, enabling the user to go back to any previous version of the file, or at the users discretion this could all be discarded and the new file alone kept.

3.1.4 Protocols

As was discussed in section 2.2.4 the TCP/IP protocol is the most widely used protocol for internet and intranet communications. The benefits of this setup have already been discussed and it is not anticipated that for client-server communications anything else will be used in this project. HTTP and FTP are built on TCP/IP, as are other applications, such as the Java JDBC modules for databases such as MySQL.

3.1.5 Storage

The fact that this system deals with multiple versions of files requires that they be stored somewhere, the storage space available is therefore important. The system necessitates that at least the original file and the deltas to the latest version of the file are stored somewhere. The original file being the base on which the updates are applied, once the original file has been downloaded and updates applied to it by the user, the client has no need to keep the original file and it can be deleted along with the delta information. However, for the server that will be hosting these files this data has to be kept, the original file so people who do not already have it can start using the

system and all the deltas. There is a trade off here between performance and storage requirements, or the number of files required. There are different ways that the updates can be created and applied and a trade off needs to be made as to which to use.

It is possible that for each modification made to a file that a new delta is created that covers *all* the changes since the original file, this reduces the number of files required to update the original file to one, but makes it a large file. It also makes updating from any intermediate versions require a large unnecessary download. Alternatively, the delta can be created from the last update; this will reduce the size of the delta but increase the number of files required to get from original file to latest file. Figures 3.3 and 3.4 below show this difference.

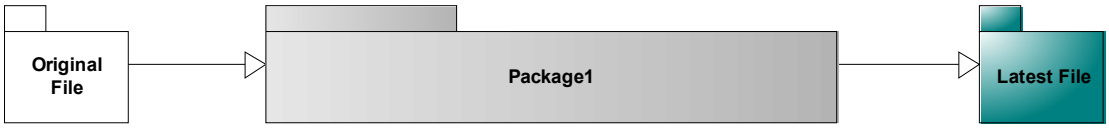


Figure 3.0: A Single delta update system

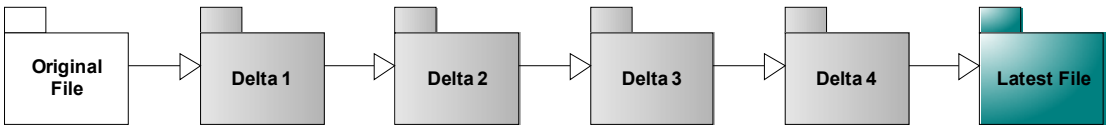


Figure 3.1: A Multiple delta update system

The big disadvantage of only using one delta is that it is not possible to return to intermediate versions of a file; it is either the original or the latest with nothing in between. The second method allows the user to generate the intermediate file versions by applying deltas from the original or latest file, or in reverse, removing the

deltas from the latest file to return to an intermediate file. This makes the system potentially very flexible at the cost of managing the delta files.

There is also the option to make many different delta files, so that only one delta file is needed to produce the latest version, from any intermediate version. The system could also include all the intermediate delta files for each version, as in figure 3.4. This scenario is shown in figure 3.5 below and is a merger of the two methods above.

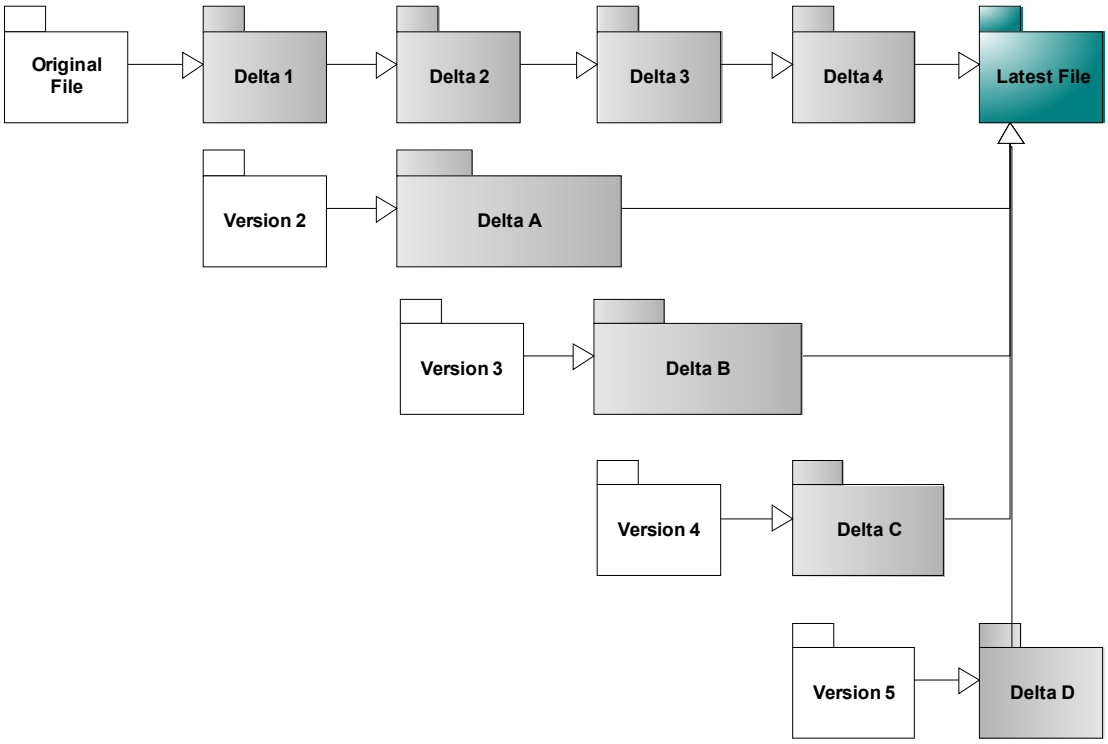


Figure 3.2: A Combination update system

The question of storage space is now important as the need for it increases exponentially with every new modification made. There will also be problems because all the large one-hit-deltas (Delta's A, B, C…), will have to be changed for every modification made to the file. After several versions of the file, this could dramatically decrease performance, as the system creates new deltas for each version

of the file there *has* been, plus the intermediate delta from the last version to the new. So not only does each full version of the file have to be stored – increasing necessary storage space, but also each of the one-hit-deltas decreases performance exponentially.

For example, say an original file starts at 1MB in size. Three changes are made to it, the first decreases the size by 300kb, the second increases the size by 100kb and the third makes no changes that affect the size of the file – only the content. The starting point is a storage requirement of 1,048,576 bytes for the original file, this is then decreased by 307,200 bytes so now the second file is stored also that is 741,376 bytes. Next, this is increased by 102,400 bytes, so a third file that uses 843,776 bytes is also stored. The same amount is stored again for the latest version that does not differ in size – only content. The total storage requirement for *just* the full files (no deltas) is now 1,048,576 + 741,376 + 843,776 +843,776 = 3,477,504 bytes.

Add onto this the size of the changes made plus some more meta-data, to give us the delta data, the total storage required is 3,477,504 + 307,200 + 102,400 + 843,776 = 4,730,880bytes, or 4.5MB.

A system that only stores single deltas that have to be applied successively, as in figure 3.3, would only require 1,048,576 (original file size) + 307,200 + 102,400 + 1,458,176 (the potential, yet unlikely, maximum if the whole file were different) = 2,916,352bytes, or 2.8MB. This is clearly a lot less than the 4.5MB required for the implementation shown (figure 3.5) and described above.

3.2 Problem statement

It has now been identified that there is a need for an easy to use system for distributing file updates to users that will use minimum bandwidth. The system will use delta files that enable the creation of a new version based on a pre-existing version. The system should be able to cope with all file types, be they ASCII text or binary.

The system should be able to configure multiple accounts to archives where file updates are stored. An account is a configuration relating to a particular archive, the server URL and subdirectory etc. An archive is a repository for updates, be it a sub directory or a database on a server. That is, the user should be able to manage multiple connections to servers that use this system to distribute their updates. A web master should also be able to manage multiple accounts relating to archive repositories where they can publish updates. These repositories need not all reside on the same server but may be over multiple servers. A user should also be able to act as a web master (publishing to archives) and an ordinary user (accessing archives) throughout the lifetime of the application.

The distribution of files will be over the internet, using either FTP or a server side database to store the files, and either HTTP or the database to retrieve the files to the client. The system should not require any special programs to be run on the server, allowing implementation on almost all severs used for web hosting.

The view of the files and updates stored on the server should be presented in a user friendly abstracted way that hides the implementation of the system from the user. The user should not see the multiple deltas that are used for updating versions of files but just a list of intermediate and latest updates available. In this way, the user can clearly see what version of a file he/she has and what the latest version available is. The user can then select the version of the file they would like to download, for

example, the user might already have version two of a file and want the third version of the file where the latest version is version five.

When a web master creates a new version of a file, they should be able to use the system to generate the delta file that details how to create the new version from the old version. Once they have done this, they should be able to upload this update to the server and thus make it available to users.

Once a file has been added to the server, users will be able to connect to the server and browse the contents of the server. The software should be able to highlight files that the user has that are out of date and allow them to select to update all out of date files in one operation. Because bandwidth is a major consideration, the application should show the size of any download and a total download size for when multiple files are selected for updating.

3.3 Requirement Specification

3.3.1 Create Delta Information

enable a user to create delta information about new files from old versions

3.3.2 Application of the delta

enable the application of the delta to *only* the correct file version the delta applies to

3.3.3 Store files to server

Method to store data (deltas & new original files) to a user specified web / FTP server.

3.3.4 Abstracted Browsing of an Archive

Allow the abstracted browsing of a store of original files and deltas on a user specified web/FTP server and sub directory of the server.

3.3.5 User to specify the file(s) to download and its version number

Allow a user to select a file and its version to download

3.3.6 Downloading of Files required to create user specified version

Downloading of the original file (if needed) and all the deltas required to create the specified file version.

3.3.7 Application to be usable over multiple platforms

cross platform

3.2 Project Aim

From the above research, an aim for this project has been produced that hopes to tackle some of the issues of limited bandwidth in an application that is accessible to home users. In other words:

To develop a cross platform automatic file updater that uses minimal bandwidth.

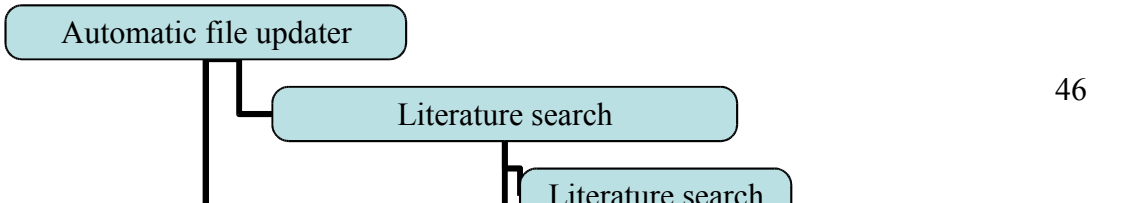
Throughout this chapter the ideas and requirements of the above aim will be discussed and ultimately lead to a requirements specification and into the designing phase.

3.2.1 Objectives

To complete the above aim, five objectives have been drawn up that must all be completed in order to complete the project.

1. A literature search review of relevant systems, protocols, techniques etc.
2. Compile requirements specification
3. Design and build system
4. Evaluate system with analysis of its effectiveness concerning its use of bandwidth as file size and quantity increase.
5. Complete report

Below is a task breakdown diagram of each of these objectives, however, each stage will require further division when it is encountered.



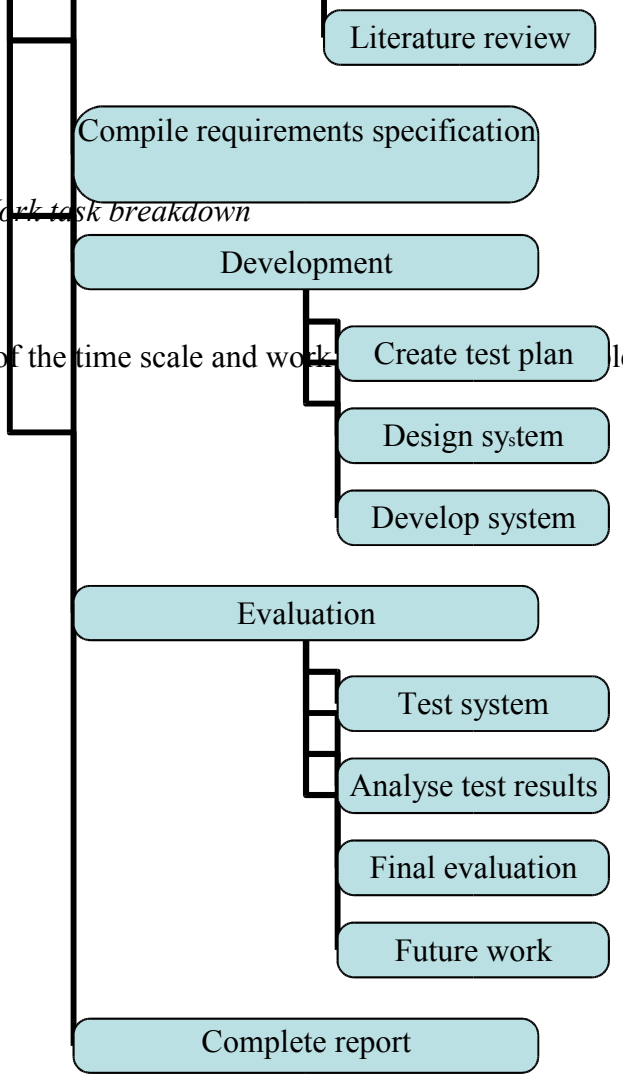


Figure 3.3: Work task breakdown

A Gantt chart of the time scale and work[illegible]le in appendix B.

4.0 Software Development

5.0 Results

6.0 Conclusions and Future Work

7.0 References

ARBECEY INVESTMENT LTD., 2002. *Download Managers* [online]. Available at: http://www.download-managers.com/ [Accessed at 28/10/2003].

HEADLIGHT SOFTWARE, 2003. *GetRight* [online]. Available at: http://www.getright.com [Accessed at 28/10/2003].

FREE BSD, 2002. *FreeBSD Binary Updater Project (binup)* [online]. Available at: http://www.freebsd.org/projects/updater.html [Accessed at 28/10/2003].

Article I.

ALLOT COMMUNICATIONS. 2003. *Effectively Managing Your Internet Connection* [online]. Available at: http://www.allot.com/html/solutions_enterprise_internet_sp.shtm [Accessed at 29/10/2003].

Article II.

FOX A. and BREWER E. A., 1996. *Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation* [online]. Available at: http://citeseer.nj.nec.com/fox96reducing.html [Accessed at: 30/10/2003].

Article III.

CLASSICBRANDING.COM, 2002. *Internet Users; HOW AND WHERE AMERICAN INTERNET USER*[S] *GOES ONLINE* [online]. Available at: http://americanonline.hypermart.net/ [Accessed at 30/10/2003].

FAN, L AND JACOBSON, P. C. Q., 1999. *Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance* [online]. Available at: http://citeseer.nj.nec.com/fan99web.html [Accessed at 30/10/2003].

WIKIPEDIA, 2003. *OSI model* [online]. Available at: http://en.wikipedia.org/wiki/OSI_model [Accessed at 4/11/2003].

CISCO SYSTEMS INC., 1996. *TCP/IP Technology* [online]. Available at: http://www.cisco.com/warp/public/535/4.html [Accessed at 4/11/2003].

SUEL, T. & MEMON, N., 2002. *Algorithms for Delta Compression and Remote File Synchronization* [online]. Available at: http://citeseer.nj.nec.com/576380.html [Accessed at 17/11/2003].

HUNT J J, VO K-P, AND TICHY W F., 1998. *Delta Algorithms: An Empirical Analysis* [online]. Available at: http://citeseer.nj.nec.com/hunt98delta.html [Accessed at 18/11/2003].

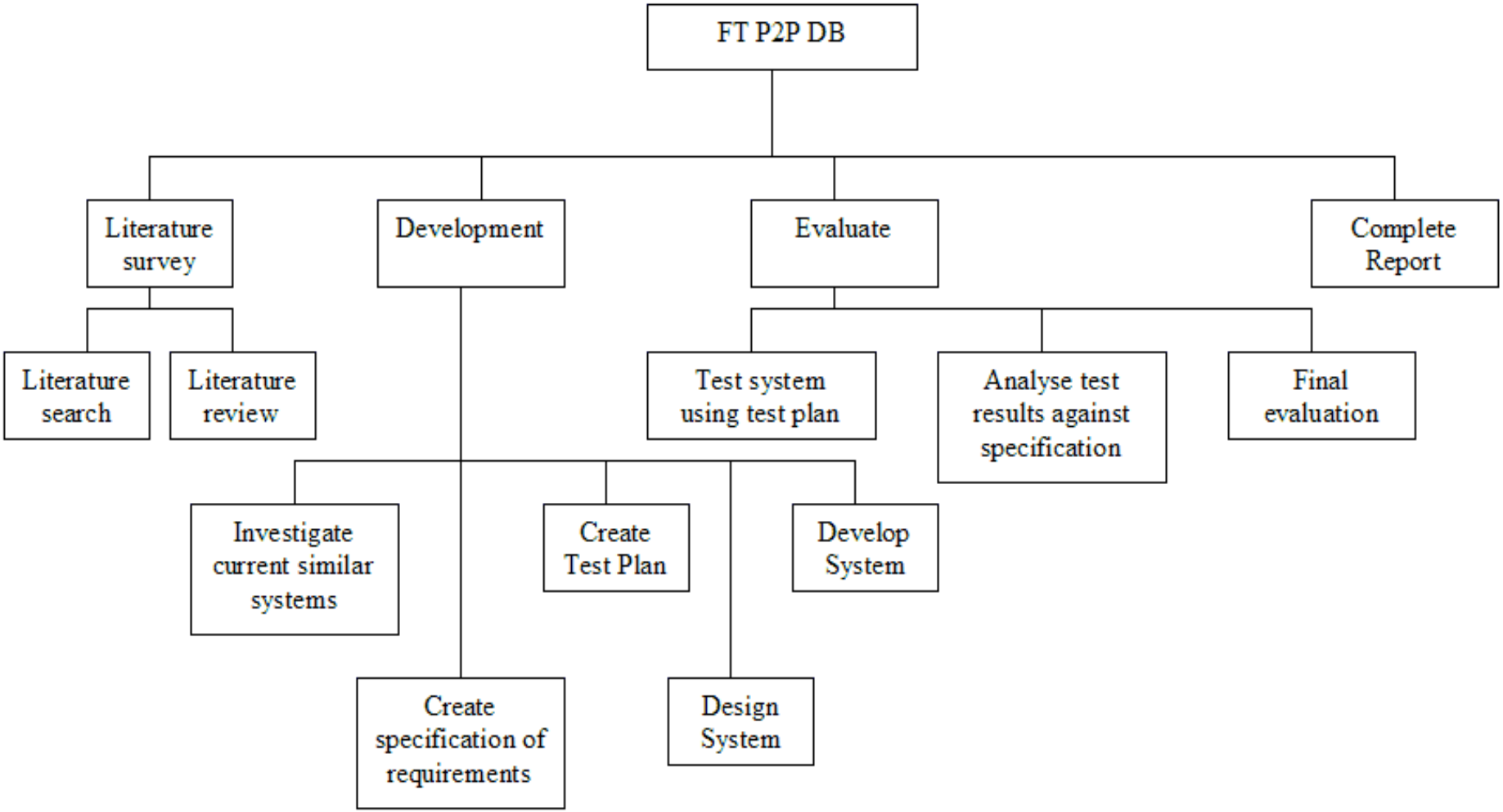
BURNS, R. C., LONG, D. D. E., 1997. *Efficient Distributed Backup with Delta Compression* [online]. Available at: http://citeseer.nj.nec.com/burns97efficient.html [Accessed at 18/11/2003].

MIT Laboratory for Computer Science and RSA Data Security, Inc. (1992). *RFC 1321 - The MD5 Message-Digest Algorithm* [online]. Available at: http://www.faqs.org/rfcs/rfc1321 [Accessed at 18/11/2003].

8.0 Bibliography

Appendix A

The initial idea for this project turned out to be unsuitable as the chosen peer-to-peer architecture would be unworkable in the proposed fault tolerant system. In addition, the alternative of pure client-server architecture was the same as work that had already been done and is now used frequently in Commercial systems from companies such as Oracle and MySQL. This appendix is a record of the work that was performed towards this idea, including the project proposal, aims and objectives and a work breakdown.


FT P2P DB
Literature survey
Development
Evaluate
Complete Report
Literature search
Literature review
Test system using test plan
Analyse test results against specification
Final evaluation
Investigate current similar systems
Create Test Plan
Develop System
Create specification of requirements
Design System

Final Year Project Proposal – BSc computing (Visualisation) (Hons)

Paul Phillips - CC006040

Distributed Software Fault Tolerant Database

We are living in an age where organisations demand 100% reliability and uptime of their IT resources especially in mission critical business systems where the cost of downtime is measured in hundreds of thousands of pounds a minute, jobs or even lives. The ability for a system to stay up and running even in the most extreme circumstances is becoming a necessity for many people. One method of trying to achieve this is through distributed software fault tolerance (FT) whereby the aim is to remove single points of failure in hardware though a software solution. This is often achieved through multiple computers running the same code in synchronisation ready to take over control should they detect an error.

This project aims to prove the software FT concept by designing and building a distributed fault tolerant database program running on multiple computers (nodes) across an ethernet network. The project will study various existing fault tolerant systems available and will aim to enable almost any number of nodes to be added and removed from the system plus the following:

Use of the message-passing paradigm for the seamless addition and removal of nodes to the system both planned and unplanned, allowing for a system to fail, be reset and then rejoin the system as if nothing happened. As a subset of this functionality there will need to be systems for fault checking, re-running code,

isolation of the fault, graceful removal of a node in the system and the synchronisation of new nodes.

Allow multi-user access to the database from any node or a computer not acting as a node but that can use the system. This will allow for access to the system remotely over the network that will improve security of the main fault tolerant system that can then be locked in secure areas.

The project will deliver a written report covering current research ideas and their possible failings. The costs and advantages of different designs and solutions that will be compared, contrasted and evaluated as to their suitability, effectiveness and overall worth to the system. The design found to be most suitable will then form the basis of the software development project that will be planned, documented and developed. The software development will use some or all of the algorithms and techniques mentioned above depending on the analysis and final design. The system will be thoroughly tested and analysed as to its impact to users both in the normal course of running and in the event of faults. Finally, the software will be evaluated against these preliminary aims and objectives, and the actual requirements specification and design. This project will be a development project drawing on present research but hopes to find a gap that the project will fulfil.

The development of the project will follow some standard techniques and methodologies, namely, SSADM, UML design, OO implementation and scientific analysis of different options and decisions that will have to be taken. The basis of this project is running multiple computers and so there are hardware requirements on this project, fortunately two old x86 machines have already been acquired for this, plus a workstation and the possibility of another computer if and/or when required.

Project Outline - Fault Tolerant Distributed Database

Introduction

This project was initially going to be based around a peer-to-peer fault tolerant database, but as we shall discuss, this no longer appears a viable option. All the literature on peer-to-peer databases that has been found so far has centred on systems like Grokster, the file sharing system and fault tolerant routing of the data through the internet. Indeed the leading edge of database research and development is going into similar systems with the advent of grid technology enabled databases, such as Oracle 10*g* that, according to its white paper, enables '*virtualization, dynamic provisioning, resource pooling, self-adaptive systems, and unified management.'* That is not the aim of this project however; the main aim is to create a fault tolerant database system and to study the principles and techniques of fault tolerance in distributed heterogeneous systems.

Distributed databases typically possess some basic features according to Ceri & Pelagatti (1985, p14):

- *Remote database access by an application program; this feature is the most important one and is provided by all systems which have a distributed database component.*
- *Some degree of distribution transparency; this feature is supported to different extents by different systems because there is a strong trade off between distribution transparency and performance.*
- *Support for database administration and control; this feature includes tools for monitoring the database, gathering the information about database*

utilisation, and providing a global view of data files existing at the various sites.

- *Some support for concurrency control & recovery of distributed transactions.*

This project sets out to achieve a sub-set of this functionality whilst incorporating fault tolerance into the database.

Aims and Objectives

This project has one very broad idea, that being, *to develop a distributed fault tolerant database.* In order to achieve this it is necessary that we break this down further and further until there is a clear idea of what is to be done. Firstly, the objectives, which are to be completed in order to close this report:

1) A literature search and review on existing distributed peer-to-peer systems, with special attention to fault tolerant systems.
2) Develop a suitable fault tolerant distributed model, design and implementation.
3) Evaluate the design and implementation based on meaningful statistical data and against the test plan and specification of requirements.
4) Complete the report.

This gives, again, a very rough outline as to what the stages of this project will be, to get a clearer understanding a requirements specification will be needed. There do have to be constraints put on this project to stop is getting out of control, firstly this project does not aim to create an N-version system where multiple different implementations produce truly identical computation results, nor does it attempt to address network or storage infrastructure fault tolerance, that the system should run on to provide true fault tolerance.

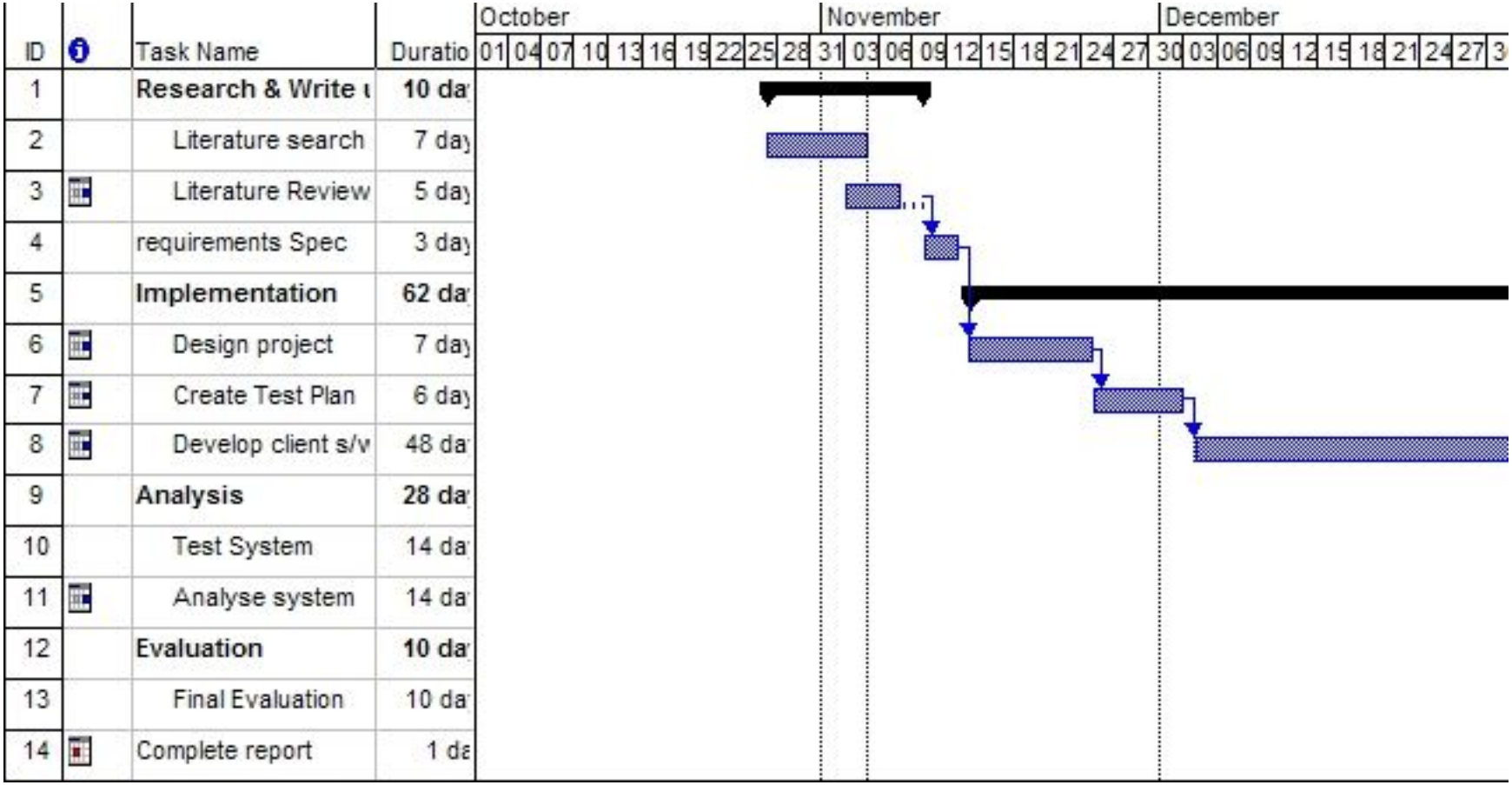
Requirements Specification

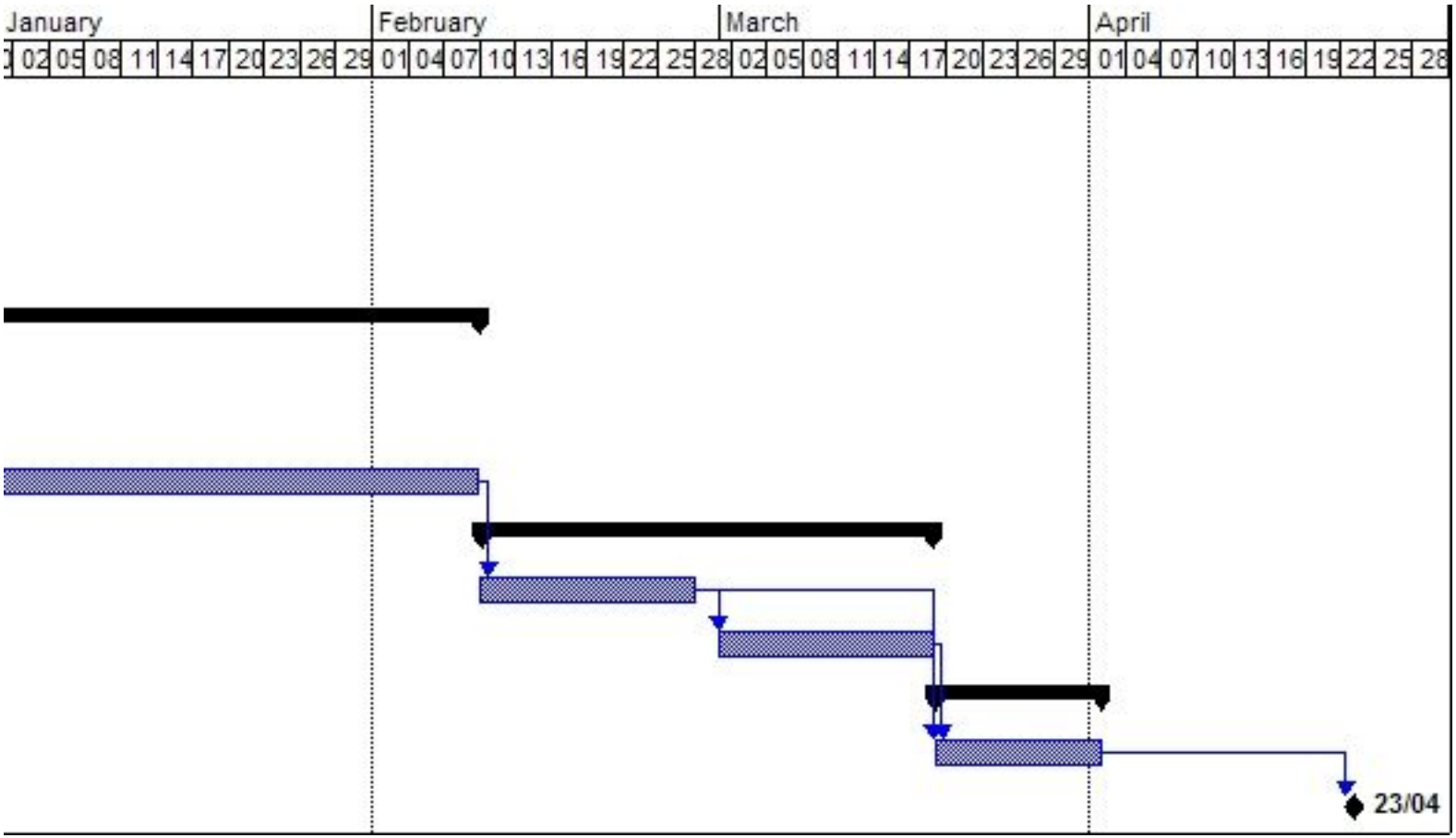
- Remotely accessible database with basic functionality:
    - Querying of data
    - Updating data
    - Creating / deleting tables
    - Inserting / deleting table rows
- Transparent distribution between hosts
- Support for concurrency control and recovery of distributed transactions including:
    - Detection of deadlocks
    - 2-phase-locked paradigm for transactions
- Fault tolerance to allow for transparent removal and addition of database servers from the system

From this document a task breakdown can then be created that will set out the individual tasks to be completed.

Appendix B

Gantt chart of workload and time scales.


ID
Task Name
Duratio
October
November
December
01 04 07 10 13 16 19 22 25 28 31 03 06 09 12 15 18 21 24 27 30 03 06 09 12 15 18 21 24 27 3
1 Research & Write u 10 da
2 Literature search 7 day
3 Literature Review 5 day
4 requirements Spec 3 day
5 Implementation 62 da
6 Design project 7 day
7 Create Test Plan 6 day
8 Develop client s/v 48 da
9 Analysis 28 da
10 Test System 14 da
11 Analyse system 14 da
12 Evaluation 10 da
13 Final Evaluation 10 da
14 Complete report 1 da


January
February
March
April
23/04