On the Effects of Bandwidth Reduction Techniques in Distributed Applications

Hanping Lufei[†], Weisong Shi[†], and Lucia Zamorano[¶]

[†] Department of Computer Science
Wayne State University
431 Stat Hall, Detroit, MI 48202
Phone: (313)577-3186
{*hlufei,weisong*}*@wayne.edu*

[¶] Department of Neurological Surgery
Wayne State University
39 Lande Building, Detroit, MI 48201
Phone: (313)966-0342
lzamorano@neurosurgery.wayne.edu

Abstract

Communication optimization plays an important role in building networked distributed applications. In this paper, we systematically evaluate four bandwidth reduction algorithms, namely direct sending, delta-encoding, fix-sized blocking, *and* vary-sized blocking, *using five types of documents including* source code, images, Web contents, Microsoft Word documents, *and* Latex files. *The experiments were performed under four representative network connection technologies. Performance evaluation results show that different approaches have different performance in terms of different metrics. Completely different results can be achieved by the same algorithm with respect to different types of documents. Network condition can affect some algorithms substantially. Furthermore, the effect of block size to the system performance was also studied.*

1 Introduction

More and more distributed applications, such as distributed and wide-area file systems [1, 8, 10, 15], Internet backup system [2, 12, 17], Web application [5] and distributed database have been deployed on top of the Internet. Meanwhile, we have witnessed the fast growth of diverse network connection techniques, such as local area network (LAN), 802.11 series wireless network (e.g., 802.11a,b,g) [9], cable modem, digital subscription line (DSL), Wireless 3G [26], Bluetooth [7], and traditional phone Dialup service etc. A real challenge in building a distributed application across these heterogeneous network environments is the bandwidth reduction technique.

Several application-specific optimization techniques have been proposed in different contexts. For example, *delta-encoding* $(Delta)$ was proposed by Mogul *et al.* in the context of HTTP traffic [14] to exploit the similarity of Web documents. Rsync [25] synchronizes different versions of the same data by using fix-sized chunks for communications. LBFS [15] takes this a step further by reusing the data chunks across multiple similar files (including multiple versions of the same file).

Although these previous results are promising, two related issues are neglected in these previous efforts. First, none of them take document characteristics into consideration. Second, the adaptability of these algorithms to heterogeneous network environments is yet to be studied in their design. In this paper we systematically evaluate four bandwidth reduction algorithms, namely *direct sending*, *delta-encoding (vcdiff)*[11], *fix-sized blocking*, and *vary-sized blocking*, in terms of two performance metrics: *computing overhead* and *bandwidth requirement*. Although there are quite a few different types of files existing on the Internet, we identified five representative file types based on their usage pattern, and used them as test cases for algorithm comparison purpose. The five types of files are: *source code* (e.g., C/C++ code), *images*, *Web contents*, *Microsoft Word documents*, and *Latex files*. The experiments were performed under four representative network connection technologies: *100Mbps switched Ethernet*, *802.11b wireless LAN*, *cable modem*, and *phone dialup*. Additionally, the effect of different block sizes on the fix-sized blocking algorithm was also investigated. Several interesting observations are found in this paper. Specifically, our experimental results show that:

- Overall, no single algorithm outperforms others in all cases. and different approaches have different performance in terms of different metrics. In general, we found that when the network condition is good, the total time is dominated by the computing overhead, while the bandwidth requirement dominates when the network is slow.

- With regards to the computing overhead, obviously, *direct sending* with the compression operation is the fastest. However, surprisingly, *delta-encoding* using *vcdiff* always generates the smallest difference between two consecutive versions.

- A completely different results can be achieved by the same algorithm when it is applied against different type of documents. Some algorithms have different performance for diverse document types. One example is that for image files, `fix-sized blocking` is the best one in computing time, but for other document types, it is the second worst one among the four algorithms.

- Network bandwidth affects the performance of algorithms substantially. For instance, `fix-sized blocking` is much better than other algorithms for image files as far as total time is concerned in fast networks such as LAN or Wireless LAN. But in relatively slow networks like cable modem or dialup, it is not the best solution anymore.

- The result of performance can be influenced by different parameter settings of the same algorithm. In the `fix-sized blocking` algorithm, changing the block size from 2K bytes to 8K bytes provides a factor of 11.1% reduction of the bandwidth requirement for some kinds of documents. While doing the same thing may result in the increasing of computing time for some other document types such as images.

In summary, each algorithm has its own advantages and disadvantages. They beat each other in different circumstances. Based on these observations an adaptive type-specific model needs to be devised to dynamically choose different approach according to different networks, files and application requirements.

The rest of the paper is organized as follows. A brief description of the four algorithms and five types of documents are presented in Section 2. Then a simple model is abstracted to depict the time spent on a typical communication operation (file update) in Section 3. Section 4 reports the details of performance evaluation, including experimental platforms, documents, performance analysis, and implications. Related work and conclusion remarks are listed in Section 5 and Section 6 respectively.

2 Background

In this section, we describe four differencing algorithms and five typical types of documents used in the following experiments.

2.1 Differencing Algorithms

Several projects have used differencing to reduce network bandwidth requirements on wide-area file system by exploiting similarity between distinct versions of the same files or Web pages, and even among arbitrary documents, such as delta-encoding for Web documents [14] and email files [4], object composition for dynamic and personalized Web contents [19], and block-based techniques for file synchronization [15, 25]. Next we will briefly describe each algorithm and the corresponding protocol.

Delta-encoding ($Delta$) — It was first proposed in the context of HTTP traffic [14]. Currently, the best delta-encoding algorithm is `vcdiff` proposed by Korn and Vo [11]. `vcdiff` is a general and portable encoding format for delta compression, i.e., combined compression and differencing.

Fix-Sized Blocking — Rsync [25] is one of the systems taking this approach. It uses *fix-sized blocking* ($Block_{fix}$) to synchronize different versions. In this approach, files are updated by dividing both files into fix-sized chunks. The client sends digests of each chunk to the server, and the server responds only with new data chunks. Based on old version and the differencing, the new version can be rebuilt. The biggest advantage of this algorithm is simplicity.

Vary-Sized Blocking — Fix-based blocking has problem to find the similarity of insertions and deletions in the middle of the file. Moreover, previous research results show that great similarity exists among arbitrary files in a file system [3, 22]. *Vary-sized blocking* is an approach to exploit this fact. *Vary-sized blocking* ($Block_{varied}$) was proposed in LBFS [15] for reducing traffic further. The idea of LBFS is the content-based chunk identification. Files are divided into chunks, demarcated by points where the Rabin fingerprint [16] of the previous 48 bytes matches a specific polynomial value. This tends to identify portion even after insertions and deletions have changed its position in the file. The boundary regions are called breakpoints. The client then computes a digest of each chunk and sends them to the server, which maintains a database of chunks from all local files. The server responds to the client with bits for new chunks. The client then sends each new chunk to the server.

Direct Sending — For comparison purposes, we implement a direct sending approach as the base line. In this algorithm, we use gzip to compress the file at the sender and decompress it at the receiver. Gzip is a popular data compression program [6] which uses LZ77 algorithm. We use `gzip 1.3.3` in our implementation.

2.2 Document Classification

As we described in Section 1, five representative document types with different characteristics are chosen for our analysis.

Source code — Source code files are basis of any software project. Modern software engineering requires the cooperation of many developers at different locations simultaneously. How to implement synchronization, update, consistency and other issues efficiently are important to the applications. One of the characters of source code is that people usually randomly add, delete or change the content of source code may evenly distributed across the file.

Image — Image files have many formats. In this paper, only BMP images are used. For the precompressed format, such as JPG and GIF, we found it is very difficult to find the similarity between two files using a regular algorithm. Therefore, a more powerful algorithm is required and this is beyond the scope of this paper.

Web Contents — With the prevalence of WWW and recent Web services, many distributed applications are Web enabled, which means, most of the server content and functionality are provided in the HTML format. Recently, several researchers found that there is a large potential for content reusability for Web content, even for dynamic and personalized Web contents [21, 18, 27] because of the use of many common templates across different Web contents.

Latex — Latex is the most powerful document editor and publishing tool, especially for the documents which have a lot of mathematical symbols. Similar to the text file, latex source files consists of text only. The unique feature of latex files is that they are usually modified sequentially during the writing stage, and are modified universally during the revise stages.

Microsoft Word Documents — Even with the appearance of many other document editing software, Microsoft Word is still the most popular software to prepare regular documents. Unlike the latex files, the format of Word documents is binary, which embeds the format and layout information as well as the content itself. As such, a minor change in a Word document might result in a big modification of the total file.

3 Basic Model and Metrics of Interests

We abstract a simple communication model, as illustrated in Figure 1, which consists of a client and a server at the ends. Logically, the model depicts the common communication paradigm between two communication entities in any distributed application.[1] Without losing generality, we assume that both ends have an old version shown as the shadow block in Figure 1. A difference, i.e., the black triangle shown in the Figure, is calculated and sent to the counterpart during the updating phase. In case that there is no old version exists on one end, more message exchanges are required to rebuild the new version. The total time associated with each file update includes two parts: *computing overhead* (T_{comp}) and *communication time* (T_{comm}). The calculation of these two times are described as follows. In Figure 1, T_{cc} and T_{sc} are the computing times on client and server side respectively. T_{comm} is the communication time between the client and the server. Let us denote the current bandwidth between the client and the server by $B_{current}$,

[1]Even for peer-to-peer distributed applications, the notion of client and server is still hold during a given period time of the communication.

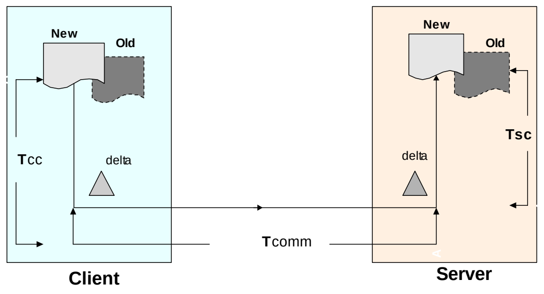


Figure 1. A basic communication model.

C_{LAN}	Pentium-4 2.0 GHz, 512MB RAM, RedHat 8.0, Realtek RTL8139 Family PCI Fast Ethernet NIC
C_{WLAN}	Mobile Pentium-3 1.7GHz, 256MB RAM, RedHat 8.0, D-LInk DWL-650 802.11b adpater
C_{CABLE}	Pentium-3 450 MHz, 192 MB RAM, RedHat 8.0, Netgear CM212 Broadband Cable Modem
C_{DIALUP}	Pentium-3 800MHZ, 512 MB, RedHat 8.0, Realtek RTL8139 Family PCI Fast Ethernet NIC
Server	Pentium-4 2.0 GHz, 512MB RAM, RedHat 8.0, Realtek RTL8139 Family PCI Fast Ethernet NIC

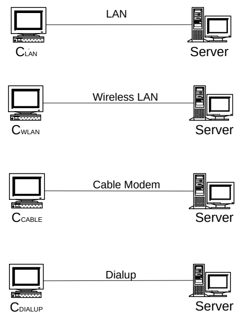


Figure 2. Four different network configurations.

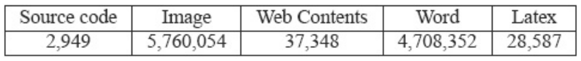
the total time by T_{total}, the delta file size by S_{delta}, and the total computation overhead by T_{comp} respectively. Then the following formulas hold: $T_{comp} = T_{sc} + T_{cc}$, $T_{comm} = \frac{S_{delta}}{B_{current}}$. In the next Section, we will compare different algorithms in terms of two performance metrics: computing overhead (T_{comp}) and bandwidth requirement (S_{delta}).

4 Performance Evaluation

In this section, we first briefly describe the four experimental platforms and document sources, then in turn compare different algorithms in terms of total time, computing time and bandwidth requirement. After that we will discuss block size effect and implications. Finally, based on these results, an adaptive approach is proposed in the end.

4.1 Experiment Platforms and Document Source

We evaluate four different algorithms in four representative state-of-the-art network platforms as illustrated in Figure 2, including 10/100Mbps Fast switched Ethernet LAN, 802.11b Wireless, Cable Modem, and 56K Dialup. To simplify the comparison, the server used in the experiment is fixed, while client are different in each scenario. The configuration of each client, namely C_{LAN}, C_{WLAN}, C_{CABLE}, C_{DIALUP} are listed in the left side of Figure 2. To evaluate these algorithms, we choose to run them in real deployments, instead of in a simulated environment. Note



Source code	Image	Web Contents	Word	Latex
2,949	5,760,054	37,348	4,708,352	28,587

Figure 3. Average size(bytes) of five document types.

that the hardware configurations of clients are not exactly same; however, we can reasonably make an assumption that the processing speed of four client network modules are roughly same.

The documents used in the experiment are obtained from different sources. The source code come from another project with 9 different versions and 17 files in each version. Images are a series of seven continuously changed pictures in BMP format. Web documents are samples of three dynamic web sites, www.cnn.com, www.nytimes.com and www.slashdotcom.com, by downloading every 10 minutes. Word documents are three versions of a master student thesis and Latex files come from seven versions of a research paper. The average size of different document types are listed in Figure 3.

4.2 Comparison of Different Algorithms

In this Section, we analyze the total time from three perspectives, namely document types, algorithms, and network connections. In the following discussion we will represent direct sending by Direct, delta-encoding by Delta, fix-sized blocking by Fix-Block, and vary-sized blocking by Varied-Block respectively. Figure 4 shows the total time in four different network environments. The x axis represents document types and the y axis shows total time in logarithmic scale. Figure 5 illustrates the computing time in different network environments. The horizontal line describes document types and vertical line stands for computing time including both client side and server side. Figure 6 reports the practical bandwidth requirement of each algorithm. The x axis still exhibits document types and y axis displays bandwidth requirements.

4.2.1 Document Types

Documents vary greatly on format, size, compression ratio and other characteristics. Document types will in turn have some effect not only on the efficiency of the algorithm but also on some evaluation metrics.

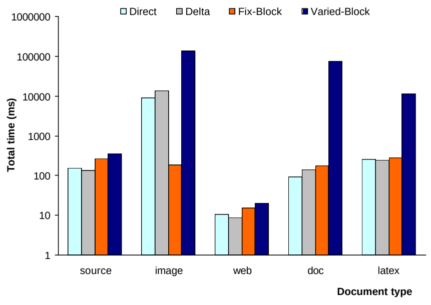
In Figure 4, as far as the source code is concerned, in the LAN, Delta is the fastest algorithm and Varied-Block is the slowest one. The three way protocol of Varied-Block algorithm determines it must spend more time in computing on both sides. Furthermore, since the average size of the source code files is 3K bytes which is close to 2K, the minimal block size of Varied-Block algorithm, so it can not take many advantages of other algorithms in terms of the final delta file size.

Delta is the best algorithm for small size source code files across various network connections. For large size source code files, a prediction could be made that Delta still might be a good choice because people usually change the source code randomly from one version to another instead of appending something at the end, which makes Fix-Block not work well because of the inflexibility of fixed block to handle random insertion and deletion.

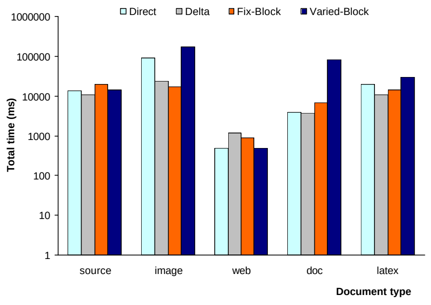
Image is an interesting file type in our evaluation. It does not follow the same pattern as source code file for different network connections. From Figure 6, in comparison with other three methods, Direct, Fix-block and Varied-block, Delta gets the smallest transfer size. But for the total time in LAN Fix-Block becomes the best one amazingly. There are two reasons: First, in Figure 5 the computing time of Delta is much slower than that of Fix-Block in unit of millisecond. Second, in 100Mbps LAN transferring 5M Bytes image file only takes about 1-2 seconds. The advantage on transfer size is dramatically reduced by the 100Mbps bandwidth of LAN. But for Varied-Block change of content may introduce new chunks, merge old chunks or split one chunk into two chunks, which will incur more computing time on comparison of different chunks. It is not good at handling images. Comparing four pictures in Figure 4, we can find that the advantage of Fix-Block decreases with the reduction of network bandwidth. In Cable Modem network it becomes worse than Delta and ever more in Dialup. It is because with low bandwidth the disadvantage of long transfers overwhelms the advantage of small computing time for Fix-Block. Therefore, we argue that for images transmission an adaptive optimization technique should be chosen according to different network environments.

In terms of Web contents, Direct needs more bandwidth requirement than others as shown in Figure 6, because there are many similarities in Web contents resulted from using the same templates. But the computing time of Direct beats other methods. In some cases computing time plays a more important role than bandwidth requirement in total time. But for most of the situations Delta is still a very good solution for Web contents.

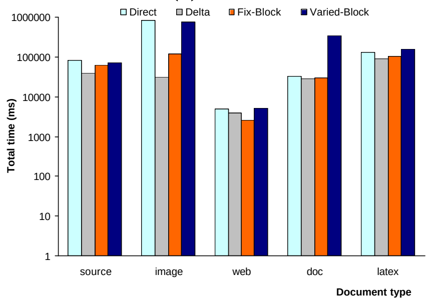
For Word documents, although Varied-Block achieves as good bandwidth requirement as Delta does, as illustrated in Figure 6, but it costs too much more time than any other algorithm. This makes the varied-block algorithm unrealistic for Word documents transmission in terms of total time. of these four algorithms demonstrates better results than others in terms of bandwidth require-



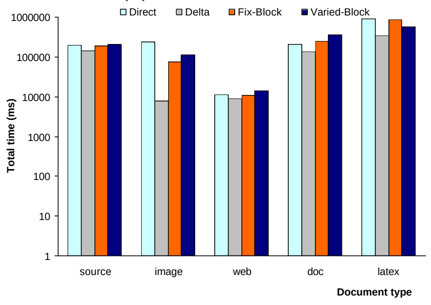
(a) LAN



(b) Wireless LAN



(c) Cable Modem



(d) Dialup

Figure 4. A comparison of total time in different differencing algorithms in four different network environments: (a) LAN, (b) Wireless LAN, (c) Cable Modem, and (d) Dialup.

ment except that, in the LAN, `Delta` seems doing slightly better than others. Similar pattern is observed for `Latex files`.

4.2.2 Algorithms

Now we are in the position to compare different differencing algorithms. Intuitively `Direct` should be the fastest one in computing time. Figure 5 validates this in most cases except for image files.[2] For bandwidth requirement `Direct` is comparable with other algorithms in all document types except image as shown in Figure 6. From Figure 4 (a) and (b) we can find `Direct` uses the least total time among four algorithms for `Word documents`. `Direct` is also useful for some small devices that can not afford expensive computing.

We can easily find that `Delta` always gets the smallest in bandwidth requirement in four methods. With regards to the total time, `Delta` is the winner for `source code`, in most network cases for `Web contents`, `Word`, and `Latex` documents. For many file types the performance of Vcdiff outperforms other algorithms especially in bandwidth requirement.

`Fix-Block` is easy to implement. Although it is not better than `Delta` for most file types it works very well for `images` in broadband network. The implementation of `Fix-Block` ensures that the computing time keeps approximately same for the image files with same file size.

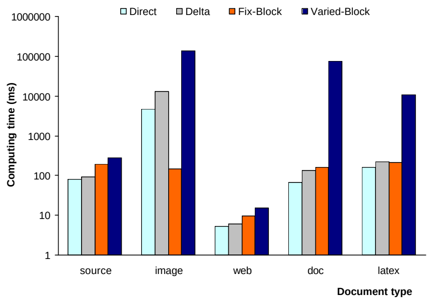
`Varied-Block` is an algorithm based on Rabin Fingerprint. The most significant contribution of this algorithm is to find the similarity of two unrelated files. From Figure 6, it can be seen that `Varied-Block` is better than `Fix-Block` for `Source code`, `Word contents` and `Latex` in terms of bandwidth requirement. But `Varied-Block` is so bad at computing time, as shown in Figure 5 that it is not suitable for any tested document types. For some infrequently changed files like system files dividing them into chunks and saving them into database in advance will greatly improve the overall performance of `Varied-Block`. One of the implementations of this idea is described in LBFS [15].

In summary, `Delta` is a good bandwidth reduction technique for many document types except `images`. `Fix-Block` is the first choice for `images` in high speed network. For low speed networks `Delta` can replace `Fix-Block` as the better choice for `images`.

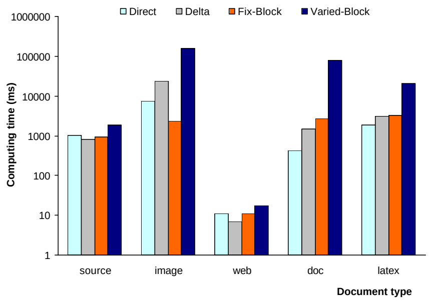
4.2.3 Network Connections

We use four types of networks in our evaluation. Their speeds range from 100Mbps to 56Kbps. With the decreasing of network bandwidth, the average total time of each

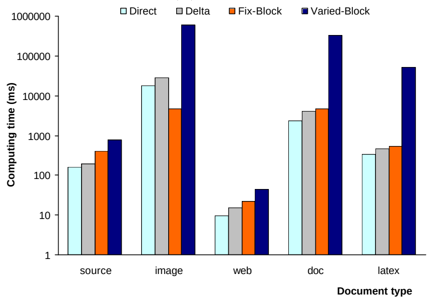
[2]We think Gzip is the culprit of the anomaly computing time for images in Direct.



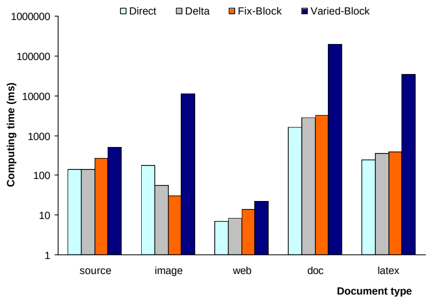
(a) LAN



(b) Wireless LAN



(c) Cable Modem



(d) Dialup

Figure 5. A comparison of computing time in different differencing algorithms in four different network environments: (a) LAN, (b) Wireless LAN, (c) Cable Modem, and (d) Dialup.

algorithm increases accordingly. The size of the final transferred bytes, which is not a concern in fast network like LAN, has more and more affect on the total time when bandwidth reduces. This can be validated by the increasing lengths of bars from LAN to Dialup.

In LAN, as shown in Figure 4(a), the diversity of total time for text files, like `source code` and `Web contents`, is much smaller than that of other files, like `images` and `Word documents`. On the other hand, the speed of LAN amortizes, to some extent, the difference of bandwidth requirements for different document types. Thus the total time is roughly dominated by the computing time. The plots of the total time in Figure 4(a) and Figure 5 follow roughly the same pattern.

In Wireless LAN, bandwidth requirements has more influence on the total time. The advantage of `Fix-block` in the total time for `images` is diminishing with the decreasing of bandwidth. Only for the `Word documents` the total time almost has the same shape as computing time because of the close bandwidth requirement among different method. An anomaly phenomena is in `Web contents` that `Delta`, who is the winner in computing time (Figure 5) and bandwidth requirement(Figure 6) turns into the loser in total time (Figure 4).

For Cable Modem and Dialup, bandwidth requirement gradually dominates the total time, as illustrated in Figure 6 and Figure 4. `Fix-block` in total time for image com-

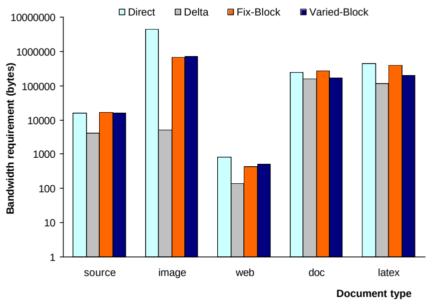


Figure 6. A comparison of different differencing algorithms for bytes transferred.

pletely loses the leading position. In Dialup we found outline of total time is completely controlled by bandwidth requirement because it dominates the total time.

4.3 Effect of Block Size

Block-based algorithms have good performance in bandwidth requirement, as illustrated in Figure 6. Furthermore it is also an excellent method for image files over high speed networks in terms of the total time. In this Section, we are interested in answering the following question: *How does the block size as a parameter affect the performance of the block-based algorithm?* In order to evaluate the effect of block size, we choose different block sizes to do the experiment in terms of computing time and bandwidth requirement.

In Figure 7, horizontal axis represents the individual versions of a specific document type. The left side vertical axis shows computing time that corresponds to three bars for each version in three block sizes test environments. The right side vertical axis stands for bandwidth requirement corresponding to three curves for three block sizes.

4.3.1 Bandwidth Requirement

It can be seen from Figure 7 that the bandwidth requirements for `source code`, `Web contents`, and `Latex` documents remain almost same in different block size cases.

For `source code`, from v2 to v5 most of the files are below 2K bytes, which explains why bandwidth requirement performance overlap for 2K, 4K and 8K block size. With the increasing of file size from v5 to v9 2K block size shows a little bit better achievement than 4K and 8K, which is because the change in text file or `source code` is always localized and for `source code` most the change is likely to be less than 2K. In other words, 2K block size is the best fitting size to reflect random changes in `source code` file among these three block sizes. Lines of 4K and 8K block sizes are still nearly identical because size of most files from v5 to v9 is still below 4k bytes. Same reasons can be used to explain the bandwidth requirement for `Web contents`.

For `Latex`, although it is hard to see any difference between these three curves from the figure, we can observe that the bandwidth requirement goes down a little with the increasing block size from 2K to 8K. The smaller the block size is, the more granularity the delta between versions is divided into, and the more overhead of bandwidth requirement is needed to organize these small blocks. So 8K block size has the best bandwidth requirement and 2K has the worst.

On the contrary, compared with `Latex`, bandwidth requirement pattern is reversed for image files. The smaller the block size is, the lower the bandwidth requirement is. Examining Figure 6, we can see that for `images` `Delta` has much lower bandwidth requirement than `Fix-Block`. By halving the block size, we achieve an obvious improvement, as shown in Figure 7. It is possible that 1k might get even better result than 2K block size, but one thing is already clear that choosing the appropriate block size can effectively reduce the bandwidth requirement.

In terms of `Word documents`, a clear relationship is shown between block sizes and bandwidth requirement. Big block size surpasses small block size as far as bandwidth requirement is concerned. As we know, `Word documents` are binary files that store information about structured text. A small modification in Word editor environment may result in a big difference in binary `Word documents`. Sometimes the size of this difference is bigger than what 2k or 4k block size can hold.

4.3.2 Computing Time

The computing time of different block sizes is very interesting. First let us have a look at the image files, 2K block size is the winner for computing time. Let us assume compression time is Tz_{2k}, Tz_{4k}, Tz_{8k} for 2K, 4K, 8K bytes block sizes respectively and T_c is the comparison time for 2K bytes block then we know that the relationship of $Tz_{2k} < Tz_{4k} < Tz_{8k}$ holds.

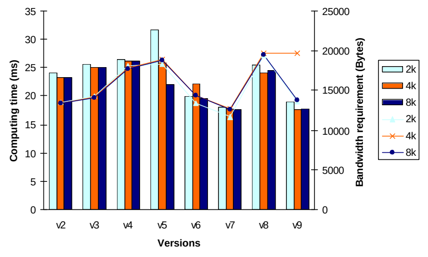
Because the differencing size is assumed to be 2K and the comparison time increases linearly with the block size we have the following relationship of total computing time for 8K bytes between these 3 cases: $4T_c + Tz_{2k} < 4T_c + Tz_{4k} < 4T_c + Tz_{8k}$ It follows the relationship shown in (b) of Figure 7.

For other document types, namely `source code`, `Web content`, `Word document` and `Latex files`, generally 2K block size has the longest computing time and 8k has the shortest among three block sizes. As we discussed in Section 2, in `Fix-block` algorithm, difference in one place may affect the following blocks, so for a 8K bytes data block 4K block size approach may need compression 2 times and 2k block size approach needs 4 times. So we have the following inequations: $Tz_{8k} < 2Tz_{4k} < 4Tz_{2k}$ Combining compression time and comparison time we have $4T_c + Tz_{8k} < 4T_c + 2Tz_{4k} < 4T_c + 4Tz_{2k}$. This explanation complies with the evaluation results shown in Figure 7. Some abnormal data, like the computing time for v8 in `source code` and v7 for `Latex`, is probably contributed to the specific distribution and size of differencing data in a specific file and some deviations in the experiment.

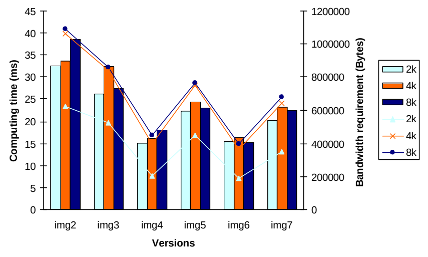
4.4 Implications

Going through the evaluation we find not even one algorithm is good for all document types and all network configurations. For example, although `Delta` benefits `source code`, `Web document`, and `Latex` documents, it delays `images` in terms of the total time. `Fix-Block` is perfect for `images` in Wireless and LAN but it is horrible in Cable Modem and Dialup. Even for the same algorithm different parameter settings may also make totally different performance. As an example, reducing block size from 8K to 2K bytes ends up with 50% improvement in the bandwidth requirement for `images`, while same action on `Word documents` may deteriorate the bandwidth requirement.

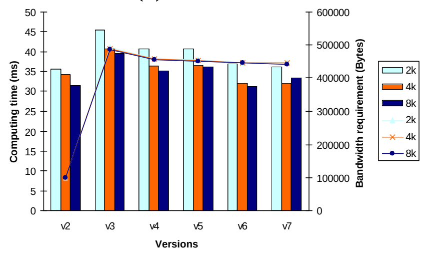
Above observations imply that an adaptive, type-specific communication optimization technique is promising to support applications deployed in heterogeneous and dynamic



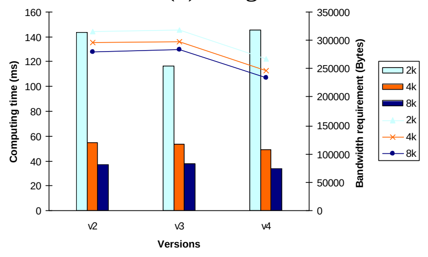
(a) Source code



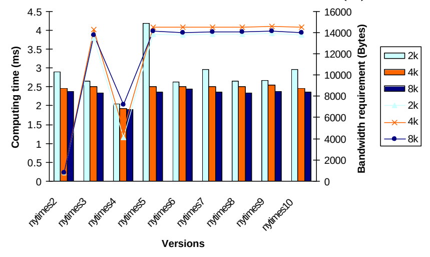
(b) Image



(c) Latex



(d) Word documents



(e) Web contents

Figure 7. A comparison of different block sizes of Fix-Block algorithm in terms of computing time and bandwidth requirements over five difference types of documents: (a)Source code, (b) Image, (c) Latex, (d) Word documents, and (e) Web contents.

changing environments,such as distributed mobile file systems, applications have multiple clients with heterogeneous devices. Currently, we have embedded this adaptive approach in Cegor file system prototype [20], and are deploying this in a computer-assisted surgery applications [13].

The basic idea is as follows. Let us denote the current bandwidth between a client and a server by $B_{current}$, the original file size by $S_{original}$, and the computation overhead of each differencing approach by T_{direct}, T_{delta}, T_{fixed}, and T_{varied}. The corresponding difference sizes are denoted by S_{direct}, S_{delta}, S_{fixed}, and S_{varied} respectively. Ideally, for each specific file and a specific connection, the approach with the minimal value of the total latency should be chosen. This is defined as follows

$$T = min(T_i + \frac{S_i}{B_{current}}), i \in (direct, delta, fixed, varied)$$

As we found in our experimental results, different approaches have different computation complexity and output delta sizes. To adapt to the dynamic changing environments, an *adaptive decision module* is built which takes as inputs the current bandwidth, the target write back file, and the last version of the same file, then outputs an appropriate differencing approach. We plan to investigate this tradeoff in our future work. The more details of this approach are beyond the scope of this paper.

5 Related work

This paper is motivated by several bandwidth reduction techniques in distributed systems. To the best of our knowledge, our work is the first effort on a comprehensive study and evaluation of these differencing algorithms in a general framework.

Delta-encoding (Delta) was first proposed by Mogul *et al.* in the context of HTTP traffic [14]. This approach could dramatically reduce network traffic in cases where a client and server shared a past version of a Web page, termed a "delta-eligible" response. Currently, the best delta-encoding algorithm is Korn and Vo's *vcdiff* [11]. *Fix-sized blocking* was used in the Rsync [25] software to synchronize different versions of the same data. In this approach, files are updated by dividing both files into fix-sized chunks. The client sends digests of each chunk to the server, and the server responds only with new data chunks. *Vary-sized blocking* was proposed in LBFS [15] for further reducing traffic. The idea behind LBFS is that of content-based chunk identification. Files are divided into chunks, demarcated by points where the Rabin fingerprint [16] of the previous 48 bytes matches a specific value. This tends to identify a portion even after insertions and deletions have changed its position in the file.

Recently, several projects such as CASPER wide-area file system [24], Pond prototype [17], Pastiche backup system [2], adopt vary-sized blocking to either improve the system performance or reduce the storage requirements. We believe our work compliments these efforts, and the result of this paper can be applied in their work directly.

Spring and Wetherall have proposed a protocol independent technique for eliminating redundant network traffic [23]. When one end wants to send data that already exists at other end, it instead sends a token specifying where to find the data at the other end.

6 Conclusions and Future Work

In this paper, a comprehensive study of four bandwidth reduction techniques is presented, in terms of two performance metrics: *computing overhead* and *bandwidth requirement*. The evaluation performed under four representative state-of-the-art network connection technologies, against five representative types of documents. We found that different approaches have different computation complexity and output delta sizes. No one is the best for all kinds of document in all network environments. Implied by these observations, we argue that an adaptive type-specific approach which considers both the computing overhead and dynamic changing network is promising. Our future work includes evaluating this adaptive approach in the context of some real distributed applications, such as computer-assisted surgery application, distributed file systems, peer-to-peer backup system, and distributed database systems.

References

[1] B. Callaghan and P. Staubach. NFS Version 3 Protocol Specification, rfc 1813, June 2000.

[2] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. *Proc. of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[3] J. R. Douceur and W. J. Bolosky. A large-scale study of file system contents. *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pp. 59-70, May 1999.

[4] F. Douglis and A. Iyengar. Application-specific delta-encoding via resemblance detection. *Proc. of the USENIX 2003 Annual Technical Conf.*, June 2003.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Bernes-Lee. RFC 2616: Hypertext transfer protocol, HTTP/1.1, 1999, `http://www.ietf.org/rfc/rfc2616.txt`.

[6] Gzip tool, `http://www.gzip.org`.

[7] J. Haartsen. BLUETOOTH– The universal radio interface for ad hoc, wireless connectivitity. *Ericsson Review*, 1998.

[8] J. H. Howard, M. Kazar, S. Menees, d. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems* 6(1):51–81, Feb. 1988.

[9] IEEE. IEEE std 802.11 — wireless LAN medium access control (mac) and physical layer (phy) specifications, 1997.

[10] M. Kim, L. Cox, and B. D. Noble. Safety, visibility, and performnace in a wide-area file systems. *Proc. of the 1st USENIX Conf. On File and Storage Technologies*, Jan. 2002.

[11] D. G. Korn and K.-P. Vo. Engineering a differencing and compression data format. *Proceedings of the 2002 USENIX Annual Technical Conference*, June 2002.

[12] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard. A cooperative internet backup scheme. *Proc. of the USENIX 2003 Annual Technical Conf.*, June 2003.

[13] H. Lufei, W. Shi, and L. Zamorano. A comparison of different data compression approaches in computer-assisted surgey. Tech. Rep. MIST-TR-2004-005, Department of Computer Science, Wayne State University, Feb. 2004.

[14] J. C. Mogul, F. Douglis, a. Feldmann, and B. Krishnamurthy. Potential Benefits of Delta-Encoding and Data Compression for HTTP. *Proc. of the 13th ACM SIGCOMM'97*, pp. 181-194, Sept. 1997, `http://www.douglis.org/fred/work/papers/sigcomm97.pdf`.

[15] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, Oct. 2001.

[16] M. O. Rabin. Fingerprinting by random polynomials. Tech. Rep. TR-15-81, Harvard Aiken Computation laboratory, 1981.

[17] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. *Proc. of the 2nd USENIX Conf. On File and Storage Technologies*, pp. 1-14, Apr. 2003.

[18] W. Shi, E. Collins, and V. Karamcheti. Modeling object characteristics of dynamic web content. *Journal of Parallel and Distributed Computing (to appear)*, Sept. 2003.

[19] W. Shi and V. Karamcheti. CONCA: An architecture for consistent nomadic content access. *Workshop on Cache, Coherence, and Consistency(WC3'01)*, June 2001, `http://www.cs.wayne.eud/~weisong/papers/wc301.pdf`.

[20] W. Shi, H. Lufei, and S. Santhosh. Cegor: An adaptive, distributed file system for heterogeneous network environments. Tech. Rep. MIST-TR-2004-003, Department of Computer Science, Wayne State University, Jan. 2004.

[21] W. Shi, R. Wright, E. Collins, and V. Karamcheti. Workload characterization of a personalized web site — and it's implication on dynamic content caching. *Proc. of the 7th International Workshop on Web Caching and Content Distribution (WCW'02)*, pp. 1-16, Aug. 2002, `http://www.cs.wayne.edu/~weisong/papers/wcw02.pdf`.

[22] M. Spasojevic and M. Satyanarayanan. An empirical study of a wide-area distributed file system. *ACM Transactions on Computer Systems* 14(2):200–222, May 1996.

[23] N. T. Spring and D. Wetherall. A protocol independent technique for eliminating redundant network traffic. *Proc. of ACM SIGCOMM'00*, pp. 87-95, Aug. 2000.

[24] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, t. Bressoud, and a. Perrig. Opportunistic use of content addressable storage for distributed file systems. *Proc. of the USENIX 2003 Annual Technical Conf.*, June 2003.

[25] P. Tridgell and P. Mackerras. The rsync algorithm. Tech. Rep. TR-CS-96-05, Department of Computer Science, Australian National University, 1996.

[26] U. Varshney and R. Vetter. Emerging Mobile and Wireless Networks. *Communications of the ACM* pp. 73–81, June 2000.

[27] C. E. Wills and M. Mikhailov. Studying the impact of more complete server information on web caching. *Proc. of the 5th International Workshop on Web Caching and Content Distribution (WCW'00)*, 2000.