Fine-Grain Adaptive Compression in Dynamically Variable Networks

Calton Pu and Lenin Singaravelu
CERCS, Georgia Institute of Technology
Atlanta, GA 30332-0280, USA
{calton, lenin}@cc.gatech.edu

Abstract

Despite voluminous previous research on adaptive compression, we found significant challenges when attempting to fully utilize both network bandwidth and CPU. We describe the Fine-Grain (FG) Mixing strategy that compresses and sends as much data as possible, and then uses any remaining bandwidth to send uncompressed packets. Experimental measurements show that FG Mixing achieves significant gains in effective throughput, particularly at higher network bandwidths. However, non-trivial interactions between system components and layers (e.g., compression algorithms and middleware settings such as block size and buffer size) have significant impact on the overall system performance. Finally, the trade-offs and performance profiles of FG Mixing are measured, observed, and found to be consistent over a wide range of combinations of compression algorithms (GZIP, LZO, BZIP2), workload compression ratios (from 1 to 4), and network bandwidth (from 0 to 400 Mbps).

1. Introduction

Adaptive compression is a classic technique to improve the effective throughput of networks in changeable environments, e.g., limited, variable, and sometimes shifting bandwidth constraints in shared or wireless networks. Recent examples include the ACE toolkit [18], which switches between compression mode when bandwidth is the constraint and no-compression mode when bandwidth is no longer a constraint, and chooses the most effective compression algorithm for each workload [19]. Despite the variety of work done on adaptive compression, there are some interesting unexplored issues.

The main contribution of this paper is a significant improvement in the effective throughput of adaptive compression through fine-grain adaptation by mixing compressed and uncompressed packets in transmission. We show that the fine-grain adaptation mechanisms provide higher throughput than either the compressed approach or the non-compressed approach even in high speed networks. The goal of fine-grain adaptation is to fully utilize both the CPU and the network to provide the highest possible throughput. We also show that achieving such full resource utilization is non-trivial. Seemingly small implementation differences (e.g., send-in-order vs. send-out-of-order) have significant impact on performance. Furthermore, careful tuning of middleware parameters (e.g., buffer size and block size) is critical for achieving the best performance.

The robustness of our evaluation and analysis is based on experimental results from a wide variety of settings. We measured the bandwidth consumed by compression algorithms with different compression speeds and efficiency (GZIP, LZO, BZIP2), workload compression ratio (from 1 to 4), and network bandwidth (from 0 to 400 Mbps).

The paper is organized as follows. Section 2 summarizes the trade-offs between CPU and bandwidth in adaptive compression. Section 3 evaluates the Simple Switching adaptation strategy. Section 4 describes the Fine-Grain Adaptive Mixing of packets. Section 5 investigates the complications arising due to algorithm parameters and competition for resources. Section 6 studies the robustness of our results in a wide range of system configurations, e.g., faster networks. Section 7 summarizes relevant related work and Section 8 concludes the paper.

2. Problem Description

Traditionally, adaptive compression work assumes that network bandwidth is the dominant bottleneck in the system. The main objective of compression is to best utilize this limited bandwidth. For example, a careful choice of appropriate compression algorithm

for each workload can increase effective transmission throughput in each case [18][19]. Under this assumption, one of several compression algorithms should always be used.

While the bandwidth bottleneck assumption holds for slow networks, in current networked and cluster systems we have a wide range of network bandwidths. For starters, today's typical Ethernet connections vary from 100Mbps to Gigabit. Furthermore, there are several reasons network available bandwidth may vary over time. For example, a large shared network may have variable competing traffic or temporary congestion in intermediate routers. Other scenarios include wireless networks where environmental conditions may influence network performance and CPU scheduling decisions based on resource availability such as battery power. In these situations, network bandwidth is the bottleneck only part of the time. Consequently, we need to investigate the adaptive compression techniques that optimize effective throughput when the available network bandwidth is no longer the only bottleneck in the system.

In this paper, we focus on the trade-offs between network bandwidth and CPU in a dynamically variable network environment. In our graphs, the available network bandwidth is the primary variable. At the low end of available bandwidth is the traditional absolute-network-bottleneck zone, where network bandwidth is always the bottleneck and 100% compression is guaranteed to win over alternatives. This is the case as long as the available network bandwidth remains insufficient to transmit all the data being compressed.

The absolute-network-bottleneck zone is delimited by Max-Compression-Bandwidth, the point where compression fully utilizes the CPU and the compressed data fully occupies the available network bandwidth. Concretely, Figure 1 shows the Max-Compression-Bandwidth at about 44Mbps for an actual combination of CPU, network, compression algorithm, and data workload (explained in Section 3.2). When network bandwidth available exceeds Max-Compression-Bandwidth, pure compression is no longer guaranteed to win, since it starts to leave some network bandwidth unused.

Above Max-Compression-Bandwidth, in the variable-bottleneck-zone, pure compression saturates the CPU and underutilizes network bandwidth. Conversely, no-compression may saturate the network bandwidth, but leaves CPU underutilized. The goal of fine-grain adaptive compression techniques to take advantage of these underutilized resources. Further, we will show that their performance critically depends on middleware settings.

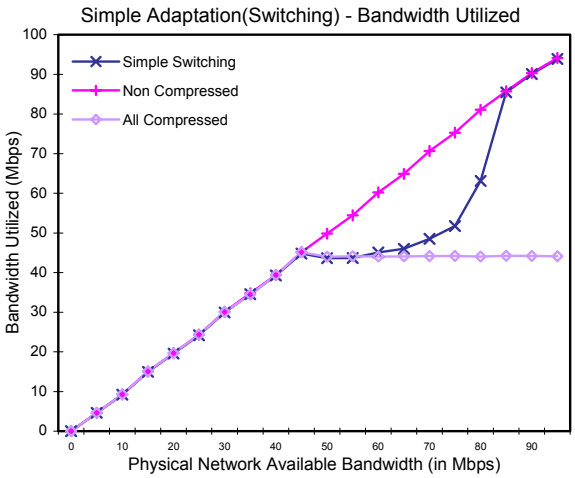

Figure 1 Bandwidth Used by Simple Switching

3. Simple Switching Adaptation

3.1. Simple Switching Algorithm

A straightforward approach to adaptive compression is to avoid the critical bottleneck in the system. If network bandwidth is saturated we adopt the All-Compressed strategy to compress all packets and improve bandwidth utilization. If CPU is saturated we switch to the Non-Compressed strategy to stop compression and send only uncompressed packets. The Simple Switching approach is in either the all-compressed or the non-compressed mode at any given moment in time. An example of the switching approach is the ACE system [18]. In this section, we evaluate the performance of the switching approach.

3.2. Experimental Setup

Our experiments were conducted on two Dell Precision 350n workstations, each running an Intel Pentium-4, 2.24 GHz processor, with 512KB cache, 533MHz front side bus, and 512MB RDRAM. (These machines arrived with their hyper-threading feature disabled by default.) Each of these machines runs Redhat Linux 7.3 with the 2.4.18-3 kernel. The machines were connected through 100 Mbps Ethernet for most of the experiments and a Gigabit Ethernet switch for Figure 14.

We used the "tc" tool under Linux to limit and control available bandwidth between the two nodes. All our implementations use TCP as the transport layer protocol. The main workload is a file containing the coordinates, velocities and types of atoms from a molecular dynamics application (one of the workloads

used in [19]). Except for the figures in Section 6.3, where we compare several different compression algorithms, all the experiments use the GZIP library, version 1.2.4 [4][8], based on the Lempel-Ziv (LZ) algorithm. The average compression ratio of GZIP on this workload is about 2 to 1. In Section 6 we describe additional experiments that show the robustness and generality of our results and analysis with respect to other workloads, compression algorithms, and faster networks.

The graphs in the paper show experimental results with sufficient repetition to achieve the 95% confidence level and a targeted error band of less than 1 Mbps. The standard deviation in all cases was less than 1 Mbps.

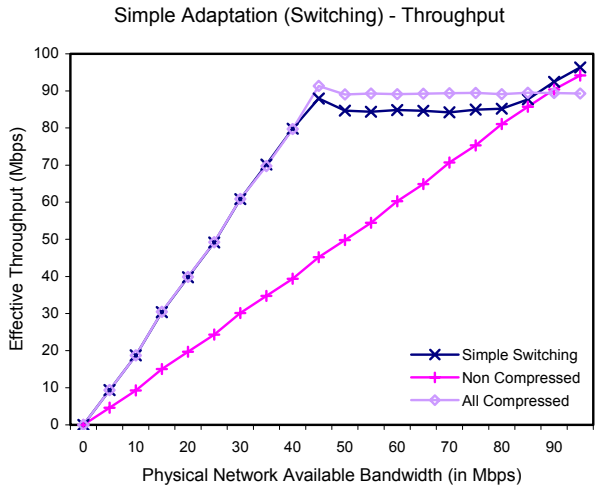

Figure 2 Throughput of Simple Switching

3.3. Evaluation of Simple Switching

Figure 1 compares the network bandwidth consumption of three strategies: All-Compressed (light diamonds), Non-Compressed (crosses), and Simple Switching (dark X's). The main experimental variable is the variation in available network bandwidth, from 0-100Mbps. The Y-axis shows the physical network bandwidth consumption by the three strategies. All-compressed sends only compressed data. Non-compressed sends only uncompressed data. Simple Switching is the strategy outlined in Section 3.1.

Figure 2 compares the effective throughput achieved by the three strategies. Figure 2 is obtained by decompressing the output of the All-Compressed and Simple Switching strategies and comparing them with the bandwidth consumption of the Non-Compressed strategy.

We divide the graphs into two zones. First, the absolute-network-bottleneck zone appears on the left side Figure 1, delimited by the Max-Compression-Bandwidth at about 44Mbps. In this zone, all three strategies use all the available bandwidth since the network bandwidth is the bottleneck. All-Compressed always wins in this zone, as shown in Figure 2. Accordingly, the Simple Switching strategy always uses the All-Compressed strategy in this zone.

Second, in the variable-bottleneck-zone (from 44Mbps and up), All-Compressed becomes unable to utilize the increasingly available bandwidth (the flat diamond curve in Figure 1, and Figure 2), since it saturates the CPU. The Non-Compressed strategy is shown as the diagonal line in both Figure 1 and Figure 2, gradually gaining on the All-Compressed in the variable-bottleneck-zone and eventually winning at about 88Mbps.

At Max-Compression-Bandwidth (about 44Mbps), when the CPU becomes the bottleneck, Simple Switching starts probing available network bandwidth by sending some uncompressed packets. If probing finds that the available bandwidth is greater than the maximum compression throughput (Product of Max-Compression-Bandwidth and average compression ratio), it is advantageous to stay in the Non-Compressed mode. On the other hand, if probing finds the available bandwidth to be less than the maximum compression throughput, it is better to compress and it reverts back to the All-Compressed mode.

Similarly, when Simple Switching enters the Non-Compressed mode, it starts to probe for available CPU. In a way analogous to the network bandwidth probing, the CPU probing compresses a small percentage (5% in our implementation) of packets. If sufficient CPU is encountered, it reverts to All-Compressed mode.

Figure 3 analyzes the mix of packets sent, separating the compressed data from the uncompressed data. The dark X's represent the total bandwidth consumed, which is divided into two components. The light diamonds show the compressed packets, which dominate on the left.

In Figure 3, we see the effect of CPU and network probing. Once we move out of the absolute-bottleneck-zone (around 44 Mbps), we see an increase in the number of uncompressed packets due to network probing. As available bandwidth increases, the duration and frequency of this probing rises, and so does the contribution of non-compressed packets to the total bandwidth. Similarly, even beyond the variable-bottleneck zone, we see residual compressed bandwidth. This is due to CPU probing and it is responsible for the slight advantage that Simple Switching has over the Non-Compressed strategy (upper right hand corner of Figure 2).

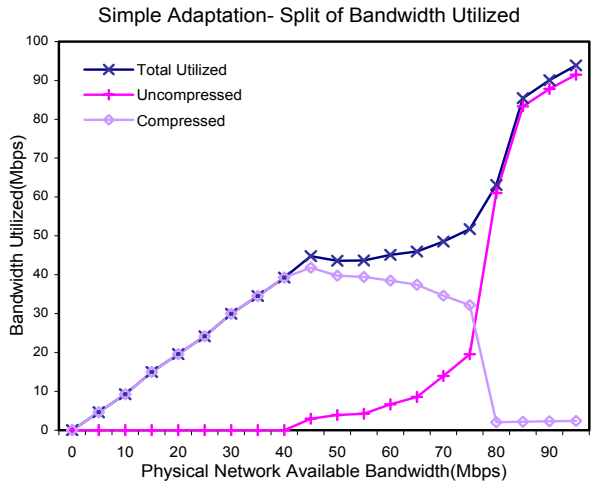

Figure 3 Packet Mix in Simple Switching

In summary, Simple Switching appears to get "the best of both worlds" by tracking the winning strategy at the two ends of network bandwidth spectrum. However, the oscillating behavior of Simple Switching in the variable-bottleneck-zone makes it under-perform with respect to the other two strategies. Compared to Non-Compressed, Simple Switching underutilizes the available bandwidth by as much as 20Mbps at 70Mbps available bandwidth (the downward curve in Figure 1). Compared to All-Compressed, Simple Switching has lower effective throughput by a non-trivial margin (up to 10% between 50Mbps and 70Mbps in Figure 2). Additionally, the residual gain due to use of compression beyond 88Mbps (Figure 2) indicates that further improvements could be had with a fine-grain adaptation techniques.

4. Fine-Grain Mixing of Packets

One way to escape from the oscillation, and hence inefficiencies found in Simple Switching is to use fine-grain adaptation, where the system sends both compressed and uncompressed packets at the same time, instead of switching modes. The goal of the Fine Grain (denoted as FG in the rest of the paper) Mixing adaptive strategy is to fully utilize both CPU and network bandwidth. We achieve this goal by compressing as much as feasible, and start sending uncompressed packets when additional bandwidth becomes available. While a relatively simple idea, FG Mixing introduces non-trivial interactions between system components and layers.

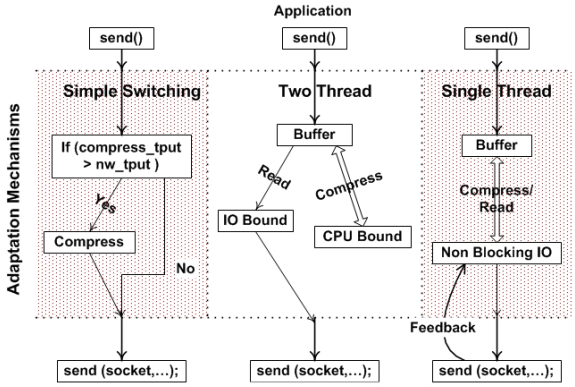

Figure 4 A Summary of the Three Strategies. Simple Switching performs a comparison to enable/disable compression. The other two rely on the slowness of IO to parallelize compression and IO.

4.1. Two-Thread FG Mixing

A straightforward implementation of the FG Mixing method uses two threads, each dedicated to one resource: CPU or network bandwidth. The Compression Thread compresses packets and the Packet Sending Thread sends packets. The two threads communicate through a common queue in temporary buffer (for a system-level implementation the TCP buffer could be used). The Compression Thread picks up uncompressed packets from the queue and puts back compressed packets. The Packet Sending Thread picks up packets (both compressed and uncompressed) from the queue and sends them out. The job of the Compression Thread is to compress the waiting packets so that all available CPU (not used in packet sending) is utilized to increase effective throughput.

We considered and implemented two variants of the above algorithm. Although their implementation differences appear minor at the first glance, they ended up with non-trivial performance differences. The first variant is called send-in-order, with both threads starting from the queue head. The second variant is called send-out-of-order, with both threads working on the next available packets on the queue. While the send-in-order variant is simpler, it requires synchronized access to the queue. As the network speed increases, the effect of synchronization delay on the Sending Thread is exacerbated. The send-out-of-order variant eliminates these delays, but it requires the receiver to sort the incoming packets in a buffer.

Figure 5 compares the effective throughput of the two variants on a 100Mbps Ethernet and Figure 6 on a Gigabit network. The send-out-of-order variant

outperforms the send-in-order variant by less than 0.5Mbps up to about 320Mbps, when the synchronization problems start to slow down the send-in-order variant. The send-out-of-order implementation starts to slow down at about 380Mbps.

The performance curves of FG Mixing (denoted by dark X's) and Non-Compressed (denoted by light +'s) together with the secondary Y axis form a trapezoid. The rising left leg of the trapezoid is incident on the All-Compressed line indicating two things – All the data sent is being compressed and that the network is the bottleneck in this region. The upper base starts where the All-Compressed line flattens out indicating a CPU bottleneck. Beyond this point the Non-Compressed and FG Mixing lines are nearly parallel indicating that any increase in the throughput of FG-Mixing is due to increase in number of non-compressed packets. This trapezoidal figure is characteristic of FG Mixing and can also be observed in later graphs.

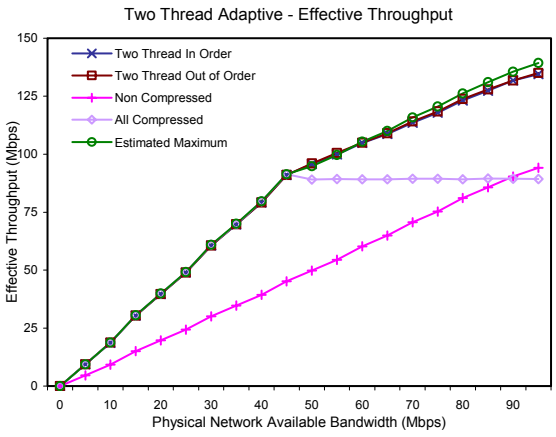

Figure 5 Throughput of 2-Thread FG Mixing

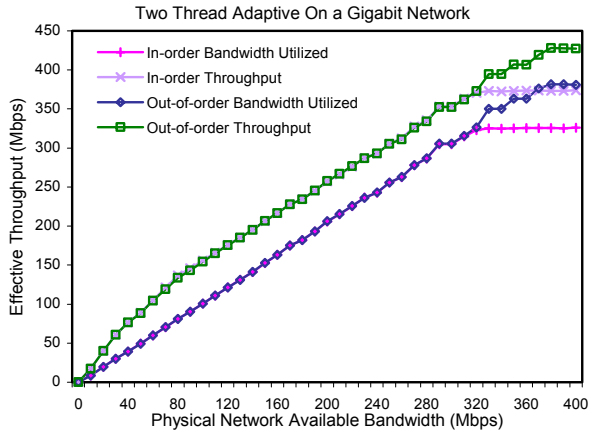

Figure 6 FG Mixing in Gigabit Network

4.2. Estimated Maximum Achievable Throughput

Although Figure 5 shows significant performance gains of FG Mixing over All-Compressed and Non-Compressed, a remaining question is whether FG Mixing could be further improved. (The Simple Switching strategy is omitted for clarity, since it emulates the combination of All-Compressed and Non-Compressed.) To answer this question, we define an estimation of the practical limit on the achievable throughput for a given workload and compression algorithm, called Estimated Maximum achievable throughput (line with circles in Figure 5).

The Estimated Maximum is informally defined as an idealized strategy. It fully utilizes all available CPU to compress and send data. In addition, it fully utilizes the residual bandwidth by sending uncompressed packets up to the physical bandwidth limit. This means in the absolute-network-bottleneck zone, the Estimated Maximum is equal to the throughput achieved by All-Compressed, since there is no excess bandwidth available. And in the variable-bottleneck-zone, the Estimated Maximum is equal to the difference between the input for the compression algorithm and its output plus the entire available physical bandwidth. We believe the Estimated Maximum achievable throughput strategy is appropriately named since it consumes all available CPU and network bandwidth.

$$EstMax = Compr_{in} + PhysBand - Compr_{out} \quad (1)$$

The first term represents full CPU utilization. The next two terms give the residual bandwidth, which is to be utilized by sending uncompressed packets. This equation assumes that sending packets consumes negligible CPU.

In Figure 5 we see a very small difference between the Estimated Maximum and FG Mixing (less than 4.8 Mbps for the send-in-order variant and 4.3 Mbps for the send-out-of-order variant). In other words, the FG Mixing (particularly the send-out-of-order variant) is near the practical limit of achievable bandwidth by adaptive compression. This is the result of fully utilizing both the available CPU and network bandwidth.

However, there are some complications in the implementation of the send-out-of-order two-thread variant, e.g., the need for the client side to sort the potentially out of order packets. We designed and implemented a refined single-threaded implementation to address these issues.

4.3. Single-Thread FG Mixing

The single thread implementation improves the two-thread variants by replacing the Packet Sending thread with kernel-level implicit multithreading, e.g., by using Unix style non-blocking sockets. As long as the TCP buffer is non-empty, a non-blocking socket will keep sending the packets as fast as TCP allows. In the mean time, the single thread keeps compressing the packets in the buffer. Thus we are able to send packets in order while avoiding the synchronization problems of the send-in-order variant of two-thread FG Mixing.

If network bandwidth is the bottleneck, the single thread strategy will keep the entire buffer compressed. This is the case of absolute-network-bottleneck zone. In the variable-bottleneck zone, since TCP will keep sending as long as the buffer is non-empty, all the available network bandwidth will be utilized with a mix of compressed and uncompressed packets. Similar to the send-out-of-order variant of FG Mixing, the single thread implementation of FG Mixing also achieves performance near that of Estimated Maximum. The performance figure for the single thread implementation is similar to Figure 5. A comparison of the above adaptive compression strategies is shown in Figure 12.

Here, we analyze the mix between compressed and uncompressed packets in Figure 7. In contrast to the complex behavior of Simple Switching (Figure 3) due to oscillation, in Figure 7 we see a steady rise of compressed packets up to Max-Compression-Bandwidth at 44Mbps and then all the excess bandwidth being occupied by uncompressed packets. In the variable-bottleneck zone, there is a slight downward slope in the total compressed packets due to the increasing need for CPU used by the network protocol layers to send a large amount of data. Figure 7 reflects the definition of Estimated Maximum (1), except for the downward slope, a detail omitted by Estimated Maximum.

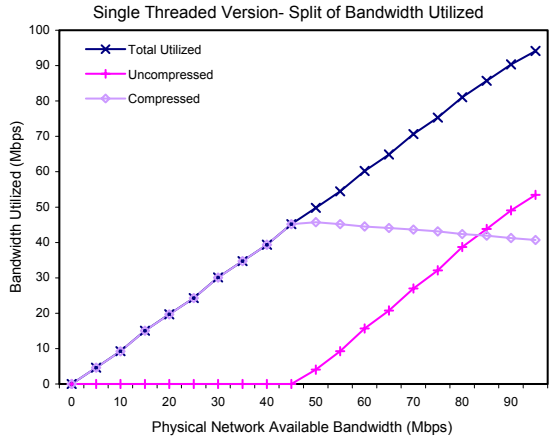

Figure 7 Packet Mix, Single Thread FG Mixing

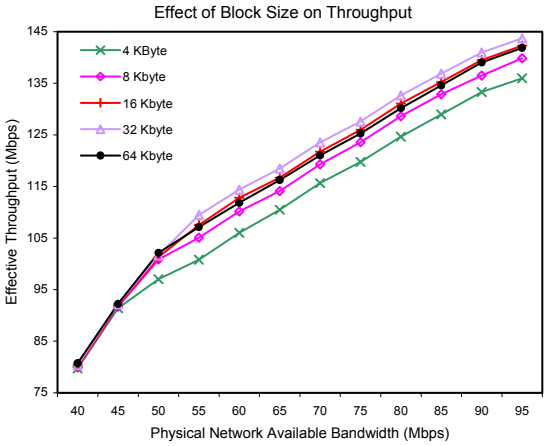

Figure 8 Sensitivity to Block Size

5. Sensitivity to Algorithm Settings and Competition for Resources

One of the important findings in our research is the interference between system components as well as the importance of selecting the right algorithm parameters for achieving the best performance. As we try to optimize resource utilization, the interactions of our algorithms with other system components, make the overall resource optimization of CPU and network bandwidth non-trivial and often non-obvious. In the subsequent experiments, we concentrate on the single-thread implementation, as it performs the best in most cases. FG-Mixing in the subsequent sections refers to the single-thread implementation described in Section 4.3 unless explicitly stated otherwise.

5.1. Effect of Block Size on Throughput

The first system parameter of non-trivial influence is the block size in the buffer being used for compression. In our implementation, each packet corresponds to a block. As we see in Figure 8, the throughput of FG Mixing increases monotonically with the block size from 4KB to 32KB, but it drops at 64KB. This apparent inconsistency is due to compression efficiency fluctuations caused by interactions within the implementation of the compression algorithm.

For the LZ class of compression algorithms, increasing the input size generally results in better compression ratio. This also leads to lower per byte compression times as input size increases. This observation is borne out in Figure 8. But in the case of 64 KB, we see an uncharacteristic decrease in

throughput. Upon further analysis, we found that the GZIP library in use (v.1.2.4) had an internal, temporary buffer of 32 KB, which makes 32 KB the most efficient block size for this implementation of GZIP.

This is an example of non-trivial interactions between system components, in this case the mismatch between the input size and internal buffer size of the compression algorithm.

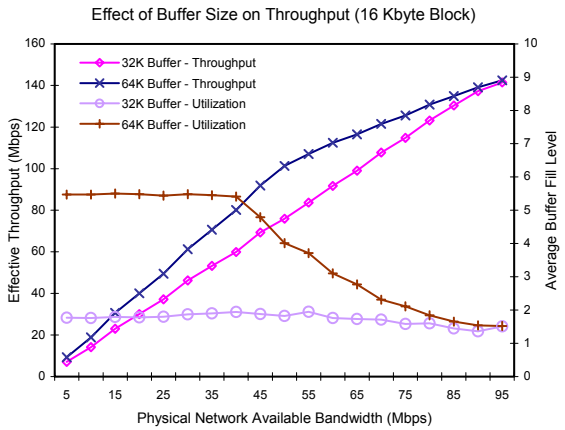

Figure 9 Sensitivity to Buffer Size

5.2. Effect of Buffer Size on Throughput

Related to the block size issue is the available buffer size. We found that a larger buffer is needed in the absolute-network-bottleneck zone to accommodate the storage of more packets when the network becomes a bottleneck. As the network becomes faster, the need for large buffer decreases.

For clarity we show only two buffer sizes to illustrate this trend in Figure 9. For a fixed block size of 16KB, we measured the performance results with 32KB buffer and 64KB buffer. In addition to the effective throughput shown on the left side, we added the buffer fill level scale on the right to show the inverse correlation between network bandwidth and buffer size. In our experiments, buffer utilization is the average of the buffer fill level, measured at intervals of approximately 100 milliseconds.

With a relatively small buffer (32KB, twice the block size), the effective buffer size is limited to two blocks. This is due to the need to reserve half of the buffer (one block) at the beginning of each compression cycle, in case the compression algorithm fails to compress the block. This limited buffer size reduces the effective throughput as shown in Figure 9.

With a larger buffer (64KB), we see a super-linear growth in effective buffer size due to successful compression. As each block is compressed, the remaining room in the buffer can accommodate more packets, since the "last block effect" only affects one quarter of the total buffer. Clearly, a larger buffer is able to cope better with bandwidth bottlenecks in the absolute-network-bottleneck zone. Once we move into the variable-bottleneck zone, the network starts to consume packets faster than the compression algorithm can produce. The result is a lower need for buffering.

The issue of network protocol buffering is non-trivial. There is evidence that a significant portion of TCP latency is due to the send-buffer size [7], where a larger buffer introduces longer latency. Our results showing a decreasing need for buffering could support adaptive send-buffer adjustments as advocated in [7]. Although in this paper we focus on bandwidth, the interactions between different quality of service dimensions such as bandwidth and latency are interesting topics of future research.

5.3. FG Mixing under Competition

Up to now we have been studying adaptive compression under light or no competing load. One of the major applications of adaptive compression is good resource utilization despite environmental changes. In Figure 10 we see the results of injecting a competing network load that occupies 50% of the network bandwidth (about 45Mbps in the 100Mbps Ethernet setup), which effectively shifts the performance of the algorithm to the left of the previous graphs by that margin. As expected, Non-Compressed loses the entire 45Mbps in throughput, All-Compressed loses only a little (the experiment was in the variable-bottleneck-zone). FG Mixing loses the uncompressed packets, but keeps the compressed packet flowing, still gaining a little over the All-Compressed.

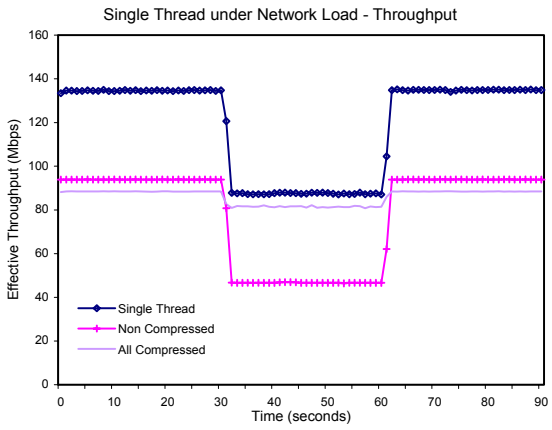

Figure 10 Competing Network Load

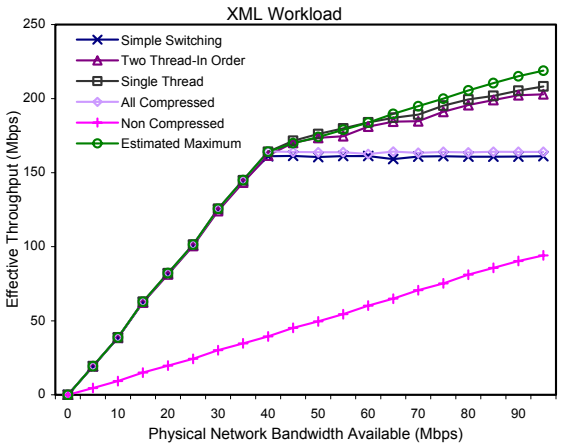

Figure 11 XML Workload

6. Robustness of Fine-Grain Mixing

6.1. Different Workloads

We tested our implementations on other workloads including commercial data in XML format, matrices from a fluid dynamics application, tar distributions and binaries with different compression ratios. As an illustrative example, Figure 11 shows the results of our experiments with the XML workload, which had an average compression ratio of 4.2, the highest ratio amongst the tested workloads. The graph maintains its trapezoid shape, with the rising leg at a steeper angle, showing a bigger gain due to the higher compression ratio. The results of other workloads (of compression ratio between 1 and 4) show the same trapezoid with expected rising legs, so they are omitted here.

6.2. Different Compression Algorithms

Since all previous experiments have used GZIP, as summarized in Figure 12, a natural question is whether the results would be the same with other compression algorithms. To test this hypothesis, we ran the same experiments with other representative compression algorithms. BZIP2 is an algorithm that runs significantly slower than GZIP, but that produces a higher compression ratio. Figure 13 shows that for a slower compression algorithm, the trapezoid is shifted to the left. BZIP2 has a Max-Compression-Bandwidth of about 10Mbps instead of 44Mbps, with the same performance gains, losses, and crossover points, but at a lower point of available network bandwidth.

In contrast, Figure 14 shows the results of using a fast compression algorithm (LZO), which consumes little CPU and offers a relatively low compression ratio. In this graph, the trapezoid is elongated towards the right side. LZO has a Max-Compression-Bandwidth of about 170Mbps, with the same performance gains, losses, and crossover points, but at a higher point of available network bandwidth.

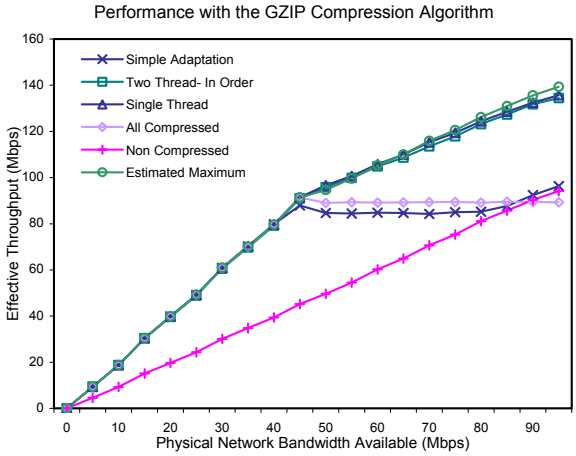

Figure 12 Summary of GZIP Compression

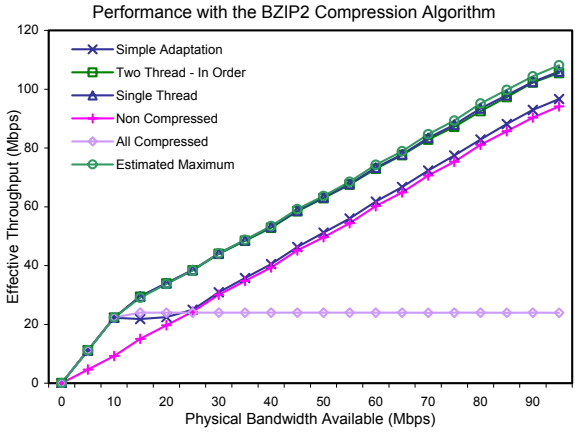

Figure 13 Comparison with BZIP2

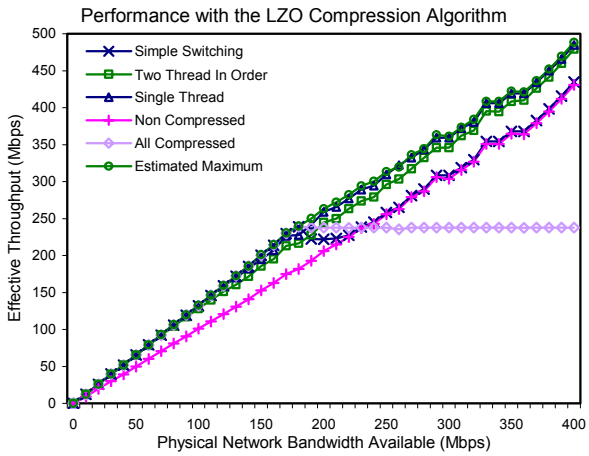

Figure 14 LZO on Gigabit Network

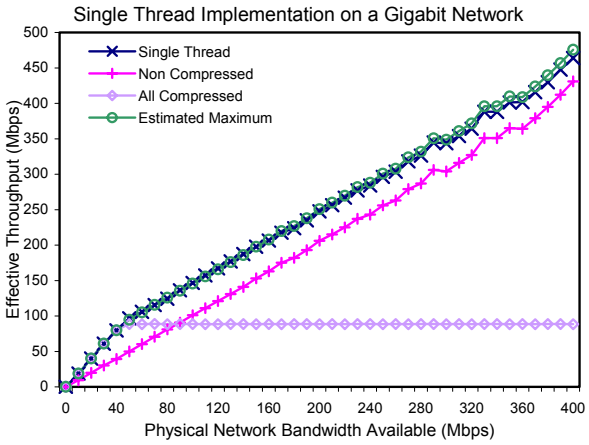

Figure 15 FG Adaptation on Gigabit Network

6.3. Faster Networks

To evaluate FG Mixing in a faster network, we replaced the 100Mbps Ethernet with a Gigabit network. The results have been seen in Figure 6, comparing the send-in-order variant with the send-out-of-order variant, and Figure 14, studying a fast compression algorithm. Figure 15 extends Figure 12 into Gigabit networks, showing the same trapezoid shape moved towards left as we increase the network bandwidth while holding the node configuration constant.

6.4. Faster CPUs

Since the trade-offs investigated in this paper are CPU and network bandwidth, we see that a faster network moves the trapezoid of interest towards left (expanding the variable-bottleneck-zone). Conversely, a faster CPU moves the trapezoid of interest towards right (expanding the absolute-network-bottleneck-zone). This is also suggested by Figure 14, where a fast compression algorithm effectively "speeds up" the CPU performance. Consequently, we omit the experiments on faster CPUs.

7. Related Work

Adaptive compression is a natural solution for networks with variable bandwidths, e.g., shared environments and heterogeneous networks such as those discussed by Douglis [5], Katz and Brewer [11] , and Mogul et al [14]. There are several ways to adapt to different situations. An interesting dimension of adaptive compression research is finding and using the best compression algorithm for each combination of data, system and network conditions. Examples include Dynamic Compression Format Selection [13] for transporting Java class files, Network Conscious Text Compression system [15] for text data and Wiseman et al. [19] for a more general dataset. Our work complements this line of research, since we focus on the trade-offs and system parameter settings for each compression algorithm.

Some adaptive compression systems adopt strategies similar to one of ours. For example, Adaptive Compression Environment (ACE) [18] adopts a strategy similar to Simple Switching. ACE uses Network Weather Service to predict CPU and network availability and identify when compression will provide better throughput. Similarly, Hu et al [9] use Remos, a network performance middleware service, to provide information about current resource usage for switching. In contrast, our systems are self-contained (with an internal probing mechanism). Another example is Adaptive Online Data Compression (AdOC) [10], which uses a multi-threaded approach, one to compress data and one to send data, to maximize effective throughput. In contrast, our fine-grained adaptation approaches achieve full utilization of bandwidth and CPU.

More powerful adaptive compression techniques can be applied if applications (e.g., multimedia streaming) can use lossy compression algorithms that typically achieve much higher compression ratio. For example, On-Demand Dynamic Distillation method [6] uses a proxy-based approach to tailoring content for clients, which is useful when rich text, images and video are transmitted to hardware constrained clients. Other papers (e.g., Badrinath et al [2] and Knutsson et al [12]) describe some of the adaptation systems used in mobile networks, such as how server-directed transcoding can be used to overcome client limitations. These application-specific approaches are also orthogonal and complementary to our research.

Although much of the previous adaptive compression work focuses on increasing effective throughput, there have been research projects that address more than one dimension, for example, improving both throughput and energy consumption in mobile environments. Aleh et al [1] study the effect of compression in lowering costs in packet switching networks. Since transmission costs in wireless networks are generally higher than computation costs, it could be advantageous to compress and send data as shown by Barr et al [3] and Poellabauer et al [16]. While our work is complementary to energy conservation, the

pursuit of multi-dimensional optimization is similar to both lines of research.

8. Conclusion

In this paper, we describe an experimental evaluation of Fine-Grain (FG) Mixing of compressed and uncompressed packets for adaptive compression in networks with dynamically variable bandwidths. The main goal of FG Mixing is to fully utilize all available system resources fairly, in this case CPU and network bandwidth. We use all available CPU to compress as much data as possible. We then use all available network bandwidth by initially sending the compressed data and then using the remaining bandwidth to transmit as many uncompressed packets as possible.

In our evaluation of FG Mixing, which is implemented in the middleware level, we found that apparently innocuous design and implementation decisions may end up with significant impact on system performance. Despite these differences, we found the trapezoid performance characterization to be consistent across different compression algorithms (GZIP, LZO and BZIP2) Similar consistency was found over a range of workloads in terms of compression ratio. We also found the trapezoidal nature repeated in Gigabit networks with the trapezoid having short legs and long bases.

In summary, with careful tuning and synchronization of system parameters, FG Mixing can achieve significant gains in effective throughput in variable bandwidth network environments.

9. References

[1] A. Aleh and Levin, K.D. "The determination of upper bounds for economically effective compression in packet switching networks". In *Proceedings of the Computer Network Performance Symposium*, pp 64-72. ACM Press, 1982.

[2] B. Badrinath, Armando Fox, Leonard Kleinrock, Gerald Popek, Peter Reiher and M. Satyanarayanan. "A conceptual framework for network and client adaptation". In *Mobile Network Applications*. Vol. 5(4), pp 221-231, 2000.

[3] K. Barr and K. Asanovic. "Energy aware lossless data compression". In *First International Conference on Mobile Systems, Applications, and Services*, May 2003.

[4] L. P. Deutsch. RFC 1952: GZIP file format specification version 4.3, May 1996. ftp://ftp.internic.net/rfc/rfc1952.txt.

[5] Fred Douglis. "On the role of compression in distributed systems". In *Proc. of the 5th workshop on ACM SIGOPS European workshop*, pages 1-6. ACM Press, 1992.

[6] Fox, A, Brewer, E, Gribble, S, and Amir, E. "Adapting to Network and Client Variability via On-Demand Dynamic Transcoding". In *Proceedings of Seventh Intl. Conf. on Arch. Support for Programming Languages and Operating Systems*. , Cambridge, MA, 1996.

[7] Goel, C. Krasic, K. Li, J. Walpole. "Supporting Low Latency TCP-Based Media Streams". In *Proceedings of the Tenth International Workshop on Quality of Service*, Miami, Florida, May 2002.

[8] Jean-loup Gailly and Mark Adler. gzip 1.2.4. ftp://wuarchive.wustl.edu/mirrors/gnu/gzip/gzip-1.2.4.tar.gz

[9] Ningning Hu. "Network Aware Data Transmission with Compression". *In Proceedings of The 4th Annual CMU Symposium on Computer Systems*, October 2001.

[10] E. Jeannot, B. Knutsson, and M. Björkman. "Adaptive online data compression". In *IEEE High Performance Distributed Computing*, Edinburgh, Scotland, July 2002.

[11] R. H. Katz and E. A. Brewer. "The Case for Wireless Overlay Networks". In *Proc. SPIE Multimedia and Networking Conf.* pp 77-88, San Jose, CA, Jan. 1996.

[12] B. Knutsson, H. Lu, J. Mogul, B. Hopkins. "Architecture and performance of server-directed transcoding". In *ACM Transactions on Internet Technology (TOIT),* Volume 3(4), pp 392-424, Nov. 2003.

[13] C. Krintz and B. Calder. "Dynamic selection of compression formats to reduce transfer delay". In *10th IEEE High Performance Distributed Computing*, August 2001.

[14] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. "Potential benefits of delta encoding and data compression for http". In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 181-194. ACM Press, 1997.

[15] N. Motgi and A. Mukhargee. "Network conscious text compression system (NCTCSys)". In *Proc. of International Conference on Information Technology: Coding and Computing, 2001*, pp 440-446, April 2001.

[16] Christian Poellabauer and Karsten Schwan. "Energy-aware media transcoding in wireless systems". In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications,* March 2004.

[17] Neil T. Spring and David Wetherall. "A protocol-independent technique for eliminating redundant network traffic". In *Proceedings of ACM SIGCOMM*, August 2000.

[18] Sezgin Sucu and Chandra Krintz. "ACE: A resource-aware adaptive compression environment". In *International Conference on Information Technology: Coding and Computing (ITCC03)*, April 2003.

[19] Yair Wiseman, Karsten Schwan, and Patrick Widener. "Efficient end-to-end data exchange using configurable compression". In *24th ICDCS*, Tokyo, March 2004.