Energy and Performance Evaluation of Lossless File Data Compression on Computer Systems

A Thesis Presented
by

Rachita Kothiyal

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

Technical Report FSL-09-02
May 2009

Abstract of the Thesis

Energy and Performance Evaluation of Lossless File Data Compression on Computer Systems

by

Rachita Kothiyal

Master of Science

in

Computer Science

Stony Brook University

2009

Data compression has been claimed to be an attractive solution to save energy consumption in high-end servers and data centers. However, there has not been a study to explore this. In this thesis, we present a comprehensive evaluation of energy consumption for various file compression techniques implemented in software. We apply various compression tools available on Linux to a variety of data files, and we try them on server, workstation and laptop class systems. We compare their energy and performance results against raw reads and writes. Our results reveal that software based data compression cannot be considered as a universal solution to reduce energy consumption. Various factors like the type of the data file, the compression tool being used, the read-to-write ratio of the workload, and the hardware configuration of the system impact the efficacy of this technique. We found that in some cases, compression can save as much as 33% energy and improve performance by 37.85%. However, in other cases we found that compression can *increase* energy consumption 7 times and *deteriorate* performance 4 fold.

To my parents and my sisters, Ruchi and Rachna.

Contents

List of Figures **vii**

List of Tables **viii**

Acknowledgments **ix**

1 Introduction **1**

2 Background and Related Work **3**
- 2.1 Power Management Approaches 3
- 2.2 Energy Saving Techniques for Storage 4
- 2.3 Saving Energy Using Compression 4

3 Metrics **6**

4 Experimental Methodology **8**
- 4.1 Experimental Setup 8
- 4.2 File types and Compression Tools 9
- 4.3 Benchmarks 10
- 4.4 Read-Write Model 11

5 Evaluation **14**
- 5.1 Terminology 14
- 5.2 PPMd Analysis 15
- 5.3 Server Results 15
  - 5.3.1 Text File Analysis 15
  - 5.3.2 Binary File Analysis 20
  - 5.3.3 Random File Analysis 21
  - 5.3.4 Zero File Analysis 21
- 5.4 Desktop Results 23
  - 5.4.1 Text File Analysis 23
  - 5.4.2 Binary File Analysis 24
  - 5.4.3 Random and Zero File Analyses 26
- 5.5 Laptop Results 26

5.5.1 Text file Analysis 27
5.5.2 Binary file Analysis 29
5.5.3 Random and Zero file Analyses 29
5.6 Summary of Evaluation 31

6 Conclusions **33**

7 Future Work **34**

Bibliography **35**

List of Figures

3.1 Generic model of a system 7

5.1 PPMd results for the server 16
5.2 PPMd results for the desktop 17
5.3 Text file results for the server 18
5.4 Break-even read/write ratio for text files on a server 19
5.5 Binary file results for the server 20
5.6 Random file results for the server 21
5.7 Zero file results for the server 22
5.8 Text file results for the desktop 24
5.9 Binary file results for the desktop 25
5.10 Random file results for the desktop 26
5.11 Zero file results for the desktop 27
5.12 Text file results for laptop 28
5.13 Binary file results for laptop 29
5.14 Random file results for the laptop 30
5.15 Zero file results for laptop 31

List of Tables

3.1 Performance and Energy Metrics 7

4.1 Testbed Setup 9
4.2 Various invocations of the compression tools 11
4.3 Compression ratios 12

5.1 Break even read/write ratio for the server 23
5.2 Break even read/write ratio for the server 25
5.3 Break even read/write ratio for the server 30

Acknowledgments

My immense gratitude to my advisor, Dr. Erez Zadok, for his constant guidance and motivation. I thank Dr. Samir Das and Dr. Jennifer Wong for being on my defense committee, and providing valuable suggestions. I am thankful to everyone at the File Systems and Storage Lab (FSL), for making it a great learning experience. I also want to specially thank Vasily Tarasov and Priya Sehgal, for their help through the project.

This thesis was made possible in part thanks to a 2008 IBM Faculty Award and a 2009 NetApp award. The thesis is also sponsored in part by the Stony Brook Advanced Energy Research and Technology Center (AERTC, `www.aertc.org`).

Chapter 1

Introduction

Until recently, power management research was mostly directed towards battery powered portable computers and mobile devices [4, 32, 37, 38, 49, 55, 56]. The motivation behind these efforts has been to enhance user satisfaction by reducing the frequency of battery recharges. However, the growing costs of power and cooling have now caused researchers to look at the same issue on desktops and commercial servers [10, 11, 17, 25, 27, 40, 47, 65]. Data centers and servers primarily deal with data. Data compression has been suggested an effective way of saving energy in such systems. To the best of our knowledge, there has not been a study evaluating these claims.

In this thesis, we study several compression algorithms, implemented in software, applied to various types of data files, on three different classes of machines, and evaluate all in terms of performance and energy metrics. We use four different types of files for our experiments: *zero*, *text*, *binary*, and *random*. These file types exhibit different levels of data redundancy, with zero being the highest and random being the lowest. Our benchmarks include five popular compression utilities on Linux: gzip, lzop, bzip2, compress, and ppmd. File compression is known to be computationally intensive, but can reduce the amount of I/O being incurred due to a reduction in file size. The aim of this study is to evaluate each of the compression tools, and determine if the savings due to reduced I/O (both in time and energy) are worth the added overhead at the CPU and memory. To be able to view the effects of compression/decompression on energy and performance simultaneously, we use the energy-delay product metric [22] for our analysis.

Our results reveal that software based data compression cannot be considered a universal solution to reduce energy consumption in computer systems; it greatly depends on the type of data files being compressed, the compression algorithm applied, the workload of the system, and the hardware configuration. As we expected, compressing zero files was found to almost always save energy, compared to raw reads and writes, no matter what compression algorithm was used. We realize that such high levels of redundancy are not common in real-life settings, but we include it in our study to evaluate the best-case scenarios. Second to zero files, we observed that text files exhibited the most potential for energy savings by compression, followed by binary files. Although some utilities always performed better than plain writes and reads for text files, other tools required some number of reads for every write to result in energy savings. This is because compression typically consumes more CPU than decompression. To represent the possible savings in such cases, we developed a simple read-write model: it calculates the minimum number of decompressions required to offset the extra energy expended by a single compression. This number can be useful in deciding whether or not a workload whose read-to-write ratio is known would benefit from compressing its data files using a particular compression tool. Finally, also as expected, random files showed no energy or performance benefits upon compression. Again, we included random files to be able to evaluate the worst-case scenarios for compression.

The rest of the thesis is organized as follows. Chapter 2 provides some background and discusses related work in the area. In Chapter 3 we talk about the various metrics used for evaluating our results. We describe the details of the experimental methodology and present the read-write model of evaluation in Chapter 4. We present the actual experimental results obtained from the various benchmarks in Chapter 5, and we conclude in Chapter 6.

Chapter 2

Background and Related Work

Section 2.1 begins with an overview of some existing techniques for power management in computing systems. In Section 2.2, we present various power management solutions for primary storage media. In Section 2.3, we address compression techniques implemented at various levels and their energy impact. We also draw out important distinctions between our work and other research in this area.

2.1 Power Management Approaches

Energy management techniques can be implemented at several levels in a computer system. The fundamental idea behind these approaches has been to transition a component to a lower power mode or to turn it off completely when not in use. Lorch et al. discuss software techniques to utilize the power saving provisions provided by the various hardware components, such as the CPUs, disks, displays, wireless communication devices, main memory, etc. [37]. Dynamic Voltage and Frequency Scaling (DVFS) techniques have been widely employed for reducing CPU power consumption [10, 55, 56]. DVFS allows processors to dynamically switch to different operating voltages and frequencies. Choosing a lower voltage would translate to a reduction in power consumption. However, since voltage cannot be changed independent of the frequency, it would also result in some degree of performance degradation. Several processors support *Clock Gating* as a means to halt idle components, and save power [14, 21, 24, 46].

Su et al. proposed and evaluated several CPU cache designs based on *Gray codes* and cache organization [49]. As Gray codes require only one bit modification to represent consecutive numbers, Su et al. were able to obtain significant energy savings because of reduced bit switching. They also found that cache sub-banking [50] (i.e., organizing cache into banks), was an effective way to reduce energy consumption of caches. Power Aware Page Allocation [32] reduces the memory energy consumption by adding energy awareness to the operating system's virtual memory page allocator. The authors explored various page allocation policies to harness the power management features of emerging DRAM devices.

The OS has also been used to monitor the usage of hardware resources, in order to transition the components to low power states during periods of inactivity [4, 18, 38]. Zeng et al. propose an Energy-Centric Operating System (ECOSystem), which allows energy to be managed as a first-class resource by the OS [64].

2.2 Energy Saving Techniques for Storage

One of the earliest ideas for energy conservation in disks was to spin them down when idle. The controls on when to spin them down have ranged from simple threshold-based policies to intelligent prediction policies [15, 16, 35, 58]. Techniques such as Massive Array of Idle Disks (MAID) [11], Popular Data Concentration (PDC) [43], and write off-loading [40] are based on the idea of directing the requests to a subset of the disks or special logging devices. This increases the idle time between requests, hence justifying the spin down of the unused disks. GreenFS [27], is a stackable file system for client systems. It services I/O requests from remote servers in addition to adding a flash layer to the storage hierarchy. In enterprise settings, with existing backup server infrastructure already in place, the energy cost of network transfers for small transfers is much smaller than spinning up and writing to the local disk. This allows the hard disks to be powered down for longer, and hence save more energy. Many vendors, (e.g., NetApp, EMC, etc.,) provide a large NVRAM to cache disk writes.

Analogous to DVFS for CPUs, Gurumurthi et al. [47] proposed disks which can dynamically change their rotation speeds based on the request traffic, thereby lowering their power consumption. Zhu et al. [65] considered storage cache replacement techniques to selectively keep some blocks from the disk in the main memory cache, to increase the disk's idle times; this allows disks to remain in low power mode for longer.

Another approach taken by many researchers, distinct from the disk spin-down policy, has been to reduce the energy consumed by head seek operations. Essary et al. present a Predictive Data Grouping technique [17] which attempts to co-locate related data blocks on disk through remapping and replication. Huang et al. proposed a file system, FS2 [25] which dynamically replicates data so that the nearest copy of the data can be served on a request. As the mechanical movement of the disk head is reduced by these techniques, it results in power savings. Interestingly, the increased proximity of the data to the disk head also reduces the seek and rotational delays, which translates to better performance.

2.3 Saving Energy Using Compression

Compression has been widely used to reduce traffic and latencies on communication channels (Data bus, network, etc.) [6, 9, 26, 31, 59], and save storage space [2, 44]. Over the last decade, compression has been implemented at various levels of the memory hierarchy and proved to be a successful method of saving energy. For example, several encoding schemes have been proposed for compressing the contents of the CPU instruction cache [7, 8, 33, 62]. These techniques, called *code compression*, map the program instructions into a set of much shorter instructions, thereby reducing the memory requirements and bus traffic. A decompressor, typically between the cache and the CPU, translates the compressed instructions to the normal program instructions before execution on the CPU. Various compression algorithms have been employed on CPU data caches as well [29, 30, 51, 53].

Benini et al. propose a hardware implementation of the compression-decompression logic between the main memory and the CPU cache for embedded processor systems [5]. On a cache write-back, compressed data is written to main memory, while decompressed data is written from main memory to the cache. IBM's Memory Expansion Technology (MXT) [52] has made main memory data compression commercially available to a wide range of systems. Kandemir et al. extend compression to multi-bank memory systems, by compressing infrequently used data, and transitioning those banks to lower power mode after a threshold idle time [28]. Sadler et al. employ lossless compression on data communication in sensor networks to reduce energy expenditure [48].

The work most closely related to ours, albeit in a different environment of embedded and mobile devices, is that of Barr et al. [3]. Because the energy cost of a single bit wireless transmission is many times that of a single 32-bit computation in a handheld computer, they apply lossless compression techniques to data before transmitting. In their work, Barr et al. analyze various data compression algorithms from the energy perspective. Unlike their work, our study is focused on file compression and on server, desktop and laptop systems. Our goal, is to investigate potential energy savings in the storage stack, rather than from transmission over a network. Typically on the systems of our interest, the energy cost of computation is much higher than performing I/O to the storage. While Barr et al. found `lzop` and `compress` to be the most energy efficient, we found only `lzop` to be widely applicable in our test environments.

Another related work by Xu et al. [61] explores data compression as a means to reduce battery consumption of hand-held devices when downloading data from proxy servers over a wireless LAN. They assume that the data from the proxy server is available in compressed format and hence focus their study only on the energy costs related to decompression. Our target systems differ from theirs in that our systems would have to incur the costs of both reads and writes. Hence, we take into account the energy costs of both compression and decompression in our analysis.

Data compression for storage can be implemented at both hardware [13, 41] and software levels. However, in this work we focus our analysis on software implementations only, so as to minimize variations due to hardware changes.

Chapter 3

Metrics

The increasing number of studies in the area of green technologies have revealed a problem of lack of agreement on a proper metric for comparing energy efficiency of computer systems. The choice of an appropriate metric depends on several factors: which component of a system the metric will be applied to, what are the purposes of comparison and how different the systems are. For our study, the metric must be generic enough to express the energy efficiency of the system as a whole. We also would like the metric to be usable for different purposes of comparison. Therefore, we present in this section, not one, but several metrics based on a simple view of a computer system. This family of metrics allows us to describe the energy efficiency profile of a system from several angles, which offers enough scope for a broad analysis.

We define a *system* as any device capable of performing computational work. The work is provided to the system by a user as a list of *tasks*. A task is a logically independent unit of work that the user wants the system to perform. The rational metric representing the performance of such a system is its computational power: the number of tasks the system is able to perform in a unit of time. For the purposes of our discussion, however, it is more convenient to use the inverse value of computational power: the time required to finish a single task. We denote this value as T and measure it in seconds per task. Notice that the notion of a system and a task are highly conceptual here. Depending on the specific scenario, the system can be a CPU that is executing instructions, a disk drive performing I/O requests, a server executing compression algorithms on a piece of data and writing the results to a disk, and more.

While performing computational work, the system consumes electrical energy. In other words, the system converts electrical energy (typically measured in Joules) to computational work (measured in tasks accomplished). In terms of power consumption, we are mostly interested in the effectiveness of this conversion: the number of tasks the system is able to perform by using a unit of energy—or in its inverse form—the energy consumed by the system to perform a single task. We denote the latter value as E and measure it in Joules per task. Figure 3.1 provides the system view we used in our study.

Many projects use the plain metric E to compare energy efficiency of different systems [12, 23, 55, 56]. However, this metric ignores the amount of time it takes to complete a task, T. For example Gonzaleze et al. [22] showed that it is fairly easy to improve a processor's energy efficiency E, but it typically leads to degraded performance of the chip. Sometimes, it is reasonable to ig-

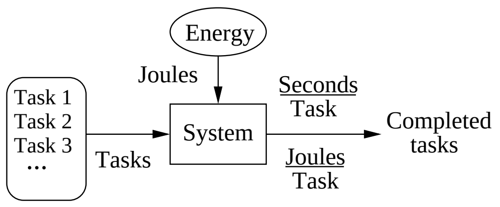


Figure 3.1: System view for energy efficiency estimations.

nore T. For instance, when each system already has the desired performance characteristics [12]. However, in some cases we would like to have a unified metric that gives us a solid understanding of both the system's energy efficiency and its performance. For such a metric we need to take into account both quantities. It is useful to know, for example, how many tasks per Joule per second the system can produce:

$$\frac{Tasks}{Joules \times Seconds}$$

This metric has a clear physical meaning: given its value, one can multiply it by the amount of energy and time available and obtain the number of tasks the system is able to perform under these constraints. The inverse of this metric can be written in the following form

$$\frac{Joules \times Seconds}{Tasks} = \frac{Joules}{Tasks/Seconds} = \frac{Joules}{Throughput}$$

Again, this number has a natural meaning: how many Joules we pay for the speed of execution of a task, as tasks per seconds in the denominator is the throughput of the system. This metric is widely known as *energy-delay* [22, 36, 66]. We denote it as ET.

We believe that omitting any of the metrics represented above (T, E, ET) takes away valuable information about the system. T gives a good understanding of performance, but does not convey power consumption. E provides reliable information about energy efficiency, but ignores the performance. The ET metric has an intuitive underlying physics and is valuable to compare systems in the general case, but is not applicable when one is interested in energy savings or performance only. For these reasons we adopt all three metrics in this thesis. The metrics we used are summarized in Table 3.1. By convention we omit tasks unit from the table, as all units are implicitly per task.

Metric	**Notation**	**Unit**
Time	T	$Seconds$
Energy	E	$Joules$
Energy-delay	ET	$Joules \times Seconds$

Table 3.1: The metrics and corresponding units we used to evaluate performance and energy efficiency of a system.

Another well-known metric of energy efficiency is $energy \times delay^2$ [39]. However, it is specific to the situations where the voltage applied varies from system to system, for instance for comparison of different DVFS levels.

Chapter 4

Experimental Methodology

This section details the setup used for our evaluations. We describe our testbed and the instruments used for energy measurement in Section 4.1. We describe the various benchmarks and the motivation behind their selection in Section 4.3. We present our evaluation model in Section 4.4.

4.1 Experimental Setup

We used three different machines for our experiments. The first was a Dell PowerEdge SC1425 rack-mountable server, with 2 dual-core Intel® Xeon^{TM} CPUs at 2.8GHz, 1GB RAM, 73GB primary hard disk (SCSI SEAGATE ST373207LW, 10000 RPM) and a dedicated 20GB partition on a separate hard disk (SCSI SEAGATE ST373207LW, 10000 RPM) for the tests. The server was running the Fedora Core 6 (kernel 2.6.20-1.2952.fc6) distribution of Linux.

The second machine was a desktop system, with an Intel® Pentium® CPU at 1.7GHz, 1GB RAM, 20GB primary hard disk (WDC WD200BB-00AUA1, 7200 RPM) and a 20GB test partition on a separate disk (Maxtor 6E040L0, 7200 RPM). It was running the same 2.6.20-1.2952.fc6 Linux kernel as the server.

The third machine was an Acer Aspire 5600 Laptop, with Intel Core Duo processor at 1.6GHz, 1GB RAM, 100GB primary disk (Toshiba MK1032GAX, 5400 RPM) of which a 30GB partition was used as the test partition. It was running Ubuntu 8.10 (kernel 2.6.27-9-generic) Linux flavour. In order to simplify our evaluation, we enabled only one processor unit on all these three machines by using the `maxcpus = 1` boot time parameter in Linux. Table 4.1 summarizes the configuration of our testbed.

As our goal is to study the energy impact of data compression on the entire system, and not on a component in isolation, we measure the total energy of the machine. Hence, we used a WattsUP Pro ES [54] power meter to measure the energy consumption of the system under test, instead of a current clamp attached to a digital multimeter [19, 20], which can provide component level energy measurements. The WattsUP Pro ES is a plug-in style power meter, which allows power measurements by plugging in the AC supply of the test machine in the meter's receptacle. It calculates the cumulative energy in Watt-hours (1 Watt-hour = 3,600 Joules) every second, and stores in its non-volatile memory. It has a 1 second time resolution and a 0.1 Watt-hour (360 Joules) resolution for energy measurements; it has an accuracy of $\pm 1.5\% + 3$ counts of the

Specification	Machine type		
	Server	**Desktop**	**Laptop**
CPU model	Intel Xeon	Intel Pentium 4	Intel Core Duo
CPU speed	2.8 GHz	1.7 GHz	1.6 GHz
No. of CPUs	2 dual core	1 single core	1 dual core
CPU DVFS support	No	No	Yes
CPU c-states support	No	No	Yes
L1 cache size	16K	8K	16K
L2 cache size	2M	256K	2M
FSB speed	400 MHz	400 MHz	533 MHz
RAM size (actual)	2048M	1152M	2560M
RAM type	DIMM	RIMM	SODIMM
Disk RPM	10000	7200	5400
Disk transfer rate	320Mbps	133Mbps	100Mbps
Disk cache	8MB	2MB	16MB
Disk spindown on idle	No	No	Yes
Machine age	3 yrs	6 yrs	2.5 yrs
Average Idle Power	218W	91W	17W
SPEC CPU2006 sjeng score	6.89	4.47	8.54

Table 4.1: Hardware specification of the machines comprising the testbed.

displayed value. We used a `wattsup` Linux utility [57] to download the data from the meter over a USB interface to the test machine. For measurements on the laptop system, we powered it through the AC power and removed the battery to ensure that all the power being used was reported.

4.2 File types and Compression Tools

Power consumption in the evaluated systems depends on the effectiveness of compression, which is typically measured by *Compression Ratio* (CR) defined as:

$$CR = \frac{Original filesize}{Compressed filesize}$$

Compression ratio is heavily affected by the type of input data file. Hence, we include the file type as one of the dimensions for our evaluation. In order to have a representative set of possible data files, we chose to run the tests on four types of files of size 2GB each: *zero*, *text*, *binary*, and *random*. These represent files with highly redundant data, regular text files, binaries, and files with highly random data, respectively. These files denote the best-to-worst cases of compression, in order. We chose the file size to be 2GB to ensure that each test ran for a considerable amount of time, thereby reducing the scope of errors and high standard deviations arising out of even slight differences in recorded values across multiple iterations of the test. Also, the 2GB file, being larger than the system RAM (1GB), forces I/O to take place. We created the zero file by writing zeroes to the file. We generated the text file by concatenating source files from the Linux kernel and other open source projects. We created the binary test file by combining object files from the Linux kernel, Linux libraries and other open source executables. We created the random file by reading from `/dev/urandom`.

Another factor influencing the compression of a file is the compression algorithm itself. This constitutes the second dimension for our analysis. We examined five popular compression utilities available on Linux: compress, gzip, lzop [42], bzip2 and ppmd. They have significant differences in implementation and cover a wide range of compression algorithms. Bar and Asanovic discuss these tools and their algorithms in detail [3]. The compress utility, regarded as the oldest, implements the Lempel-Ziv-Welch (LZW) algorithm which is a variant of the LZ78 algorithm. It uses m bits (9–16) to encode the input symbols, and stores the string-to-code mapping in a dictionary. Although based on the same LZ77 algorithm, gzip and lzop differ significantly in their implementation. As lzop was designed with the main goal to improve compression/decompression speed, it tends to be generally faster than gzip. In particular, lzop uses a 16KB hash table, enabling it to be cache resident, thus reducing the frequency of cache misses during its execution. It is also implemented using macros instead of function calls, to reduce performance overheads. The bzip2 utility is based on the Burrows Wheeler Transform (BWT); it achieves better compression ratio than the Lempel-Ziv based tools, at the expense of compression speed. The block size for compression (100k–900k) can be specified at command invocation. A larger block size typically increases the compression ratio, while increasing the memory footprint. PPMd implements the Prediction with Partial Match (PPM) algorithm. PPM is known to produce the best compression ratio compared to all other algorithms. It, however, uses considerably greater time and memory resources.

4.3 Benchmarks

Writing an uncompressed file involves reading the input and writing it to disk. We will refer to this as a *plain-write* in the rest of the thesis. Writing a compressed file involves reading the uncompressed input, compressing it, and writing the compressed file to disk. We shall call this *compress-write*. Similarly, we use the term *plain-read* to denote reading the uncompressed file from the disk; and we use *decompress-read* to indicate reading the compressed file and decompressing it. Each of the operations described above (plain-write, compress-write, plain-read, and decompress-read), constitute a task which we defined in Section 3.

The aim of this study is to compare a *plain-write* to *compress-write*, and a *plain-read* to *decompress-read*, in terms of both energy consumption and performance. We therefore broadly have four types of benchmarks: *plain-write, compress-write, plain-read, and decompress-read*. As mentioned above, we used four different compression tools, each of which can be invoked with tunable parameters. For example, gzip allows the user to specify an *effort* parameter in the range 1–9 to choose between speed of compression and compression ratio; a choice of 1 would result in fast compression, but poorer compression ratio; and a 9 would give the best compression ratio, but would be slower than 1. Table 4.2 lists the various parameter values considered for the compress-write benchmarks. For each of the compression tools we chose the default invocation, and the options which provide the best and worst case of compression speed (if not already covered by the default option). Table 4.3 lists the compression ratios achieved by compressing different types of files using various compression applications.

Invocation	Algorithm	Implications

gzip –1
gzip –6 (default)
gzip –9
	LZ77	max_chain = 32, other parameters
		max_chain = 32, other params
		max_chain = 4096, other params

lzop –1
lzop –3 (default)
lzop –9
	LZ77	max_chain = 4, other params
		nearly identical to lzop-1
		max_chain = 4096, other params

bzip2 –1 bzip2 –9 (default)	BWT	Use 100K block size Use 900K block size (default)
compress –b 10 compress –b 16 (default)	LZW	Use 10 bit codes Use 16 bit codes
ppmd –o 2 ppmd –o 16	PPM	Predict next character based on last 2 seen Predict next character based on last 16 seen

Table 4.2: Parameters used for invocation of various compression tools for compress-write benchmark.

We used the Auto-pilot test suite infrastructure [60] to run the benchmarks. Auto-pilot measures the time required to run a benchmark and reports it in terms of Elapsed, System, User, and Wait times. We developed an Auto-pilot script plug-in to measure the energy consumed while running the benchmark. The plug-in relies on the Linux utility described in Section 4.1 to communicate with the meter. The plug-in uses the utility to send a command to clear the meter's internal memory before starting the benchmark. After the benchmark has finished execution, we invoke the utility to send a command to read the data from the meter, and extract the total energy expended (in Joules) while running the benchmark. Since the benchmark themselves run for a significant time, any energy measurement errors due to the measurement tool itself are negligible.

We ran all tests at least five times and computed the 95% confidence intervals for the mean elapsed, system, user, and wait times using the Student's-t distribution. In each case, unless otherwise noted, the half widths of the intervals were less than 5% of the mean. In all bar graphs, we show the half widths using an error bar. Wait time is elapsed time less system and user time and mostly measures time performing I/O, though it can also be affected by process scheduling.

We ran the tests on a dedicated hard disk, with the partition formatted with the Ext2 file system and mounted using the default options. To ensure that writes to the partition were flushed to the disk during our measurements, we unmounted the partition at the end of each test iteration.

4.4 Read-Write Model

The best case for compression would be when compress-write outperforms plain-write, and decompress-read fares better than plain-read, in terms of a metric. However, there might be scenarios when only one of these comparisons favor compression. For example, for a given compression tool, compress-write might require more energy than plain-write, but expends less energy for a decompress-read than a plain-read. Notice that the metric we consider in this example is energy,

Tool	**File Type**			
	Text	**Binary**	**Rand**	**Zero**
None	1	1	1	1
gz-1	4.16	1.81	0.95	~10^2
gz-6	4.79	1.81	0.95	~10^3
gz-9	4.84	1.81	0.95	~10^3
lzo-1	3.52	1.53	0.95	~10^2
lzo-3	3.51	1.53	0.95	~10^2
lzo-9	4.37	1.81	0.95	~10^2
bz-1	5.09	1.81	0.95	~10^6
bz-9	6.11	2.09	0.95	~10^7
c-10	1.17	1.04	0.8	~10^2
c-16	2.07	1.17	0.8	~10^5
ppmd -o 2	3.86	1.81	0.95	~10^3
ppmd -o 16	7.7	2.44	0.95	~10^3

Table 4.3: Compression ratios achieved by various compression utilities on 2GB files.

but the argument applies to the other metrics as well (e.g., time or energy-delay). Compression might still achieve energy savings in such a case if the number of reads is more than a "break-even" value to amortize the extra energy consumed by a single compress-write.

Workloads are characterized by a *read-to-write* ratio (n), which represents the distribution of read and write I/O requests. There have been extensive studies to characterize workloads based on this parameter [34, 45]. Given a workload, with knowledge about its read-to-write ratio and the type of file data it handles, we can use this break-even value (n_{be}) to decide if compressing the data files would be beneficial. We formalize this by the following model.

For a given metric M, let M$_w$, M$_c$, M$_r$, and M$_d$ be the measured values of M on a plain-write, compress-write, plain-read, and decompress-read, respectively. Let n_{be}^M represent the break-even read-to-write ratio to obtain energy savings. Assuming we first need to write once before reading, the following inequality must hold to compensate the excess energy expended during the write:

$$M_c - M_w \leq n_{be}^M \times (M_r - M_d)$$

Solving for n_{be}^M, we get

$$n_{be}^M \geq \frac{(M_c - M_w)}{(M_r - M_d)}$$

where $M \in \{T, E, ET\}$

We calculate and present the n_{be}^M values for the T, E and ET metrics for the various compression tools and test files in Tables 5.1, 5.2 and 5.3 of Section 5.

We define the energy savings (E_{sav}) for decompress-read and compress-write vs. plain-read and plain-write for a given value of read-to-write ratio, n:

$$E_{sav} = (n \times (E_r - E_d)) + (E_w - E_c)$$

where, E_w, E_c, E_r, and E_d is the energy expended in plain-write, compress-write, plain-read, and decompress-read, respectively.

Note that a negative value of E_{sav} means energy loss. Figure 5.4 presents the values of E_{sav} for n ranging from 0 to 30 for different compression algorithms applied on a text file. The value of n for which E_{sav} becomes zero is the n_{be}^{E}.

Chapter 5

Evaluation

In this chapter, we evaluate the effect of compression and decompression on energy savings and performance, based on the metrics: time (T), energy (E), and energy-delay (ET), as discussed in Chapter 3. Section 5.1 explains the terms we use in this chapter. We present the results of the PPMd compression utility on the server and desktop machines, in Section 5.2. In Section 5.3 we analyze the results of the other four compression utilities (gzip, lzop, bzip2, and compress) on the server class machine. We evaluate the results obtained on the desktop machine in Section 5.4, and those for the laptop in Section 5.5. Finally, we summarize our observations in Section 5.6.

5.1 Terminology

We present the graphs of the PPMd results for the server and desktop systems in Figures 5.1 and 5.2. The x-axis of these plots denote *file_type-level*, where *file_type* is the type of the input data file: Zero, Text, Binary, and Random; *level* is passed as a parameter to ppmd: o2, denoting the use of order of 2 for compression, and o16 denoting the use of order 16. Both these invocations use 256 MB of memory. We plot the Time, Energy, and Energy-Delay metrics for compressing and decompressing using the PPMd tool. For reasons elaborated in Section 5.2, we present the results for PPMd and those for the other four compression tools (gzip, lzop, bzip2, and compress), in separate graphs.

Figures 5.3, 5.5, 5.6, and 5.7 show the metrics plotted for text, binary, random, and zero files respectively for the server machine. These figures evaluate the gzip, lzop, bzip2, and compress utilities. Figures 5.8, 5.9, 5.10, 5.11 and Figures 5.12, 5.13, 5.14, 5.15 are the corresponding graphs for the desktop and laptop systems, respectively. In all these figures, the x-axis denotes *alg-mode-level*, where *alg* is the type of the compression/decompression algorithm: gzip, lzop, bzip2, or compress; *mode* is either Compression or Decompression; *level* is passed as a parameter to the compression/decompression algorithm to control the compression ratio (CR). Similarly, we use the notation *alg-level*, to refer to a given tool operating at a specific compression level.

The time result figures show the total time required to compress-write or decompress-read a 2GB file using the compression utilities discussed above, compared to plain-writes and plain-reads, respectively. The y-axis on this graph denotes the elapsed time, which constitutes of the *system* time, *user* time, and *wait* time.

The second type of metric plotted is energy. These results compare the total energy required in performing a plain-write/plain-read versus a compress-write/decompress-read. On the y-axis we have the total energy, constituting of *active* and *passive* energy. Passive energy is the energy that is consumed by an idle system, for the elapsed period, without any other activity. For calculating the passive energy, we first need to estimate the average power consumption of the idle system. To compute this, we let the system idle ten times for 10 minutes each, recorded the energy consumed, and we verified that the standard deviations were small. We then divided the total energy measured by the duration of the idleness, yielding the average idle power of the system. Passive energy can be obtained by multiplying the average idle power with the elapsed time. Active energy is the extra energy required, apart from the passive counterpart, to complete the required task. In our graphs, we represent energy in units of Kilojoules, where $1KiloJoule = 10^3 Joules$.

The *energy-delay product* (ET) metric, as discussed in Section 3, compares the ET results of compression/ decompression versus pure writes/reads. Similar to the energy results, the total ET also consists of an active and passive component. We have plotted the ET results in units of MegaJoule-seconds, and in KiloJoule-seconds in some cases.

5.2 PPMd Analysis

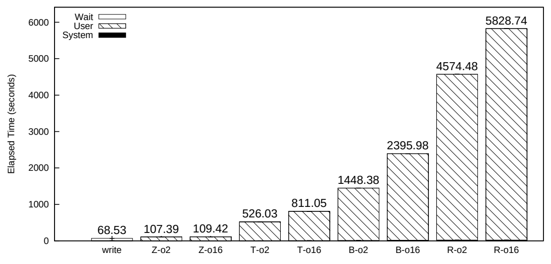
PPMd is based on the Prediction with Partial Match (PPM) algorithm. PPM is known to produce the best compression ratio, at the expense of considerably greater time and memory resources. For example, PPMd yields a compression ratio of 7.7 on a text file (Table 4.3), but consumes about 10 times more time and energy to compress than a plain write. It consumes approximately 30 times more time and energy during decompression as compared to a normal read. Even in the best case scenario of highly redundant data (Zero file), we see PPMd to be worse than plain I/O. Unlike all other compression utilities, which often decompress faster than they compress, PPMd has to perform similar operations during compression as well as decompression. Hence, it is equally slow and energy exhaustive during both compression and decompression of files. As PPMd does not save time or energy during either compression or decompression, for any type of file, it cannot prove to be better than plain I/O. Hence, we do not include it further in our evaluation. In the following sections, we will therefore only evaluate the other four compression utilities (gzip, lzop, bzip2, and compress).

5.3 Server Results

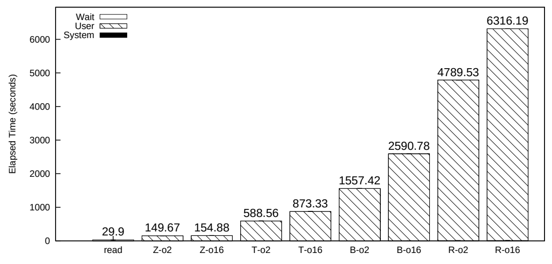
In this section, we present and discuss the results of the benchmarks run on the server class machine. We discuss the results by the file type in the following four sections (5.3.1, 5.3.2, 5.3.3 and 5.3.4).

5.3.1 Text File Analysis

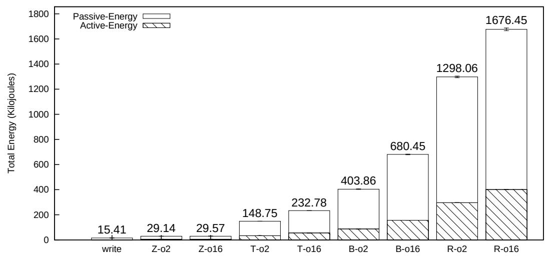
As we observe in Figures 5.3(a) and 5.3(b), the plain-read and plain-write spend most of their time performing I/O. The compress-read and compress-write on the other hand, spend most of their time performing computation on the CPU. They have a significantly smaller portion of time



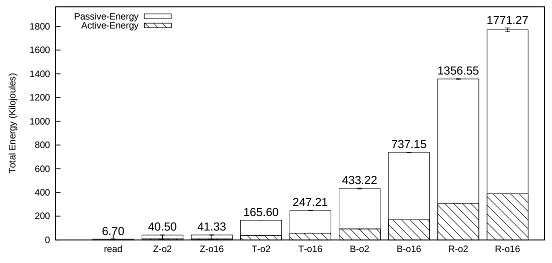
(a) **Time** taken for write vs. compression



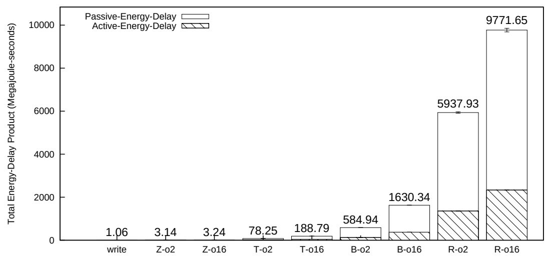
(b) **Time** taken for read vs. decompression



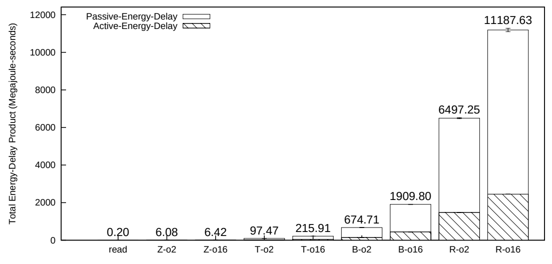
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



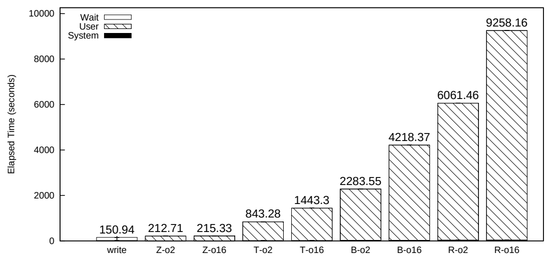
(e) **ET** for write vs. compression



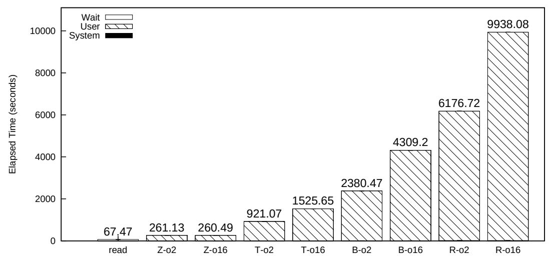
(f) **ET** for read vs. decompression

Figure 5.1: **PPMd results for the server:** The time, energy and energy-delay product (ET) for compressing/decompressing text, binary, random and zero files using PPMd. In (a) and (b), the values along the x-axis are of the form *file_type-level*, where *file_type* is the type of the file: Text, Binary, Random, or Zero; *level* is order2 or order16; Both the levels use 256MB of RAM. The energy results in (c) and (d) represent the total energy (kilojoules) required for compressing/decompressing the various files with PPMd. In (e) and (f) we use Megajoule-seconds to denote the energy-delay product to compress-write/decompress-read the files.

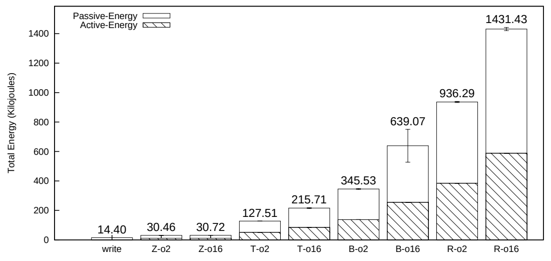
spent on I/O, compared to their plain-read and plain-write counterparts. This is expected, because compression results in a reduction of the file size (as shown in Table 4.3). We also observe that the heights of the compression plots (both time and energy)of a given compression tool, increases with the effort level. This shows that the harder a tool tries to achieve a better compression ratio, the more resources it needs. However, the corresponding heights in the decompression plots of a particular compression tool are only slightly affected by the effort level of compression.



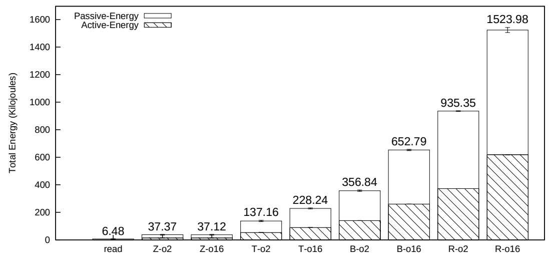
(a) **Time** taken for write vs. compression



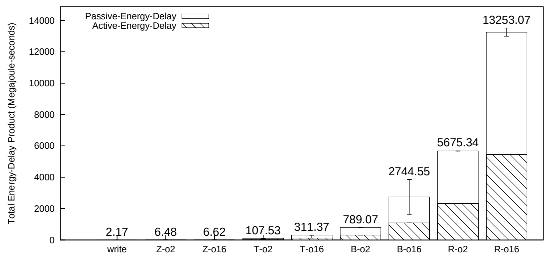
(b) **Time** taken for read vs. decompression



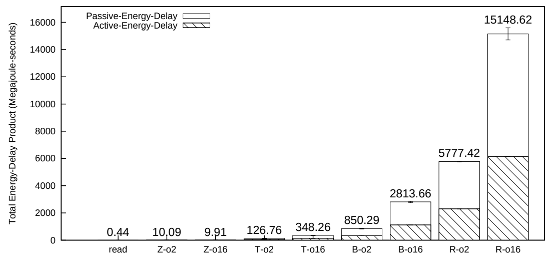
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



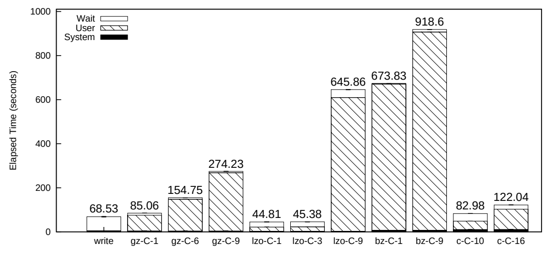
(e) **ET** for write vs. compression



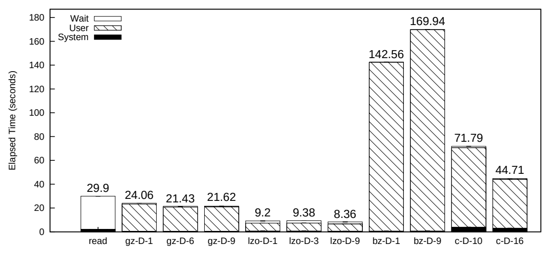
(f) **ET** for read vs. decompression

Figure 5.2: **PPMd results for the desktop**

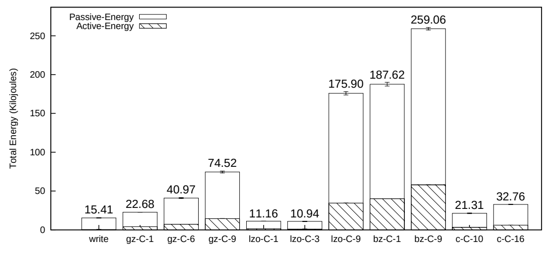
As we observe in Figures 5.3(a), 5.3(b), 5.3(c), and 5.3(d), lzo-1 and lzo-3 always outperform pure writes and reads in terms of both time and energy consumption for text files. lzo-1 and lzo-3 take 34% lesser time to compress than a plain-write, and 69% lesser time to decompress than a plain-read. They save approximately 29% energy compared to plain-writes, and 68% compared to plain-reads. The plots for the ET metric in Figures 5.3(e) and 5.3(f) show that compressing text files using lzo-1 or lzo-3 is beneficial in terms of the ET metric. They exhibit ET values which are lower by 53% than plain-write, and 90% lesser than that for plain-read. The compress utility however does not fair better than plain-write or plain-read by any metric. We observed that although the compress tool achieves a worse compression ratio than lzo, the compress tool takes significantly more time than lzo. Specifically it has a significantly higher system time than the other compression utilities. We ran `strace` and found that compress performs multiple read and write system calls in units of 1024 bytes, instead of a more optimal unit such as 4KB (page frame size), thereby increasing system time. This, in addition to the fact that lzo is designed for speed, is the reason for this behavior.



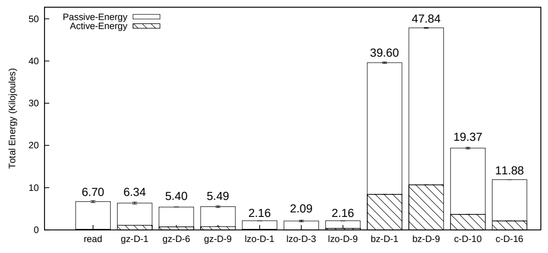
(a) **Time** taken for write vs. compression



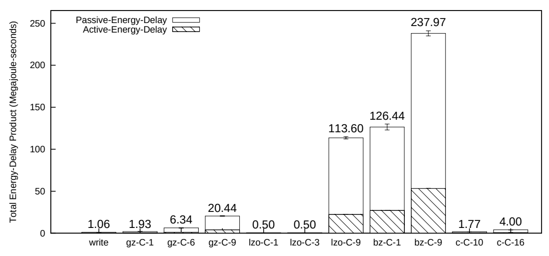
(b) **Time** taken for read vs. decompression



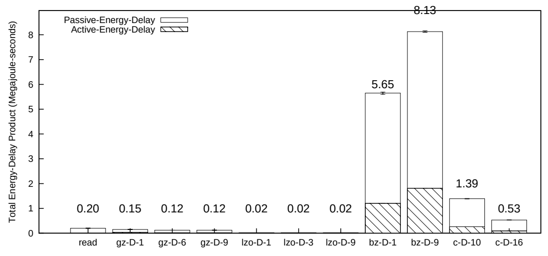
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



(f) **ET** for read vs. decompression

Figure 5.3: **Text file results for server:** The time, energy and energy-delay product (ET) for compressing/decompressing a 2GB text file. In (a) and (b), the values at the x-axis are of the form *alg-mode-level*, where *alg* is the type of the compression/decompression algorithm: gzip, lzop, bzip2, or compress; *mode* is Compression or Decompression; *level* is the parameter passed to the compression/decompression algorithm. The energy results in (c) and (d) represent the total energy (kilojoules) required for compressing/decompressing the file with the corresponding algorithm. Section 5.1 describes the terms *Active-Energy* and *Passive-Energy*. In (e) and (f) we use Megajoule-seconds to denote the energy-delay product to compress-write/decompress-read the same file.

Conversely, gz-6, for example, takes more time and requires more energy to compress-write than plain-write, but it saves during the decompress-read of the same file compared to a plain-read. This implies that compression on text files with gz-6 can still be helpful for workloads which read a file more often than write. As discussed in Section 4.4, we achieve significant energy savings without compromising performance when the read-to-write ratio exceeds the *break-even* value.

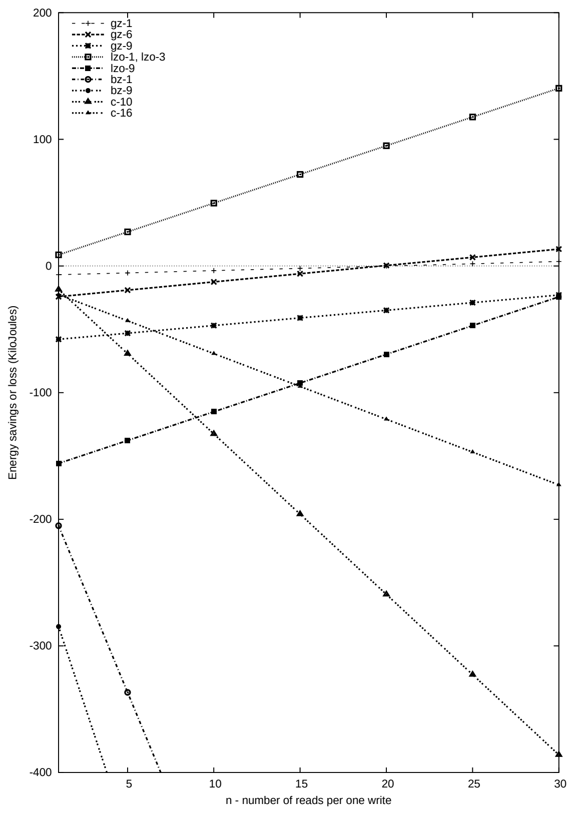
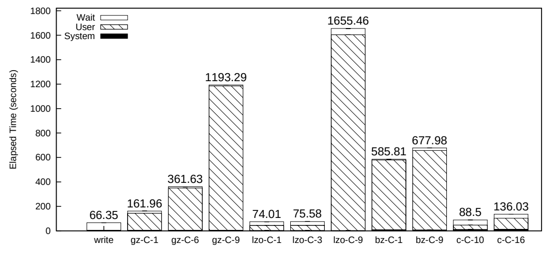
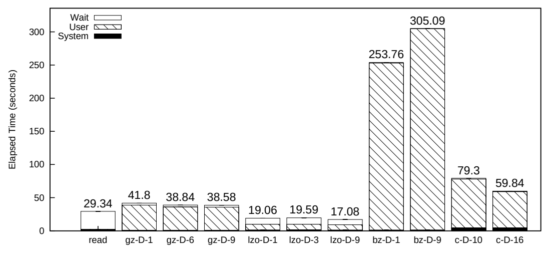


Figure 5.4: Energy savings or loss (E_{sav}) compared to plain read and write depending on the number of reads per one write for text files on the server system. The lzo-1 and lzo-3 results are so close, that we put them on the same line.

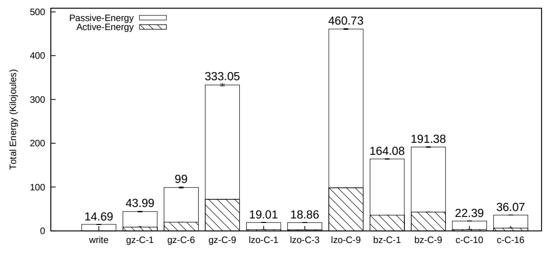
Figure 5.4 demonstrates the dependency of energy savings or loss on the read-to-write ratio on a server system. The y-axis denotes the energy savings, E_{sav}, (in Kilojoules). A positive value of E_{sav} means some energy savings, whereas a negative value denotes energy loss. $E_{sav} = 0$ indicates neither energy savings nor energy losses. The value of n for which $E_{sav} = 0$ is the n^E_{be}. The plots for gz-1, gz-6, gz-9, and lzo-9 cross the $E_{sav} = 0$ line, denoting that there exists a read-to-write ratio when the corresponding algorithm becomes beneficial in terms of energy. For example, gz-1 crosses the $E_{sav} = 0$ line when n is equal to 20.2. This means that if for every write the system experiences 21 or more reads, we can save energy. The lines for lzo-1 and lzo-3 (coincided because of the proximity of results) are always above the $E_{sav} = 0$ line, indicating that these compression tools save energy for any ratio. Conversely, the plots for bz-1, bz-9, c-10, and c-16 never cross the $E_{sav} = 0$ line, implying that these tools expend so much more energy compared to plain-write and plain-read that they are unable to amortize the energy losses for any read-to-write ratio. Both bz-1 and bz-9 consume more time and energy during compress-write as well as decompress-read, because of the algorithmic complexity of bzip2, which uses the Burrows-Wheeler transform, the move-to-front transform, and Huffman coding.



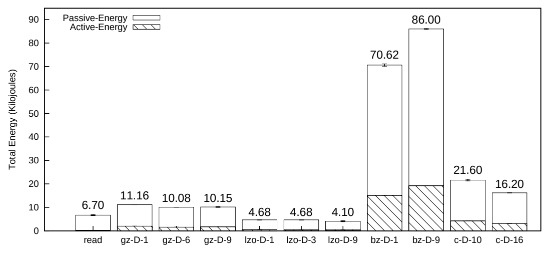
(a) **Time** taken for write vs. compression



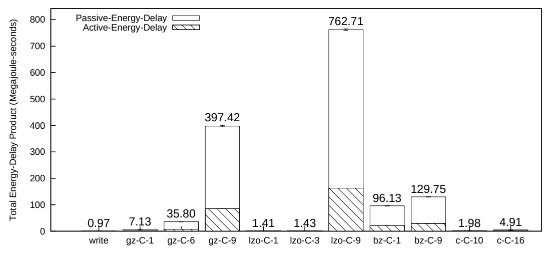
(b) **Time** taken for read vs. decompression



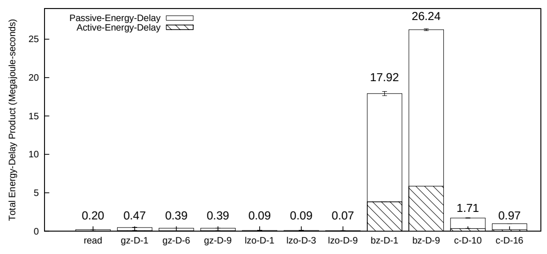
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression

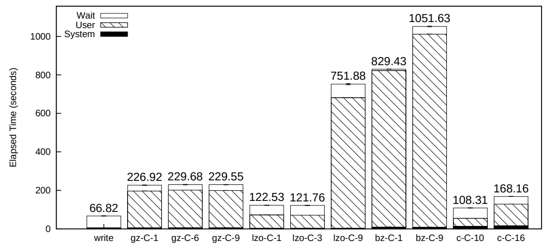


(f) **ET** for read vs. decompression

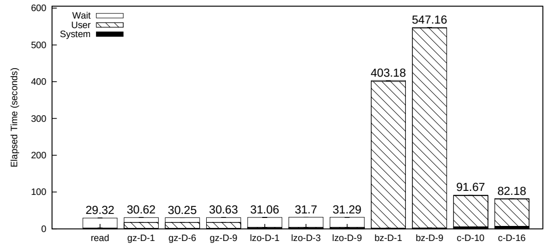
Figure 5.5: **Binary file plots for the server**

5.3.2 Binary File Analysis

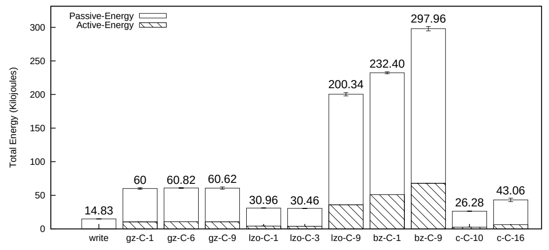
In the case of a 2GB binary file, as shown in Figure 5.5(c), the energy consumption during compress-write using both lzo-1 and lzo-3 is greater than plain-write. Conversely, both of them save energy during decompress-read, seen in Figure 5.5(d). Hence, similar to the discussion in Sections 4.4 and 5.3.1, we compute that lzo-1 and lzo-3 save energy only when $n_{be}^E \geq 2$. However, if we consider the energy-delay metric, shown in Figures 5.5(e) and 5.5(f), the value of the break-even ratio changes to 4 and 4.3, for lzo-1 and lzo-3, respectively. We also see that lzo-9 consumes significantly more energy during compression compared to pure-write, that it is difficult to recoup the over-consumption of energy through multiple decompress-read workloads. This is evident from the large value of n_{be}^E (~10^2) in Table 5.1. All the compression utilities, other than lzop, have a greater energy consumption than plain-writes and plain-reads. Hence, they cannot be considered as good candidates for compression with energy savings in mind.



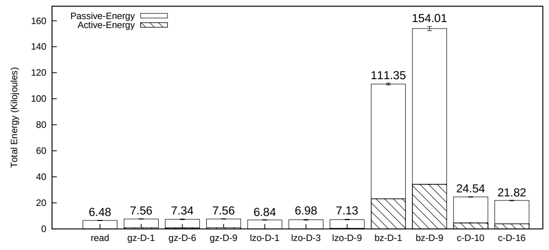
(a) **Time** taken for write vs. compression



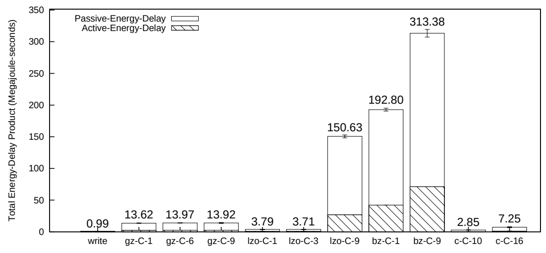
(b) **Time** taken for read vs. decompression



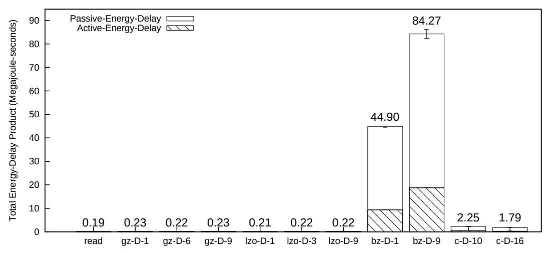
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



(f) **ET** for read vs. decompression

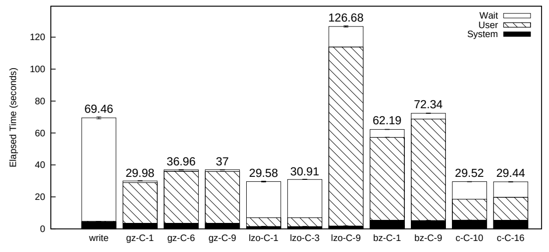
Figure 5.6: **Random file plots for the server**

5.3.3 Random File Analysis

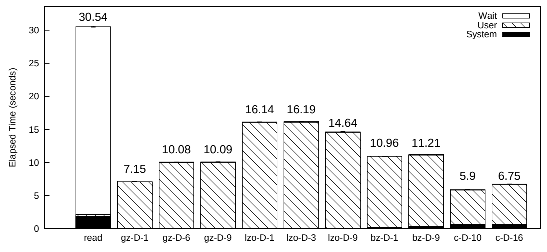
It is evident from Figure 5.6 that no compression utility saves energy during compression or decompression. Consequently, the E values for the compression utilities is also greater than that of plain-read and plain-write. The reason is that compression utilities find it difficult to discover repeated patterns in a random file, which inherently has a high entropy [1]. Therefore, the tools waste a lot of CPU time and energy trying to compress, but do not gain much in terms of CR, as shown in Table 4.3. Hence, modern storage systems should recognize high entropy files, such as multimedia, random, encrypted, etc., and write them directly to the disk without compression.

5.3.4 Zero File Analysis

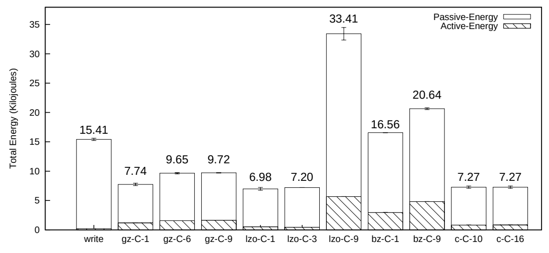
As expected, all the compression utilities, except lzo-9, consume less energy than writes and reads (Figure 5.7). The energy consumption by lzo-9 compress-write is almost twice that of plain write, but it recovers from the energy losses for $n_{be}^{E} \geq 5.4$ (Table 5.1). Considering the ET metric, the



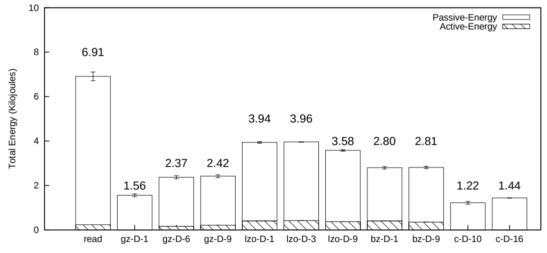
(a) **Time** taken for write vs. compression



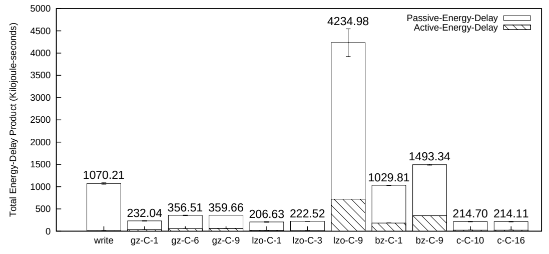
(b) **Time** taken for read vs. decompression



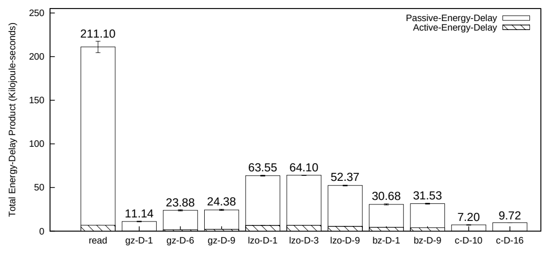
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



(f) **ET** for read vs. decompression

Figure 5.7: **Zero file plots for the server:** The energy-delay product (ET) is in KiloJoules-sec.

break-even ratio for lzo-9 rises to 20. Most of the compression utilities achieve a high CR without increasing the ET because of the large frequency of repeated patterns. From this observation, we can suggest that if a storage system consists of files with a large number of repeated patterns (e.g., log files) compression is definitely a better alternative in terms of saving on energy without performance degradation.

Summary of Server results. Table 5.1 contains the calculations of n_{be}^{T}, n_{be}^{E} and n_{be}^{ET} for the compression tools discussed in this thesis, for the server system. The break-even value varies depending on the compression tools used and their CR values. Often, the higher the CR, the slower the utility operates, consuming more energy, thereby raising the break-even ratio. Although the break-even ratio for some utilities (gz-9, lzo-9, etc.) is greater than the read-to-write ratio on a common server system [34, 45], we can consider them beneficial for a read-intensive workload (e.g., public FTP mirrors). In this case, tools with a higher CR can be applicable, especially if storage space and network traffic are a great concern.

Tool	Text	Binary	Rand	Zero
gz-1	2.8 / 20.2 / 18.3	$\times$	$\times$	$\forall$
gz-6	10.9 / 19.7 / 62.5	$\times$	$\times$	$\forall$
gz-9	24.9 / 49.0 /~10^3	$\times$	$\times$	$\forall$
lzo-1	$\forall$	0.8 / 2.1 / 4.0	$\times$	$\forall$
lzo-3	$\forall$	1.0 / 2.0 / 4.3	$\times$	$\forall$
lzo-9	26.8 / 35.4 /~10^2	~10^2 / ~10^2 /~10^3	$\times$	3.6 / 5.4 / 19.9
bz-1	$\times$	$\times$	$\times$	$\forall$ / 0.3 / $\forall$
bz-9	$\times$	$\times$	$\times$	0.2 / 1.3 / 2.4
c-10	$\times$	$\times$	$\times$	$\forall$
c-16	$\times$	$\times$	$\times$	$\forall$

Table 5.1: The number of reads for each write required to benefit from compression for T (n^T_{be}), E (n^E_{be}) and ET (n^{ET}_{be}) metrics on a **server system**, separated by a /. For break-even values greater than 100 we only report magnitudes. $\forall$ denotes that it is beneficial to use compression for any read-to-write ratio. The symbol $\times$ denotes scenarios when no savings can be made. Values less than 1 represent that just one read can compensate for multiple writes.

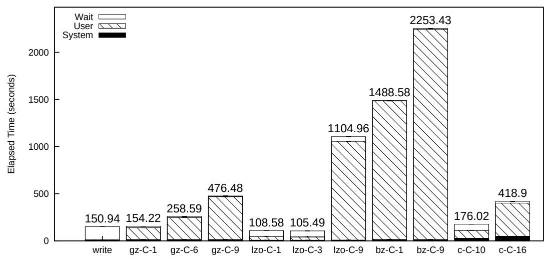
5.4 Desktop Results

In this section, we present and discuss the results of the benchmarks run on the desktop class machine. Overall the results on the desktop show similar trends as those on the server machine (Section 5.3).

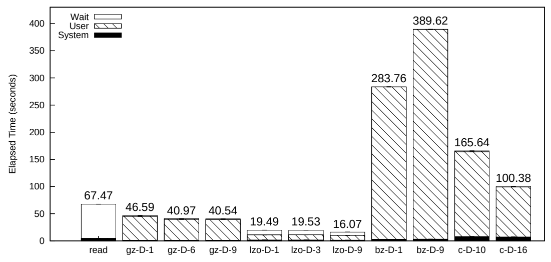
5.4.1 Text File Analysis

As the desktop is equipped with components relatively slower than those on the server machine, we observed that tasks took longer to complete on the desktop machine. For example, since the desktop has a slower disk (7,200 RPM) than the server (10,000 RPM), the reads and writes are much slower than on the server. From Figures 5.8, 5.9, 5.10, and 5.11 we see that the disk on the desktop is about twice as slow as that on the server, in terms of performing I/O. Similarly the slower CPU (1.7 GHz) causes the compression tasks to take longer as well. Correspondingly, the average power requirements of these components are lower than that of the higher performance server class machine. The average idle power of the desktop was 91 Watts (compared to the 218 Watts of the server).

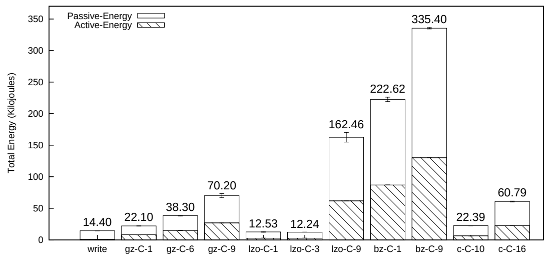
Similar to our observations on the server class machine, we find lzo-1 and lzo-3 to always do better in performance and energy savings than plain-read and plain-write. The gzip utility takes more time and energy during the compression phase, but saves both time and energy while decompressing. Hence, gzip can also prove beneficial, provided that the workload has a read-to-write ratio greater than the corresponding break-even value (n_{be}). We calculate and present the n_{be} values for the desktop system in Table 5.2. However, notice that, unlike in the server, gz-1 never saves energy for any value of n. This is because the energy consumption from our measurements for decompressing is almost the same as that of the plain-read. Hence, any number of reads will



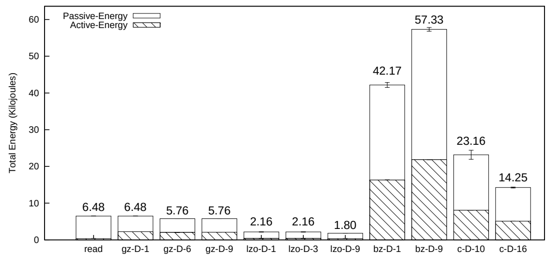
(a) **Time** taken for write vs. compression



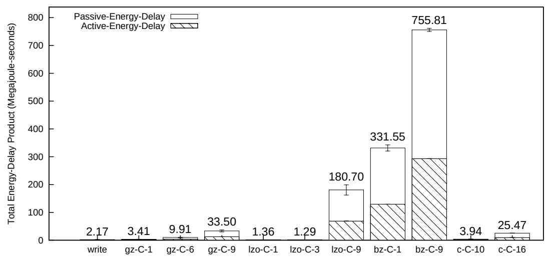
(b) **Time** taken for read vs. decompression



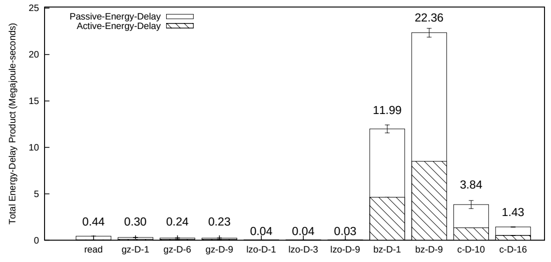
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



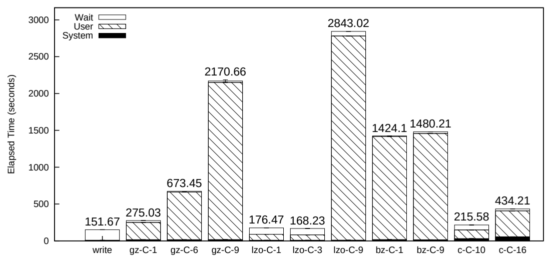
(f) **ET** for read vs. decompression

Figure 5.8: **Text file results for desktop**

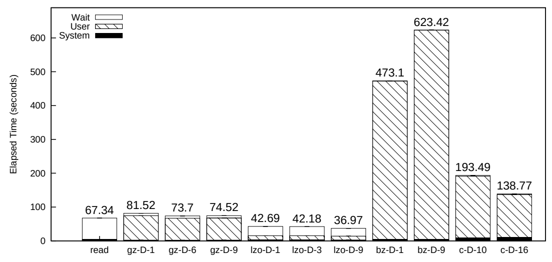
not be sufficient to compensate for the extra energy expended during the compress-write. We also notice that the n_{be}^{E} values for gz-6 and gz-9 are higher than those for the server machine (Table 5.1).

5.4.2 Binary File Analysis

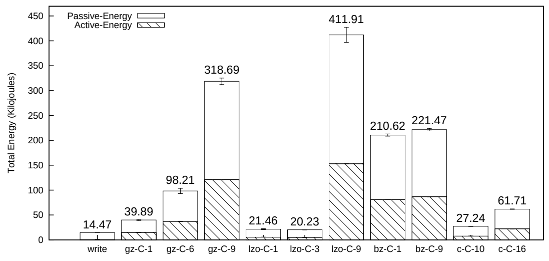
Similar to the server results, we observe that only lzo proves to be beneficial for compressing binary files on the desktop. We see from Table 5.2 that the n_{be} values for the desktop are higher than those for the server. lzo-1 and lzo-3, require about 4 reads for every write to compensate for the additional energy requirements, compared to about 2 for the server machine. Compression is more expensive than a plain write ($E_c - E_w$) on the desktop than the server, and the savings from decompression over the read ($E_r - E_d$) are less prominent on the desktop than the server. Hence the n_{be}^{E} is higher (Section 4.4).



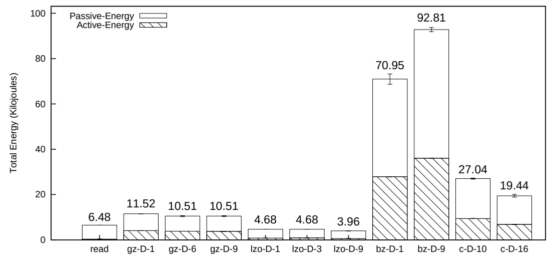
(a) **Time** taken for write vs. compression



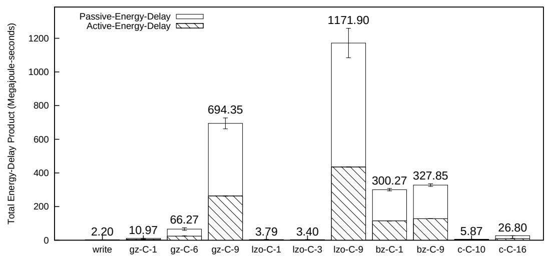
(b) **Time** taken for read vs. decompression



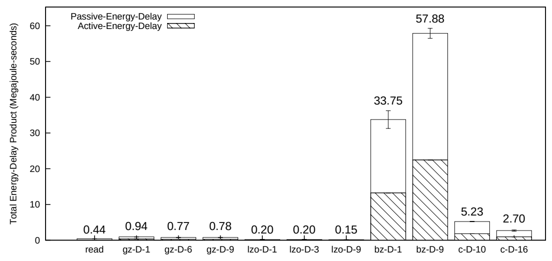
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression

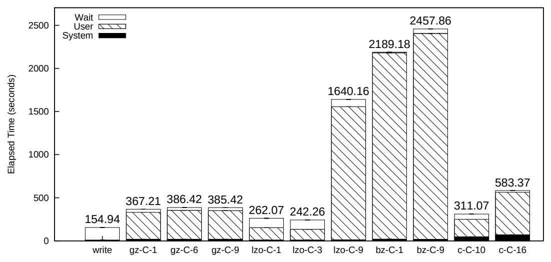


(f) **ET** for read vs. decompression

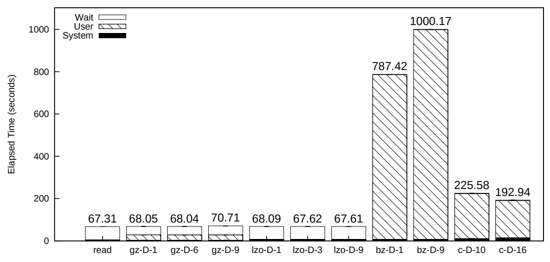
Figure 5.9: **Binary file results for desktop**

Tool	**Text**	**Binary**	**Rand**	**Zero**
gz-1	0.2 / $\times$ / 9.1	$\times$	$\times$	$\forall$
gz-6	4.1 / 33.2 / 38.4	$\times$	$\times$	$\forall$
gz-9	10.8 / 78.6 /~10^2	$\times$	$\times$	$\forall$
lzo-1	$\forall$	1.0 / 3.8 / 6.7	$\times$	$\forall$
lzo-3	$\forall$	0.9 / 3.5 / 5.9	$\times$	$\forall$
lzo-9	18.6 / 31.6 /~10^2	89 / ~10^2 /~10^3	$\times$	1.9 / 4.2 / 15.3
bz-1	$\times$	$\times$	$\times$	$\forall$ / 1.3 / 1.4
bz-9	$\times$	$\times$	$\times$	$\forall$ / 1.7 / 2.3
c-10	$\times$	$\times$	$\times$	$\forall$
c-16	$\times$	$\times$	$\times$	$\forall$

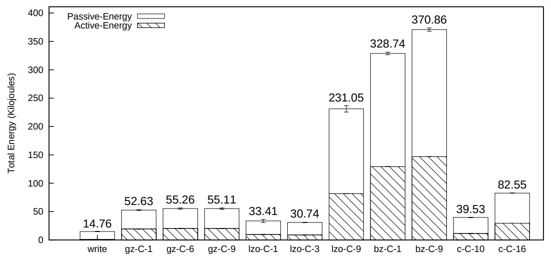
Table 5.2: n_{be}^{T}, n_{be}^{E} and n_{be}^{ET} values for various compression tools for the desktop system.



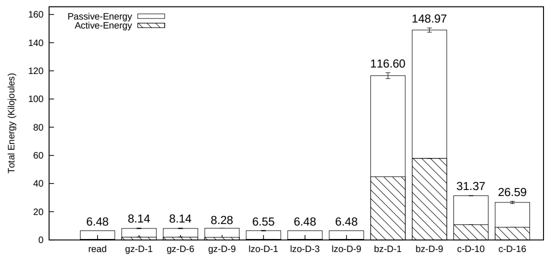
(a) **Time** taken for write vs. compression



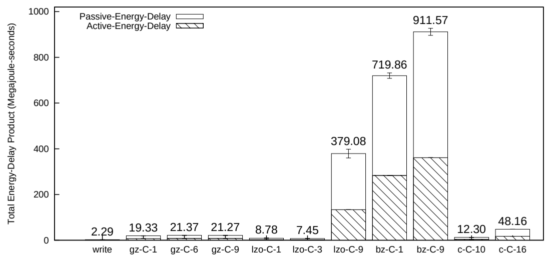
(b) **Time** taken for read vs. decompression



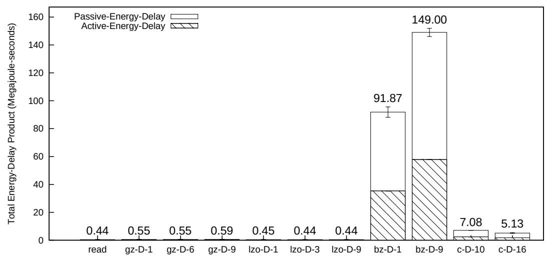
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



(f) **ET** for read vs. decompression

Figure 5.10: **Random file results for desktop**

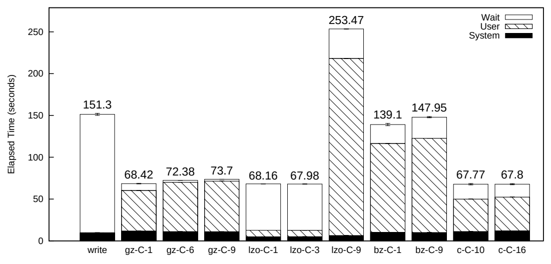
5.4.3 Random and Zero File Analyses

As the trends we see for the random and zero files for the desktop are not much different than that on the server, we only present the graphs here. Discussion similar to the server machine case follows here.

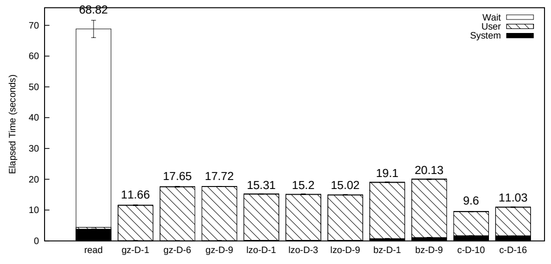
Summary of desktop results. We summarize the evaluation results for the desktop system in Table 5.2. We observe trends similar to those on the server system (Table 5.1).

5.5 Laptop Results

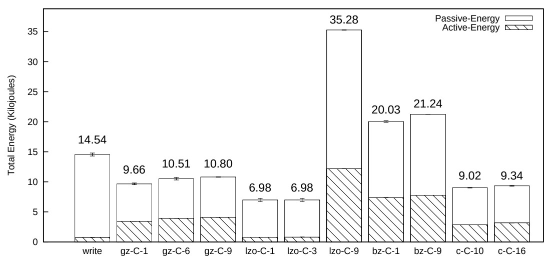
In this section, we present and discuss the results of the benchmarks run on the laptop.



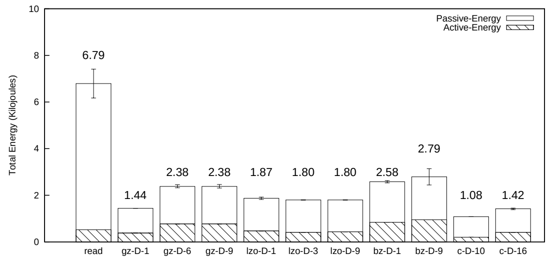
(a) **Time** taken for write vs. compression



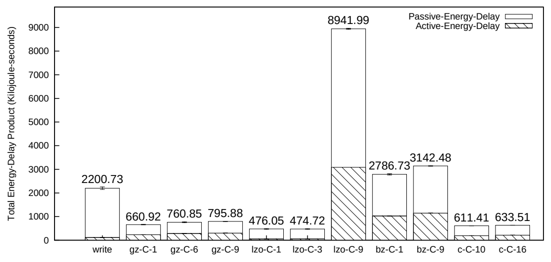
(b) **Time** taken for read vs. decompression



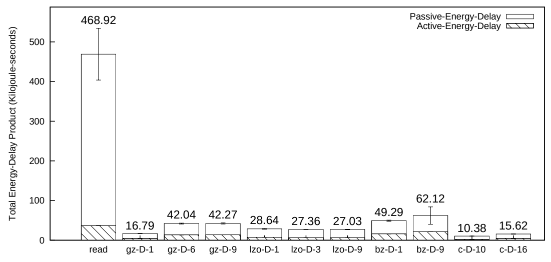
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



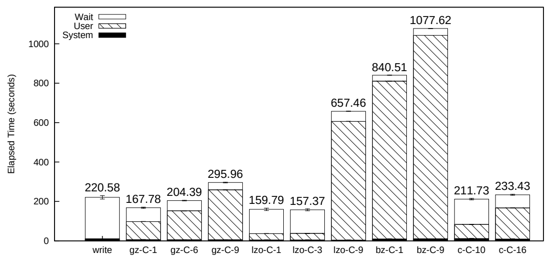
(f) **ET** for read vs. decompression

Figure 5.11: **Zero file results for desktop**

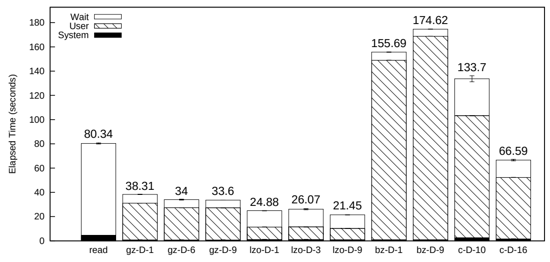
5.5.1 Text file Analysis

Similar to our observation on the server and desktop systems, we find lzo-1 and lzo-3 to always perform better than plain-write and plain-read in terms of both performance and energy savings (Figure 5.12). Using lzo-1 or lzo-3 for compression achieves an average of 28% performance improvement compared to a plain-write; and 69% performance improvement compared to a plain-read. Their energy consumption is 23% lower than plain-write, and 75% lower than a plain-read. lzo-9 and gz-9, like on the server and desktop, need a break-even value of reads to writes in order to be beneficial (Table 5.3).

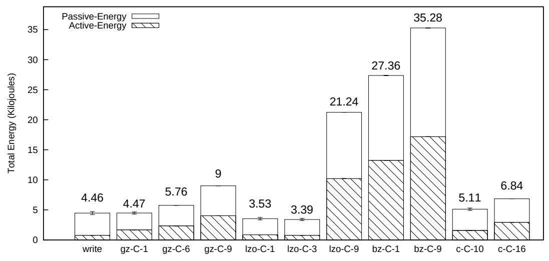
The disk on the laptop is much slower than the CPU. Hence, the CPU intensive gz-1 and gz-6 finish faster than the plain I/O to the disk. We also see that although compressing the file using gz-1 and gz-6 take less time than a plain-write, they require more energy than the plain-write. That is, although gz-1 and gz-6 always perform better than plain-write and plain-read, they save energy only after some number of reads per write. Such behavior was not seen on the server and desktop systems, where savings in time directly translated to savings in energy. The reason for this is the



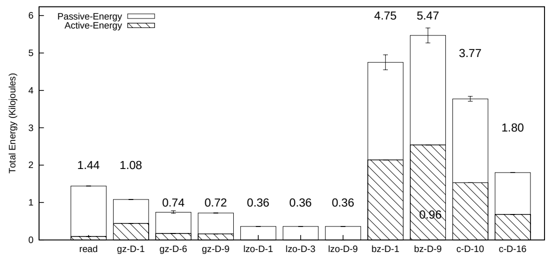
(a) **Time** taken for write vs. compression



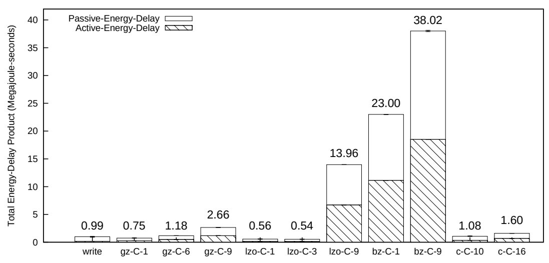
(b) **Time** taken for read vs. decompression



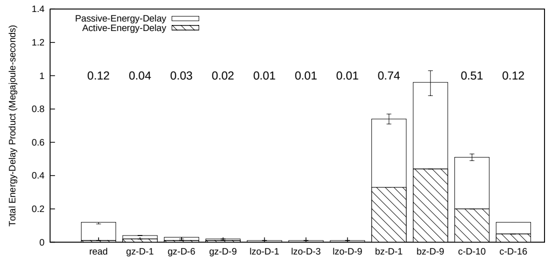
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression

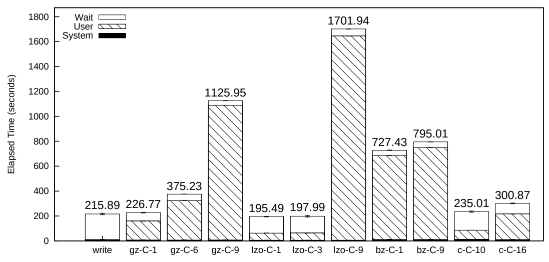


(f) **ET** for read vs. decompression

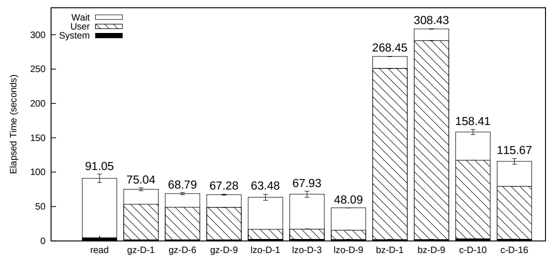
Figure 5.12: **Text file results for laptop**

superior power management ability of the laptop, which results in different levels of average power consumption with different workloads. For example, when an I/O-intensive workload like plain-write is being executed, the processor is dynamically switched to a lower frequency, and hence a lower power state. Therefore, the average power consumption during execution of a plain-write will be lower than that of compress-write, which requires both the CPU and disk to be active and in their highest power states. As the server and the desktop did not support Dynamic Voltage and Frequency Scaling (DVFS) based power management, they always ran in full power mode, irrespective of the component usage.

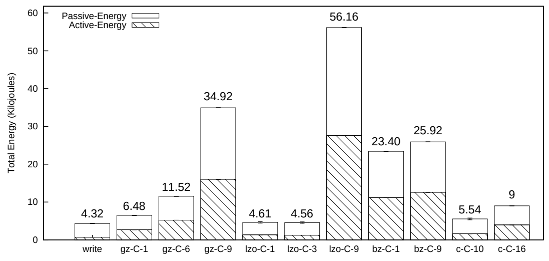
Another interesting observation is that in spite the faster frequency of the desktop's CPU than the laptop's (Table 4.1), bzip2 and higher effort invocation of some tools (lzo-9, gz-6, gz-9) tend to be significantly faster on the laptop, than on the desktop. This is because the desktop has much smaller CPU caches than the laptop (Table 4.1). When a compression tool is invoked with a higher effort parameter, it attempts to look ahead more to obtain an even longer match. This, and algorithms with large memory footprints (e.g., bzip2), result in more cache misses on the desktop



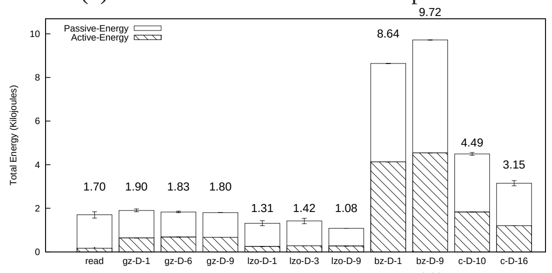
(a) **Time** taken for write vs. compression



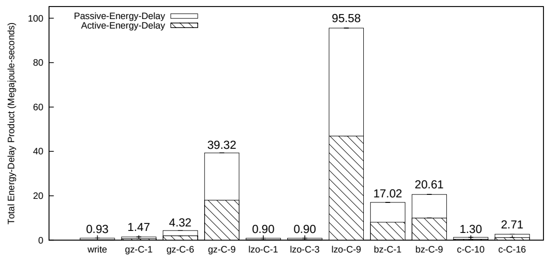
(b) **Time** taken for read vs. decompression



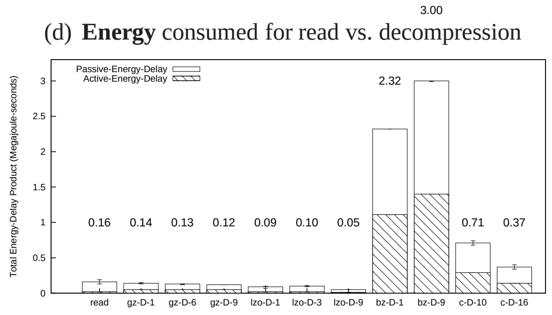
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



(f) **ET** for read vs. decompression

Figure 5.13: **Binary file results for laptop**

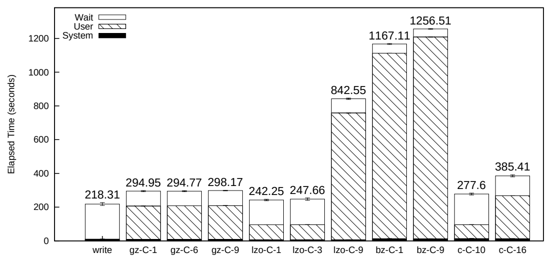
than on the laptop, thereby increasing the execution time on the desktop.

5.5.2 Binary file Analysis

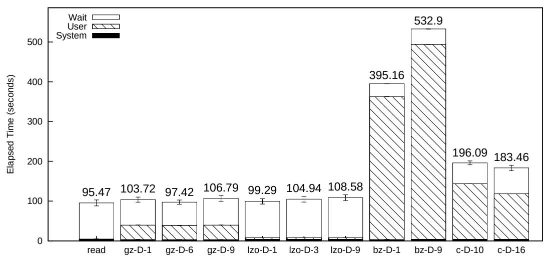
Overall, the trends are again similar to those of the server and desktop. From Figure 5.13 we notice that although gzip and lzo finish faster than a plain-read/plain-write, only lzo saves energy. It, however, needs some number of reads per write to achieve energy savings (Table 5.3). The corresponding n_{be} values are, however, much smaller in magnitude than those for the server and desktop. Finally, bzip2 and compress do not save either time or energy.

5.5.3 Random and Zero file Analyses

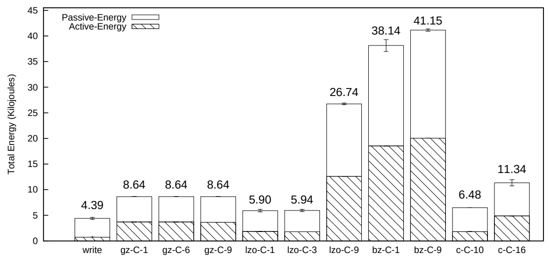
We see similar results for random and zero files as seen on the server and desktop machines. Hence, we omit the discussion here for brevity.



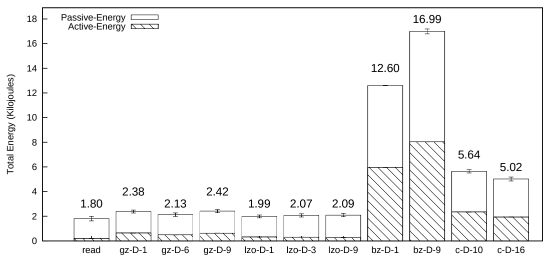
(a) **Time** taken for write vs. compression



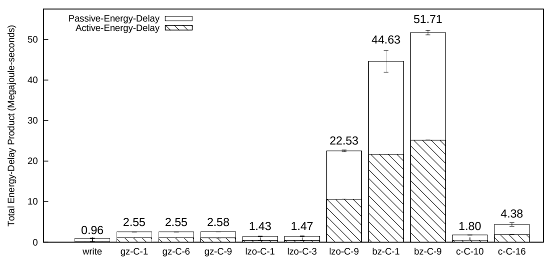
(b) **Time** taken for read vs. decompression



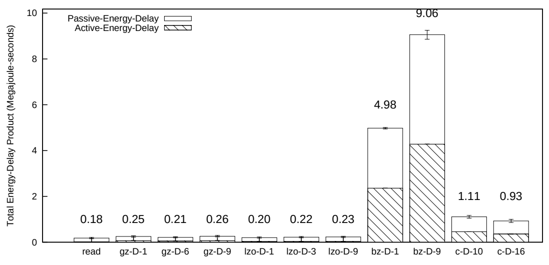
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression

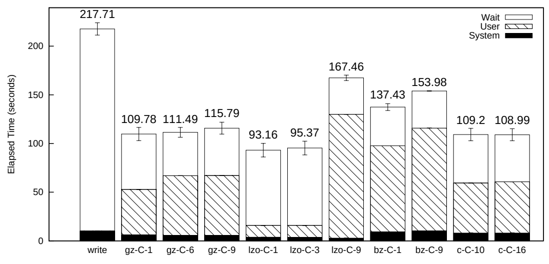


(f) **ET** for read vs. decompression

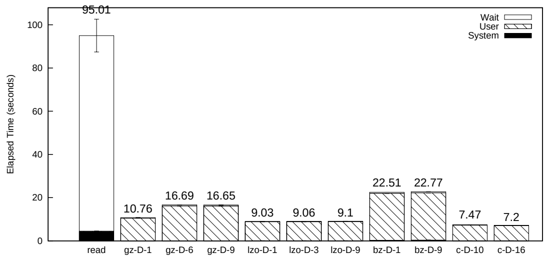
Figure 5.14: **Random file results for laptop**

Tool	Text	Binary	Rand	Zero
gz-1	$\forall$ / 0.03 / $\forall$	0.7 / $\times$ / 46.8	$\times$	$\forall$
gz-6	$\forall$ / 1.9 / 2.1	7.2 / $\times$ / ~10^2	$\times$	$\forall$
gz-9	1.6 / 6.3 / 18.3	38.3 / $\times$ / ~10^2	$\times$	$\forall$
lzo-1	$\forall$	$\forall$ / 0.7 / $\forall$	$\times$	$\forall$
lzo-3	$\forall$	$\forall$ / 0.9 / $\forall$	$\times$	$\forall$
lzo-9	7.4 / 15.5 / ~10^2	34.6 / 84.2 / ~10^2	$\times$	$\forall$ / 0.3 / $\forall$
bz-1	$\times$	$\times$	$\times$	$\forall$
bz-9	$\times$	$\times$	$\times$	$\forall$
c-10	0.2 / $\times$ / $\times$	$\times$	$\times$	$\forall$
c-16	0.9 / $\times$ / $\times$	$\times$	$\times$	$\forall$

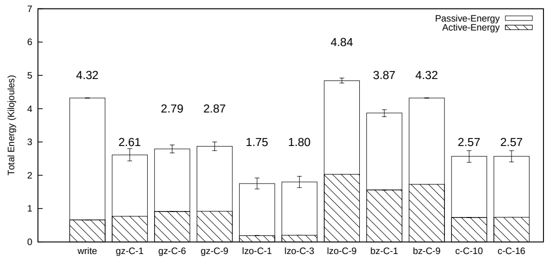
Table 5.3: n_{be}^{T}, n_{be}^{E} and n_{be}^{ET} values for various compression tools for the laptop system.



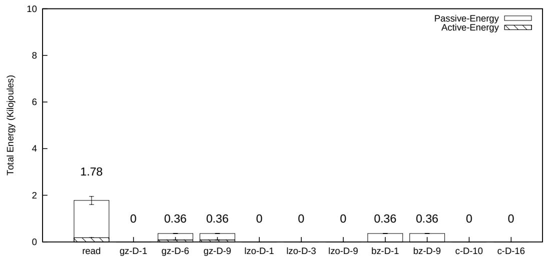
(a) **Time** taken for write vs. compression



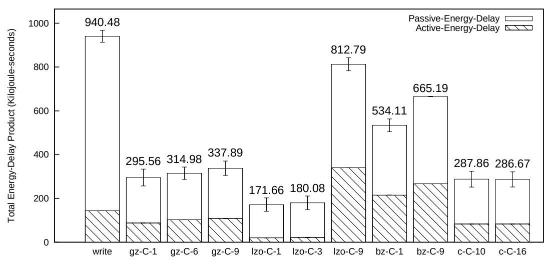
(b) **Time** taken for read vs. decompression



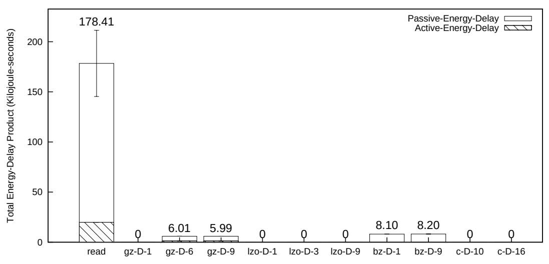
(c) **Energy** consumed for write vs. compression



(d) **Energy** consumed for read vs. decompression



(e) **ET** for write vs. compression



(f) **ET** for read vs. decompression

Figure 5.15: **Zero file results for laptop**

Summary of Laptop results. Table 5.3 summarizes the results of our experiments on the laptop system. In general, we find n_{be} values on the laptop to be much smaller than those on the desktop and server machines. In some case we also see that while a tool is not beneficial from the energy perspective, it exhibits the potential for savings from the energy-delay perspective. Like mentioned earlier in Section 5.5.1, this is because different workloads consume different amounts of average power on the laptop. Even though some compression might finish faster than a plain-write, it consumes more energy than the plain-write. In such cases, while these compression tools are worse than plain-write in terms of E, they have a smaller ET value than the plain-write. Hence they display a value for n_{be}^{ET}, even when they have no n_{be}^{E}.

5.6 Summary of Evaluation

On the server and desktop systems, we notice a strong linear dependency between energy and time found in all experiments. This indicates that all the compression and plain I/O tasks require almost the same amount of average power. This leads to the conclusion that the fastest algorithm is the most energy-efficient one. On the laptop system, equipped with Dynamic Voltage and Frequency Scaling (DVFS) for the CPU, we found that the average power for plain I/O was different from that for a compression task. All the compression tasks, however, exhibited linearity between energy and time. In general, the time required to accomplish the task consists of the time required to perform an I/O and the time required to compress (or decompress) the data. The time for completing I/O operations, in turn, depends on the amount of data to be written, whereas the compression time depends on the algorithm used. This means that an optimal compression tool should have a high compression ratio and low compression time. There is a clear trade-off between compression ratio and the speed of compression.

The file data type affects the compression ratio dramatically. None of the compression tools we considered provided advantages in energy consumption for the files with random content. For zero files, however, almost all tools provided benefits for both reading and writing. For text and binary files, we observe situations when compress-write is less energy-efficient than plain-write, but decompress-read is more energy-efficient than plain-read. We calculated the break-even ratio of reads to writes (n_{be}) in such cases. For most of the compression tools, this value is significantly higher than the read-to-write ratio on common server systems, which has been found to be typically 2–4 [34, 45]. Some notable exceptions were lzo-1 and lzo-3, which are always beneficial for text files. They also save energy in case of binary files, if the read-to-write ratio is at least 2 (or about 4 in terms of ET metric) for the server machine, 4 for the desktop, and 1 for the laptop system. We recommend the use of lzo-1 and lzo-3 in all cases, except the situations where disk space is a greater concern than energy or performance. We also recommend that future systems recognize high-entropy files (e.g., encrypted, random, etc.) and avoid compressing them at all.

In general, we observed similar trends on the three classes of machines we included in our experiments. However, depending on the individual components on each of these machines, they exhibited potential for energy savings at different break-even values.

Chapter 6

Conclusions

In this thesis, our research contribution was to investigate the validity of the assumption that data compression is an effective technique to reduce energy consumption in server systems. We evaluated several compression tools on three different classes of machines running Linux, on a variety of data files and compared them against raw reads and writes based on performance and energy metrics. Our experimental results suggest that no generalized conclusion regarding the efficacy of compression can be drawn. It greatly depends on the data redundancy of the file, the compression algorithm being used, the read-to-write ratio of the workload, and the hardware configuration of the system. We found that compressing zero files is beneficial for almost all the compression tools. Random files are better-off not being compressed at all. Text files, when compressed with lzop using options 1 and 3, will always save energy, irrespective of the workload's read-to-write ratio. We developed a simple read-write model to evaluate energy savings in cases where only compression or decompression saves energy. When applied to text and binary files, it reveals that only gzip and lzop can offer energy savings; in most cases, on the server and desktop, the break-even read-to-write ratio is significantly greater (more than 20) than that found in common workloads. The laptop system however requires much smaller corresponding read-to-write ratio to achieve energy savings. Other than on zero files, bzip2 and the compress utility never save any energy.

Chapter 7

Future Work

We intend to extend this study to a wider range of systems, including systems with multiple cores and multiple CPUs, more CPUs with Dynamic Voltage and Frequency Scaling (DVFS), different disk speeds, etc. We also plan on conducting our study on real server workloads. Compression significantly reduces the storage requirement for data, and hence can result in lesser spinning disks. We plan to extend our current model to factor in the additional power savings thus achieved. We are currently also working on extending gzipfs [63], a stackable compression file system, to include the various compression algorithms we wish to compare. In the future, we plan to explore and evaluate data de-duplication as an energy saving technique.

Another interesting direction would be to include archivers in the evaluation. Archivers generally work by combining multiple files into one. In scenarios where we have several small files, with similar content or format, compressing their archive would typically result in better compression. Decompression on an archive to read one file will, however, be more expensive than if the file was individually compressed.

Bibliography

[1] Ke Yang Alina Oprea, Michael K. Reiter. Space-efficient block storage integrity. In *Proceedings of the NDSS Symposium*, 2005.

[2] P. A. Alsberg. Space and time savings through large data base compression and dynamic restructuring. *Proceedings of the IEEE*, 63(8):1114–1122, 1975.

[3] K. C. Barr and K. Asanovi. Energy-aware lossless data compression. *ACM Transactions on Computer Systems*, 24(3):250–291, 2006.

[4] L. Benini, A. Bogliolo, S. Cavallucci, and B. Ricco. Monitoring system activity for OS-directed dynamic power management. In *Proceedings of the 1998 international symposium on Low power electronics and design (ISLPED '98)*, pages 185–190, New York, NY, USA, 1998. ACM.

[5] L. Benini, D. Bruni, A. Macii, and E. Macii. Hardware-assisted data compression for energy minimization in systems with embedded processors. *Design, Automation and Test in Europe Conference and Exhibition (DATE '02)*, pages 449–453, 2002.

[6] L. Benini, D. Bruni, B. Ricco, A. Macii, and E. Macii. An adaptive data compression scheme for memory traffic minimization in processor-based systems. *IEEE International Symposium on Circuits and Systems (ISCAS '02)*, 4:IV–866–IV–869, 2002.

[7] L. Benini, A. Macii, E. Macii, and M. Poncino. Selective instruction compression for memory energy reduction in embedded systems. In *Proceedings of the 1999 international symposium on Low power electronics and design (ISLPED '99)*, pages 206–211, New York, NY, USA, 1999. ACM.

[8] L. Benini, A. Macii, and A. Nannarelli. Cached-code compression for energy minimization in embedded processors. In *Proceedings of the 2001 international symposium on Low power electronics and design (ISLPED '01)*, pages 322–327, New York, NY, USA, 2001. ACM.

[9] L. Benini, F. Menichelli, and M. Olivieri. A class of code compression schemes for reducing power consumption in embedded microprocessor systems. *IEEE Transactions on Computers*, 53(4):467–482, 2004.

[10] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, and L. C. Mcdowell. The case for Power Management in Web Servers, 2002. `www.research.ibm.com/people/l/lefurgy/Publications/pac2002.pdf`.

[11] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, 2002.

[12] CompuGreen, LLC. The Green500 List. `www.green500.org`, 2008.

[13] D. J. Craft. A fast hardware data compression algorithm and some algorithmic extensions. *IBM Journal of Research and Development*, 42(6):733–745, 1998.

[14] G. Debnath, K. Debnath, and R. Fernando. The Pentium processor-90/100, microarchitecture and low power circuit design. In *Proceedings of the 8th International Conference on VLSI Design (VLSID '95)*, page 185, Washington, DC, USA, 1995. IEEE Computer Society.

[15] F. Douglis, P. Krishnan, and B. Bershad. Adaptive Disk Spin-down Policies for Mobile Computers. In *Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, pages 121–137, Berkeley, CA, USA, 1995. USENIX Association.

[16] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the Power-Hungry Disk. In *Proceedings of the 1994 Winter USENIX Conference*, pages 293–306, 1994.

[17] D. Essary and A. Amer. Predictive data grouping: Defining the bounds of energy and latency reduction through predictive data grouping and replication. *ACM Transactions on Storage (TOS)*, 4(1):1–23, May 2008.

[18] J. Flinn and M. Satyanarayanan. Energy-Aware adaptation for Mobile Applications. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, volume 33, pages 48–63, 1999.

[19] Fluke 289 Digital Multimeter. `http://assets.fluke.com/manuals/287_289_umeng0100.pdf`.

[20] Fluke i410 AC/DC Current Clamp. `http://assets.fluke.com/manuals/i4101010iseng0200.pdf`.

[21] S. Gary, P. Ippolito, G. Gerosa, C. Dietz, J. Eno, and H. Sanchez. PowerPC 603, A Microprocessor for Portable Computers. *IEEE Design and Test*, 11(4):14–23, 1994.

[22] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-state Circuits*, 31(9):1277–1284, September 1996.

[23] B. Gordon and T. Meng. A low power subband video decoder architecture. In *Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume ii, pages 409–412, 1994.

[24] M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *Proceedings of the 35th annual conference on Design automation (DAC '98)*, pages 726–731, New York, NY, USA, 1998. ACM.

[25] H. Huang, W. Hung, and K. Shin. FS2: Dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pages 263–276, Brighton, UK, October 2005. ACM Press.

[26] E. Jeannot, B. Knutsson, and M. Bjorkman. Adaptive online data compression. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02)*, pages 379–388, Edinburgh, Scotland, July 2002. IEEE Computer Society.

[27] N. Joukov and J. Sipek. GreenFS: Making enterprise computers greener by protecting them better. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (EuroSys 2008)*, Glasgow, Scotland, April 2008. ACM. (**Won best paper award**).

[28] M. Kandemir, O. Ozturk, M.J. Irwin, and I. Kolcu. Using data compression to increase energy savings in multi-bank memories. In *Proceedings of the 10th International Euro-Par Conference on Parallel Processing (Euro-Par '04)*, volume 3149 of *Lecture Notes in Computer Science*, pages 310–317. Springer-Verlag Berlin Heidelberg, 2004.

[29] G. Keramidas, K. Aisopos, and S. Kaxiras. Dynamic Dictionary-Based Data Compression for Level-1 Caches. In *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS '06)*, pages 114–129, 2006.

[30] N. Kim, T. Austin, and T. Mudge. Low-Energy Data Cache using Sign Compression and Cache Line Bisection. In *Proceedings of the 2nd Annual Workshop on Memory Performance Issues (WMPI '02)*, 2002.

[31] C. Krintz and B. Cadler. Reducing delay with dynamic selection of compression formats. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC '01)*, pages 266–277, 2001.

[32] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power Aware Page Allocation. In *Proceedings of Architectural Support for Programming Languages and Operating Systems*, pages 105–116, 2000.

[33] H. Lekatsas, J. Henkel, and W. Wolf. Code compression for low power embedded system design. In *Proceedings of the 37th conference on Design automation (DAC '00)*, pages 294–299, New York, NY, USA, 2000. ACM.

[34] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *Proceedings of the USENIX Annual Technical Conference (ATC '08)*, pages 213–226, Berkeley, CA, 2008. USENIX Association.

[35] K. Li, R. Kumpf, P. Horton, and T. Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *Proceedings of the 1994 Winter USENIX Conference*, pages 279–291, 1994.

[36] Wei-Cheng Lin and Chung-Ho Chen. An energy-delay efficient power management scheme for embedded system in multimedia applications. In *Proceedings of The IEEE Asia Pacific Conference on Circuit and System (APCCAS)*, pages 869–872, 2004.

[37] J. R. Lorch and A. J. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications*, 5:48–63, 1998.

[38] Y. Lu, L. Benini, and G. D. Micheli. Operating-System Directed Power Reduction. In *Proceedings of the 2000 international symposium on Low power electronics and design*, pages 37–42, 2000.

[39] R. Manohar and N. Nystrom. Implications of voltage scaling in asynchronous architectures. Technical Report CSL-TR-2001-1013, Departament of Computer Science, Cornell University, 2001.

[40] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: practical power management for enterprise storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST 2008)*, 2008.

[41] J. L. Nunez and S. Jones. Gbit/s lossless data compression hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(3):499–510, 2003.

[42] M.F.X.J. Oberhumer. lzop data compression utility. `www.lzop.org/`.

[43] E. Pinheiro and R. Bianchini. Energy Conservation Techniques for Disk Array-Based Servers. In *Proceedings of the 18th International Conference on Supercomputing (ICS 2004)*, pages 68–78, 2004.

[44] T. Raita. An automatic system for file compression. *The Computer Journal*, pages 80–86, 1987.

[45] D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In *Proc. of the Annual USENIX Technical Conference*, pages 41–54, San Diego, CA, June 2000. USENIX Association.

[46] S. B. Furber. *ARM System Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[47] S. Gurumurthi and A. Sivasubramaniam and M. Kandemir and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proceedings of the 30th annual international symposium on Computer architecture*, pages 169–181, 2003.

[48] C. M. Sadler and M. Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06)*, pages 265–278, New York, NY, USA, 2006. ACM.

[49] C. Su and A. M. Despain. Cache design trade-offs for power and performance optimization: a case study. In *Proceedings of the 1995 international symposium on Low power design*, pages 63–68, 1995.

[50] C. Su and A. M. Despain. Cache designs for energy efficiency. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)*, page 306, Washington, DC, USA, 1995. IEEE Computer Society.

[51] K. Tanaka and T. Kawahara. Leakage energy reduction in cache memory by data compression. *ACM SIGARCH Computer Architecture News*, 35(5):17–24, December 2007.

[52] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schulz, T. B. Smith, M. E. Wazlowski, and P. M. Bland. IBM Memory Expansion Technology (MXT). *IBM Journal of Research and Development*, 45(2):271–286, 2001.

[53] L. Villa, M. Zhang, and K. Asanovi. Dynamic zero compression for cache energy reduction. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture (MICRO 33)*, pages 214–220, New York, NY, USA, 2000. ACM.

[54] Watts up? PRO ES Power Meter. `www.wattsupmeters.com/secure/products.php`.

[55] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, 1994.

[56] M. Weiser, B. Welsh, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. *Mobile Computing*, 353:449–471, 1996.

[57] D. Wheeler. Linux utility for wattsup pro es power meter. `www.wattsupmeters.com/forum/index.php?topic=7.0`.

[58] J. Wilkes. Predictive power conservation. Technical Report HPL-CSP-92-5, Hewlett-Packard Laboratories, February 1992.

[59] Y. Wiseman, K. Schwan, and P. Widener. Efficient end to end data exchange using configurable compression. *ACM SIGOPS Operating Systems Review*, 39(3):4–23, 2005.

[60] C. P. Wright, N. Joukov, D. Kulkarni, Y. Miretskiy, and E. Zadok. Auto-pilot: A platform for system software benchmarking. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 175–187, Anaheim, CA, April 2005. USENIX Association.

[61] R. Xu, Z. Li, C. Wang, and P. Ni. Impact of Data Compression on Energy Consumption of Wireless-Networked Handheld Devices. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS '03)*, page 302, 2003.

[62] Y. Yoshida, B. Song, H. Okuhata, T. Onoye, and I. Shirakawa. An object code compression approach to embedded processors. In *Proceedings of the 1997 international symposium on Low power electronics and design (ISLPED '97)*, pages 265–268, New York, NY, USA, 1997. ACM.

[63] E. Zadok, J. M. Anderson, I. Bădulescu, and J. Nieh. Fast indexing: Support for size-changing algorithms in stackable file systems. In *Proceedings of the Annual USENIX Technical Conference*, pages 289–304, Boston, MA, June 2001. USENIX Association.

[64] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, pages 123–132, 2002.

[65] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture*, pages 118–129, 2004.

[66] V. Zyuban and P. Kogge. Optimization of high-performance superscalar architectures for energy efficiency. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED '00)*, pages 84–89, 2000.