UNIVERSITY OF OSLO
Department of Informatics

Performance Comparison of Btrfs and Ext4 Filesystems

Meaza Taye Kebede

Network and System Administration

Oslo And Akersus University College

Of Applied Science

May 23, 2012

Performance Comparison of Btrfs and Ext4 Filesystems

Meaza Taye Kebede

Network and System Administration
Oslo And Akersus University College Of Applied Science

May 23, 2012

Abstract

This thesis presents an overall performance comparison between the Btrfs and Ext4 filesystems by using both synthetic and real world application benchmarking tools. It also compares Btrfs's transparent compression and logical volume management features with Ext4 in combination with Linux LVM and compression software, respectively. In addition to this,the Btrfs defragmentation tool is also evaluated in terms of space reduction and time required to perform the defragmentaion process. The results obtained from the Iozone benchmarking tool show a large difference between Btrfs and Ext4. However, the results of real application tests are much more similar. The results from the compression tests show that utilizing Btrfs's default compression feature brings performance improvements only for large files while the LZO compression option shows performance improvements for both small and large files. File and directory compression test results shows that bzip2 compression software is capable of providing the highest space savings but with the cost of time. On the other hand, Btrfs transparent compression with the compress-force option provides a good deal of space saving coupled with lesser time. The Btrfs defragmentation tool results show that the tool is efficient both in terms of reducing file fragmentation and its speed while performing the defragmentation process.

Acknowledgements

This research work benefited from the support of many people. My greatest intellectual debt is to Æleen Frisch who has contributed critically and substantively to make the accomplishment of this work possible. Over and above the intellectual guidance, her encouragement and unreserved support is what made accomplishing this work possible.

I owe sincere appreciation for the outstanding support of the department staff. I am especially indebted to Hårek Haugerud, Kyrre Begnum and Ismail Hassan for the unreserved support they have given me in the two years that I have been in the institute.

I am also grateful to my family and almighty God. In a very real sense, this research would not have been possible without their unfailing support. Bethsaida Taye, Meselu Taye and especially Kidus Solomon, thank you so much.

A number of friends have contributed to my academic and nonacademic life in Norway. My special thanks go to my best friends Yared, Daniel and Kidist. Thank you for being there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

A filesystem is the method and data structure that an operating system uses to keep track of files on a disk or partition [1]. The desire to develop a better performing filesystem is an issue that has been significant for decades. Currently, the advent of high performance storage devices makes it an even more crucial topic that needs due consideration. In general, having a filesystem that can provide scalability, excellent performance and reliability is a requirement for modern computer systems.

Over the years, the Linux operating system has provided different kinds of filesystems, beginning with the well known ext2, as its default base file system. More recent ones have added a variety of features and functionality having their own strengths and shortcomings.

The ext4 and Btrfs filesystems are two recently developed filesystems designed by focusing on performance and scalability. The Ext4 filesystem was developed to addressing scalability, performance, and reliability issues faced by ext3[4]. It provides support for large size filesystems and advanced features such as implementation of extents, delayed and multi-block allocations (in order to prevent file fragmentation), and persistent preallocation.

The Btrfs filestem was developed beginning in the year 2007. It provides additional features over those in the ext4 filesystem. Btrfs was is designed to deliver significant improvements in scalability, reliability, and ease of management [2]. The Btrfs filesystem has built-in support for software RAID, including balancing multiple devices and recovering from corruption. It also supports live resizing and device addition and removal [3], as well as transparent compression, creation of snapshots and support for subvolumes.

Even though current technology is able to provide huge capacity storage devices at affordable prices, the demand for more storage space is never satisfied. Often, the most appropriate solution for providing the desired storage space is

by utilizing multiple disks that can integrate and function as one huge storage device. This in return requires a means of efficiently managing these devices without creating a negative impact on overall system or I/O performance. Logical volume management is typically used to handle the management of such storage devices.

File fragmentation is another factor that affects I/O performance. Different filesystems provide different mechanisms for preventing fragmentation. Nevertheless, it is not entirely possible to entirely eliminate the occurrence of fragmentation. The ageing process of filesystems and the addition and deletion of files will end up creating fragmentation, affecting the overall I/O performance. Defragmentation is a solution that is used to tackle the unpreventable occurrence of fragmentation in files. Both filesystms provide online defragmentation functionality as a solution for the occurrence of fragmentation.

The aim of this research is to undertake an I/O performance comparison between Btrfs and ext4 filesystem. It will examine their general performance for a variety of tasks. It will also determine if there is a performance impact associated with the added features of compression and logical volume management which are part of Btrfs and available via separate software for ext4.

1.2 Problem Statement

The research described in this thesis compares the Btrfs and ext4 filesystems, focusing on the following questions:

- How does the performance of the two filesystems compare for a single partition (i.e., used without volume management features)?
- How does Btrfs perform while using its built-in logical volume management system as compared with ext4 filesystem with the Linux Logical Volume Manager (LVM)?
- What is the performance impact on Btrfs while using its compression feature?
- How effective is Btrfs built in compression feature as compared with ext4 with compression software?
- How much efficient is the defragmentation feature of Btrfs?

For this discussion, the term performance refers primarily to I/O throughput of the filesystem, and the term efficient refers to the ability to perform defragmentation faster and reduce fragmentation to a lower level.

1.3 Thesis Outline

This paper is organized in the following manner:-

The first chapter provides the motivation of the research peaper and specify reasearch questions that needs to be addressed in this research paper.

The second chapter provides background information about filesystems in general, detailed feature design and structure of Btrfs and Ext4 filesystems and also related works that have been done on Btrfs and Ext4 filesystems.

The third chapter explains the experimental setup , hardware and software specification as well as about the selected benchmarking tools.

The fourth chapter present results obtained form different benchmaking tools used for this project.

The fifth chapter present analysis and discussion based on the result obtained form the fourth chapter.

The sixth chapter is dedicated for conclusion and suggestion for future works.

Chapter 2

Background and Literature

This chapter will discuss background information about fileystems, logical volume management, block allocation methods and compression. Sections 2.2 and 2.3 will provide detailed discussions of the features and design of the Btrfs and Ext4 filesystems, and the last section will describe related works.

2.1 Filesystem

Filesystems determine the way that the storage of data is organized on a disk. Linux operating systems have different kinds of filesystems with features that differentiate them from one another. Each type of filesystem has its own set of rules for controlling the allocation of disk space to files and for associating related data about each file (known as metadata) with that file. Metadata includes its filename, the directory in which it is located, its permissions and its creation and modification dates[1].

The flexibility of the Linux operating system in supporting multiple filesystems arises from its implantation of abstraction in its low-level filesystem interface. This is possible because the Virtual Filesystem Switch (VFS), a special kernel interface level, defines a common, low-level model that can be used to represent any particular filesystem's features and operation[14]. In addition to this abstraction of the lowest-level file operation from the underlying filesystem, the VFS also connects physical (block) devices to the actual filesystems that are in use[13].
The following figure shows the components of Linux filesystems.

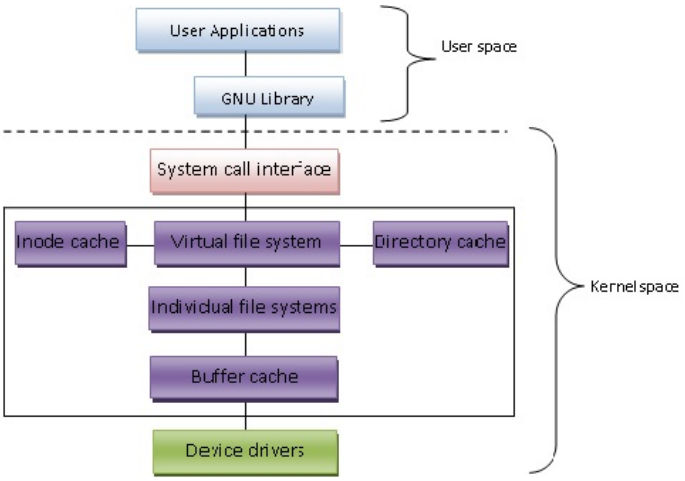

Figure 2.1: Architecture of Linux filesystem components

User space contains the user applications and the GNU C Library (glibc), which provides the user interface for the filesystem calls, namely open, read, write and close. The system call interface, which is acting as a switch, redirects system calls from user space to the appropriate locations in kernel space within the VFS. The VFS is the primary interface of the underlying filesystem. It in turn exports a set of interfaces to the individual filesystems. Each individual filesystem must implement the common set of interfaces that is required by the VFS [14].

The Linux filesystem support includes some caching features. The Inode and dentry caches contain recently used filesytem objects for fast access and improved performance. The other type of cache is the buffer cache that is used to buffer requests between the block devices and filesystems[12].

2.1.1 Linux Filesystem Data Structures

Linux views all filesystems as a common set of objects, which are categorized into four major parts. The first one is the superblock that describes the structure and maintains the state of filesystems. The second major object is the Inode (short for index node) which contains metadata that is used to manage objects and specify which operations are permitted on those objects. The third object type is the directory entry (dentry), which represents a directory entry as a single component of a path. The final major object is the file object, which represents an open file associated with a process[14].

Superblock

The Superblock is a structure that represents a filesystem as a whole, together with all required information that is necessary to manage the filesystem. This information includes the name, size and state of the filesystem, a reference to the underlying block device and filesystem metadata information.

Inode

An Inode is the data structure on disk that describes and stores a file's attributes, including its physical location on disk. Inodes are created at the initial stage of filesystem creation. Historically, the number of Inodes equals the maximum number of files of all types that can exist in a filesystem[15]. Inodes hold information such as the type of file, its access permissions, its user and group owner ids, the time of the most recent modification done to the file, the size of the file and the disk address of the file's data blocks. In general, Inodes store all information about the file except the name. The filename is stored in the directory where the file is located, together with the Inode number of the file.

2.1.2 Traditional filesystems

The Berkeley Standard Distribution (BSD) fast filesystem is the traditional filesystem used all but the earliest Unix systems. It was designed to address the performance limitations of the original System V filesystem[15]. The BSD filesystem supports filesystem block sizes of up to 64KB. Even though the increased block size over System V improves performance, it will also creates internal fragmentation as a result of wasted space. In order to tackle this problem, the BSD filesystem additionally divides a single filesytem block into fragments, and each block can be broken down in to two, four or eight fragments, which can be addressed separately [15].

The BSD filesystem divides the filesystem partitions into cylinders groups, which are comprised of one or more consecutive cylinders. Each cylinder groups will have a copy of the Superblock, a fraction of the Inodes for the filesystem and data blocks, and the block map that describes available blocks in the cylinder group[15]. The Superblock is replicated in each cylinder group for the purpose of redundancy. Since each cylinder group contains a free block map, Inodes and blocks, together with the copy of Superblock, the occurrence of data loss on some part of the disk will not affect other cylinder groups that do not belong to the affected cylinder group. The BSD filesystem directory structure is a linear list which contains a length field and the file name whose length can be up to 255 bytes.[19]

The major drawback of the BSD filesystem is its demand to perform filesystem checking at every boot, which takes a long time. This slowness is intolerable, especially with the huge storage devices of the current era.

The default Linux filesystem for many years was the Ext2 filesystem. Ext2 inherits most characteristics from BSD filesystem and makes changes to three basic features. The first change is the elimination of fragments. The increase in disk space and file size makes the demand of partitioning blocks into fragments less important[19]. As a result, the Ext2 filesystem provides a single allocation unit, the block size, for all allocations. The second change made by

Ext2 is its usage of fixed size blocks instead of cylinder groups to divide the filesystem partition, since block size is more meaningful for newer hard disk types. The third and basic change made with Ext2 is utilization of buffer cache to store metadata until it is flushed to disk, in contrast to the BSD filesystem which writes out metadata immediately to disk[19].

2.1.3 Allocation Methods

Filesystems use different kinds of allocation methods to allocate disk blocks for file storage. The type of allocation method selected and implemented in a filesystem is one of the determining factors for its overall performance since effective disk space utilization and quick access to a file depends on the space allocation technique used by the filesystem[10]. In general, there are three widely used allocation methods.

Contiguous Allocation

The contiguous allocation method requires a file to occupy a set of contiguous blocks on the disk[10]. The location of a file is defined by the disk address of the first block and the size of the file. Since all records are placed next to each other, sequential access of a file is fast. Moreover, random access is also fast as it only requires getting the starting block and size of a file, which is stored in the directory entry, to locate it.

The difficulty encountered with this allocation method is finding space for new file. Two common strategies, namely first fit and best fit, are used to select an unallocated segment for the requested space of the new file [10]. The former searches for a space until it finds one that is big enough to fulfil the requirement, while the latter searches for the smallest possible unallocated segment or hole that is big enough to hold the required size. Even though these strategies may help in locating the total amount of space needed for the new file, preallocation is still a major issue. Since a file can grow from time to time, the currently allocated space might end up being unable to fulfil the new size requirement, causing the file to require relocation. This is detrimental to performance and causes filesystem fragmentation.

Extent-based allocation maintains all the advantages of contiguous allocation techniques while at the same time provides a solution to prevent this problem. Instead of allocating a single block, this technique initially allocate a contiguous chunk of space (an extent) that can be enlarged by adding another chunk of contiguous space as the demand arises. In extent based allocation, the location of a file's block is recorded as a location and a block count, plus a link to the first block[10].

Linked Allocation

The linked allocation technique uses a linked list of disk blocks for each file. The directory entry for a file contains pointers to the first and last file blocks[10]. Each data block uses 4 bytes of its space for a pointer to the next block of the file. The the last block specifies an end-of-file value in this location. This scheme is effective for sequential file access, but it does not support direct access for a single block. Direct access is only possible if implemented with a table which stores all pointers to a file.

This technique also has the advantage that it eliminates external fragmentation and allows files to increase size easily. Its greatest shortcoming is reliability. Since disk blocks are linked by pointers, a problem occurring within a single pointer can make all the remaining blocks in the chain inaccessible without rebuilding the filesystem metadata.

Indexed Allocation

In this allocation method, an index block is allocated for each file that is created. The index block of a file contains pointers to all of the data blocks for that file, essentially an array of disk block addresses[10]. The directory entry for the file contains a pointer to this index block. Indexed allocation supports both sequential and direct access. It eliminates the occurrence of external fragmentation and also the problem of file growth exhibited by the contiguous block allocation technique.

However, one of the shortcomings associated with this technique is the occurrence of internal fragmentation as a result of a free space wastage on index blocks. The other issue is the the overhead associated with having an index block, which is most significant for small files.

2.1.4 Logical Volume Management

The Linux Logical Volume Manager version 2 (LVM2) provides a higher-level view of the disk storage on a computer system than the traditional view of disks and partitions. It can combine multiple physical block devices to create a single large logical block device that overcomes the storage limitation imposed on a single device. This logical device, often called a volume group, can be subdivided into logical entities known as logical volumes or simply volumes. Creating filesystems using logical volumes eliminates the administration overhead by providing greater flexibility in storage space allocation. Since volumes can be resized or relocated while a filesysstem is mounted on top of them, it offers more flexibility on adjusting the required amount of storage space[5].

Internally, LVM2 is divided in to two major parts. The first part is a device mapper, which is a kernel space program that is responsible creating virtual block devices and mapping their content in other block devices. It establishes a mapping between logical blocks (i.e., logical volumes) and physical devices. The second part is a user space tool that is comprised of different commands which are used to manage logical volumes[17].

In LVM2, extents are used as a common measurement of block size used for mapping physical volumes in to logical volumes. The default size of an extent is 4 MB and there is no limit to the number of extents per physical or logical volume. The extent size that is selected at initial stage is used for shrinking or extending logical volumes accordingly. The size of extents are required to be of the same size within a single volume[17].

2.1.5 Transparent compression

Transparent compression is a way of providing automatic, on-the-fly data compression for an entire filesystem without any user knowledge or intervention. The major advantage of compression is saving disk space but it also can provide reduced disk I/O operations, which in turn leads to improvement in the filesystem's overall performance compared[11].

2.1.6 Fragmentation

Modern filesystems have implemented different ways to eradicate fragmentation. Even if using delayed and multiblock allocation methods used by modern filesystems minimize the occurrence of fragmentation in files, they don't entirely eradicate it. Over time, fragmentation will still occur within a filesystem[7]. Fragmentation can be categorized into internal and external fragmentation. Internal fragmentation occurs when a file does not fill up a block allocated to it completely and results in a wastage of space that can not be used for any other purpose. The second type of fragmentation which is external fragmentation. This occurs when the logical blocks that make up a file are scattered all over the disk. This type of fragmentation has a negative impact on performance.

2.2 The Btrfs filesystem

Btrfs (the name stand for b-tree filesystem) is a copy-on-write (COW) Linux filesystem which is intended to address the lack of pooling, snapshots, checksums and integrated multi-device spanning in traditional Linux filesystems[3]. It has many features such as its support for snapshots of a live system, including rollback to a previous state, its capability to perform offline conversion of Ext3 and Ext4 filesystems, online block device addition and removal, and online volume growth and shrinking. Btrfs is designed to solve the problem of scalability that often occurs with large and fast storage[8].

2.2.1 Btrfs Design and Data Structure

Btrfs uses b-trees to store generic objects of varying data types in a single, unified data structure. A b-tree is a tree data structure that allows tree nodes (also known as leaves) to have more than 2 child nodes. B-trees are designed for performance, and perform operations like searching, insertion and deletion in logarithmic time.

Inside the b-tree, root nodes consists of two fields: the key, which holds information about the item contained in the leaves of a tree, and the block pointer, which provides information about the disk location of the next node or leaf in the b-tree[2].

Btrfs uses three types of on-disk structures, namely block headers, keys and items. The block header contains information about the block, including a checksum for the block contents, the universal unique identification (UUID) of the filesystem that owns the block, the level of the block in the tree, and the block number where this block is supposed to live.

Leaves of the tree hold the item and data fields, they grow toward one another. Items are combinations of keys and data, where the offset and size field of the item indicates the location of the item in the leaf. This way of storing the key with the data makes efficient use of space compared to the usual way of storing of only one kind of data in any given filesystem block[20].

Items are sorted by their 136-bit key, which groups related items together via a shared key prefix (and thus automatically optimizes the filesystem for large read and write operations). Small files can be stored directly in the tree leaves, while large files are allocated by extents. This technique both lowers the overhead and reduces fragmentation.[2]

A key is divided into three chunks, which are the object id, type and offset fields. Each object in the filesystem has an object id, which is allocated dynamically on creation. The object id field allows all items for a given filesystem object to be logically grouped together in the b-tree. The offset field of the key stores the byte offset for a particular item in the object. The type field indicates the type of data stored in the item[20].

Btrfs component b-trees

A newly-created Btrfs filesystem contains five types of b-trees[12], as illustrated in Figure 2.2:

- The tree of root trees b-tree keeps track of the location of all the roots of the filesystem b-trees. It serves as a directory for all other tree roots.

- The extent tree holds information about extents allocated for the filesystem.
- The filesystem tree which contains the files and directory information.
- The chunk tree holds information about chunks of the device that are allocated and the type of data they hold.
- The checksum tree checksums of all data extents within the filesystem.

The Btrfs filesystem Superblock contains two pointers. The first pointer points to the tree of root trees, and the second pointer points to the chunk tree, which is responsible for device management[20].

Btrfs Inodes are stored in struct Btrfs_Inode_item. The Btrfs Inodes store the traditional Inode data for files and directories (as returned by the stat system call). The Btrfs Inode structure is relatively small, and does not contain any embedded file data or extended attribute data[2].

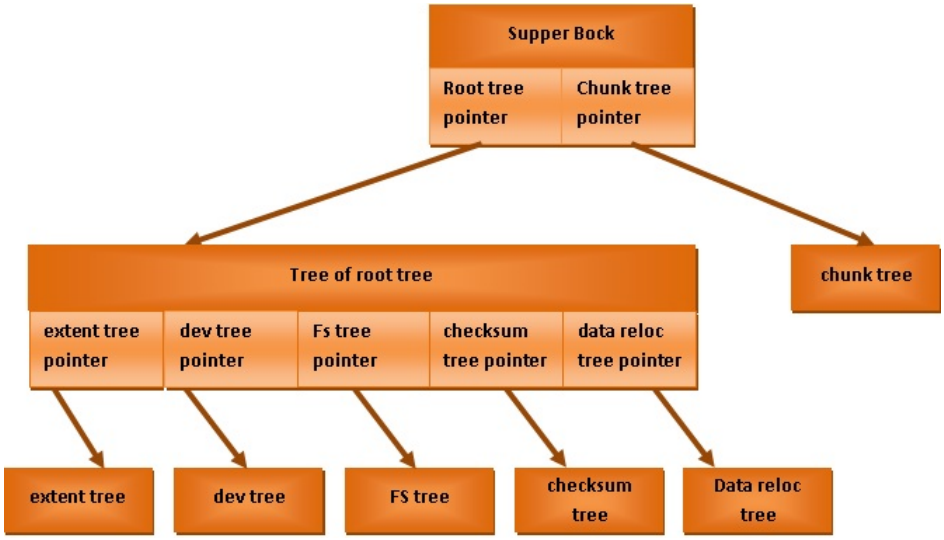

Figure 2.2: Btrfs Btree

2.2.2 Large filesystem support

As a 64-bit filesystem, Btrfs addresses up to 16 exabytes (16,384 petabytes), both in terms of the maximum volume size and the maximum file size[12].

2.2.3 Dynamic Inode allocation

When creating the filesystem, only a few Inodes are established, rather than creating all Inodes that will ever exist at the very beginning. Based on the actual filesystem use, additional Inodes are created and allocated, which is

suitable for data de-/compression in real-time. This means favoring speed over the best possible compression ratio.

2.2.4 Compression

Btrfs implements transparent compression with two kinds of compression schemes, LZo and Zlib, with Zlib being the default method[11]. This feature can be turned on at the mount option, and any new writes will be compressed. Moreover, Btrfs automatically identifies what should and should not be compressed to make this feature more efficient[11]. Both LZo and Zlib are of a lossless compression technique, i.e the original data can be recovered exactly from its compressed data counterpart.

- Lempel-Ziv-Oberhumer (LZO) compression is a data compression library that is suitable for data de-/compression in real time, and it which favours speed over compression ratio. It is a block compression algorithm that compresses a block of data into matches (using a sliding dictionary) and runs of non-matching literals[25]. Unlike Zlib, LZo supports a number of algorithms.
- The Zlib compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data. It supports DEFLATE algorithm that provides good compression on a wide variety of data with minimal use of system resources[26].

2.2.5 Built in volume management and RAID support

Btrfs implements software RAID as part of the filesystem. Currently it supports RAID 0 (disk striping), RAID 1 (disk mirroring) and RAID 10[12]. The built-in logical volume management feature of Btrfs eliminates some complications that can arise while using LVM2[11].

2.2.6 Subvolumes

Btrfs has the capability of taking parts of a filesystem and remounting them as the root for another filesystem. This is useful it you want to limit user access to a certain potion of a directory structure. For example, if there is a subdirectory that users need to access without being allowed access to other parts of the main directory, then the user subdirectory can be mounted as a subvolume. To the user, it appears as the root (top level) directory for that directory tree[12].

2.2.7 Snapshots

Btrfs is capable of creating snapshots of a filesystem or sections of filesystem[11]. This is advantageous since snapshots can be used for backup operations or for any other purpose as required. Moreover, Btrfs allows creation of writeable snapshots and is capable of taking a snapshot of a snapshot[12].

2.2.8 Delayed Allocation

Like many other modern filesystems, Btrfs uses delayed allocation to allow for better disk allocation. This means that Btrfs will only allocate space on the disk when the kernel decides it needs to get rid of dirty pages in memory. This technique results in much larger allocations being made and much larger chunks of sequential data, which in turn makes reading the data back faster[11].

2.2.9 Online defragmentation

Even though efficient allocation mechanisms are used to prevent fragmentation, fragmentation happens anyway over time, and it can severely impact performance. To address this problem, Btrfs implements an online defragmenation tool that operates while the filesystem is mounted and in use[12].

2.2.10 Checksumming

Btrfs checksums all data and metadata for detecting errors and providing filesystem integrity.

2.3 The Ext4 filesystem

The Ext4 filesystem, which is also known as the fourth extended filesystem, is a journalled filesystem. As a successor of the well known Ext3 filesystem, it maintains some of its features and is capable of maintaining backward comparability. Moreover, it also makes possible the online migration of Ext3 filesystem to Ext4[7]. Ext4 achieved various improvement in terms of scalability, reliability and overall performance when compared with its predecessor[7].

The Ext4 filesystem has a default Inode size of 256 bytes, a larger size that its predecessor. The additional space is required in order to store additional fields such as extended attributes, timestamps (with time measurements in terms of nanoseconds) and file checksums. The on-disk structure of the Ext4 Inode is similar with that of its predecessor except for the addition these new fields.

2.3.1 Large filesystem support

Ext4 supports a larger filesystem and file sizes, and subdirectory limits. It supports filesystems of up to 1 exabyte and files up to 16TB in size (when using 4KB blocks). Moreover, the subdirectory limit is virtually unlimited. Directory indexing was also optimized to a hashed b-tree-like structure, so although the size limits were much increased, Ext4 nevertheless supports very fast lookup times[7].

2.3.2 Extents

Ext4 allows blocks for a particular file to be stored as an extent, a contiguous sequence of physical blocks, unlike its predecessor that implements an indirect block map. Using this feature eliminates the performance problems present with the Ext2 and Ext3 filesystems while also allowing the efficient mapping of very large files to disk blocks. Ext4 is capable of mapping up to 128 MB of contiguous space in a single extent (assuming a 4KB block size).

In addition, this feature reduces the occurrence of file fragmentation and improves performance by supporting an efficient storage structure. Extents in Ext4 provide a layered approach to efficiently representing small files, as well as extent trees to efficiently represent large files[7]. A single Ext4 Inode can reference up to four extents. For large files that require more than 512 MB, Ext4 uses an extent tree. The extent tree contains two type of nodes: leaf nodes and an index node. An extent header is found in both index and tree nodes, and it contains the number of entries, the depth of the tree and the maximum capacity. The following figure compares traditional indirect mapping and the extent-based mapping of Ext4.

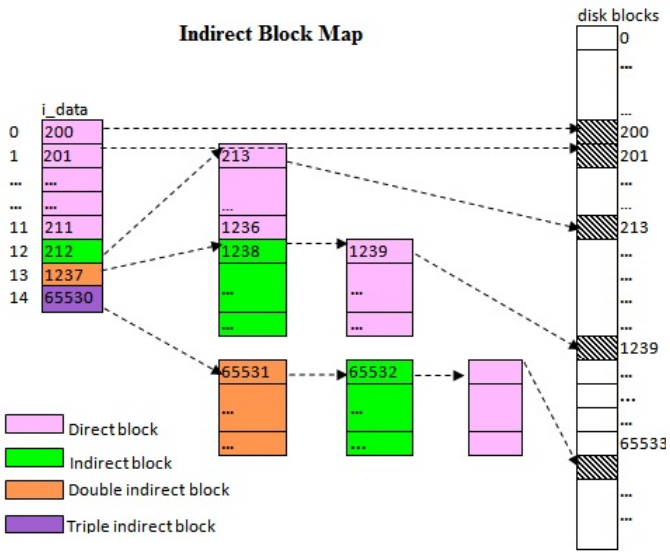

Figure 2.3: Indirect block map [15]

2.3.3 Delayed Allocation

Ext4 implements an optimization technique that delays allocation of physical blocks on the disk until the data is ready to be written on the disk. The major benefit of this technique is that delaying allocation provides the opportunity to combine multiple block allocation requests into a single request and also avoids unnecessary block allocations for files that have a short life span.

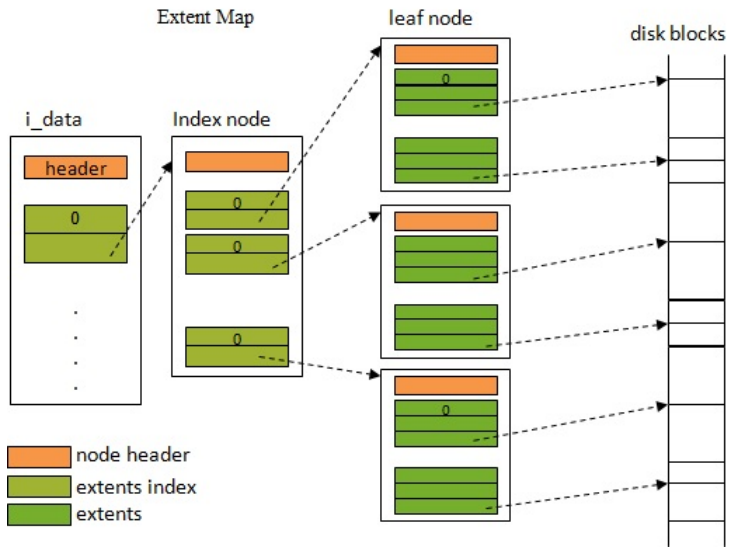

Figure 2.4: Ext4 extent map [15]

Moreover, this scheme avoids fragmentation by allocate a contiguous space on disk and writing to disk in contiguous chunks.[7]

2.3.4 Multi-block Allocation

Ext4 uses a block allocator that allocates multiple blocks at a time, making them much more likely to be contiguous on disk (which will result in faster sequential read operations). Moreover, allocating multiple blocks at a time requires many fewer calls to the block allocator, resulting in faster allocation and reduced processing time.[7]

2.3.5 Flexible block groups

Ext4 implements a feature called flexible block groups (flex_bg) that combines several block groups in to one logical block group. The first block group of the flex_bg holds data block bitmaps, Inode bitmaps and Inode tables of all other block groups in the flex_bg. The effect of this is to group the block metadata close together for faster loading and to enable large files to be continuous on disk by creating a large logical block group.

2.3.6 Journal checksumming

Journalling is a mechanism used to make sure that a filesystem remains in a consistent state in the case of a failure such as a system crash. But even with journaling, corruption is still possible if erroneous entries find their way into the journal. To combat this, ext4 implements checksumming of the journal to

ensure that only valid changes are ultimately written to underlying file system[7].

Ext4 supports multiple modes of journalling, including writeback mode, ordered mode and journalled mode. In writeback mode, only metadata is journalled, which ensure consistency of the metadata but not consistency of the data itself. Ordered mode maintains the consistency on both data and metadata by forcing the data blocks to be written out before the metadata blocks are committed. The last type, journalled mode, maintains both data and metadata consistency by logging both metadata and data and only writing the journalled data after the transaction is committed. Even though journalled mode is the most reliable of all in ensuring consistency it is not recommended because of its large performance requirements[7].

2.3.7 Online defragmentation

Although Ext4 incorporates features to reduce fragmentation within the file system (for example, extents for sequential block allocation), some amount of fragmentation will occur over time. For this reason, an online defragmentation tool exists to defragment both the file system and individual files for improved performance. The online defragmenter is a simple tool that copies files into a new Ext4 Inode that uses contiguous extents for storing the file's data[7]. Unfortunately, the defragmentation tool not yet ready to be used in a production environment and so will not be considered in this research.

2.4 Filesystem Benchmarks

Filesystem benchmarking demands careful consideration in order to properly measure the performance of filesystems and accurately measure their performance under different work environment. Since different filesystems are developed with different intentions, one filesystem is not expected not be the best for all kinds of workloads.

According to Traeger Avishay et al[30] filesystem benchmarks can be categorized into three types: macro-benchmarks, trace replays and micro-benchmarks[30].

- Macro-benchmarks exercise multiple file system operations, and are usually good for an overall view of the system's performance, though the workload may not be realistic.
- Trace-Based benchmarks consist of replaying traces. They can also provide an overall view of the filesystem performance. Traces are designed to be a representative real-world workload. However, it is vital that the trace be in fact representative of that workload (e.g., the trace should capture a large enough sample), and that the method used to replay the trace does not alter its features.

- Micro-benchmarks exercise only a few (usually one or two) operations. These are useful if one is measuring a very small change to better understand the results of a macro-benchmark, to isolate the effects of specific parts of the system, or to show worst-case behaviour. In general, these benchmarks are more meaningful when presented together with other benchmarks.

2.5 Related work

Jan Kara and co-workers[31] undertook a comparative study of the Btrfs, Ext4, XFS and Ext3 filesystems. The experiment was performed on a two-core CPU in a single SATA drive running the 2.6.29 Kernel and with a RAID system. They made the performance comparison without including any of the features that makes Btrfs unique except that of the copy-on-write feature. One of the results of the test performed on a single SATA drive shows that Btrfs takes 10% less time than Ext4 to perform the task of creating 30 kernel trees. Another test on similar setup, reading 6400 files within a directory, Btrfs shows better results than that of Ext4, although it was not as good as XFS. The third test done on the single disk setup shows that Btrfs outperforms Ext4 in a 100 thread synchronous writing of 100 files. They also reported that, in the RAID setup experiment, turning on the copy-on-write feature of Btrfs causes the performance to degrade; with a test of random writes using 16 threads using the default copy-on-write feature of Btrfs, Ext4 outperformed Btrfs

Dominique A. Heger[32] made a performance comparison among the Btrfs, ZFS and Ext4 filesystems by using the Flexible FileSystem Benchmark(FFSB) IO benchmarking set. The experiment was done on both a single disk and a RAID setup consisting of 8 Seagate driver with (Linux kernel 2.6.30). One of the major findings was that the Ext4 filesystem outperforms the others on the sequential read and mixed workloads for the single disk. Ext4 showed similar performance results with that of Btrfs for the sequential read, sequential write, random read/write and mixed tests conducted. The paper also stated that conducting the test with the nodatacow and nodatasum features of Btrfs, which turn off COW and data checksums, gained only a small improvement on the achieved throughput.

Chapter 3

Approach and Methodology

The problem statement for this project requires comparison of Btrfs and Ext4 filesystems on a single disk as well as Btrfs logical volumes with Ext4 and the Linux LVM. In addition, it will also compare the efficiency of the Btrfs filesystem compression tool with that of the Ext4 filesystem with compression software (i.e. bzip2) and also the efficiency of the Btrfs filesystem defragmentation tool.

To perform the I/O throughput performance comparison, one should use a filesystem benchmarking tool that is capable of showing how both filesystems perform under different work load. Basically there are two options: using real application and/or synthetic benchmarking tools. Real applications are more advantageous to use, especially if it is the type of application that is intended to be used with the filesystem since this will imitate the exact situation in the real environment. However, the problem associated with this type of benchmarking is the difficulty of finding such a representative real application[30].

The second alternative is using synthetic benchmark tools that are designed to simulate different workloads. Synthetic benchmarks are mostly flexible and have different parameters that can be adjusted for specific requirements. However, the problem with synthetic benchmarks is that they do not measure any real work. For example, the synthetic benchmark might add overhead that does not exist in a real application. On the other hand, a real application might incur overhead not modelled in the benchmark[30]. Wasim Ahmad Bhat et al.[27] specify that the ideal benchmark for file and storage systems combines the ease of use of synthetic benchmarks with the representativeness of real workloads.

Combining both Synthetic and real application benchmarking for filesystem I/O throughput measurements will produce a more representative result rather than solely depending on either of the two types of benchmarking tools. This project implements both synthetic benchmarking tool Iozone as well as real application tests by file and Directory read/write tests as well as using the Gaussian 09 application.

3.1 Experimental Setup

The experiments were conducted on a Dell Optiplex 745 system with an Intel(R) Core(TM)2 CPU 6600 with a clock speed of 2.40GHz. The Debian 6.0 Operating system was used with kernel 2.6.32. The system contains three 80GB hard disks. The first hard disk (sda) is used only to host the operating system while the other two hard disks (sdb) and (sdc) are used entirely for the experiment. The following table shows details about the hardware and software environments used for the experimental setup.

Component	**Model**
Computer	Dell Optiplex 745
CPU	Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz
Memory	4GB
System HDD	WDC WD800ADFS-75SLR2 ATA Device (80GB)
Benchmarked HDD	WDC WD800ADFS-75SLR2 ATA Device (80GB)
Benchmarked HDD	WDC WD800ADFS-75SLR2 ATA Device (80GB)

Table 3.1: Hardware Specifications

Name	**Version**
Debian Linux 6.0	Kernel 2.6.32.5
	Kernel 2.6.38 [1]
Ubuntu 10.04	Kernel 2.6.32-38 [2]
Iozone	3.308
Seekwatcher	0.12
Gaussian 09	09
btrfs-tools	0.19+20100601-3

Table 3.2: Software Specifications

Both of the experimental disks are divided into 3 equal sized partitions. Care was taken to ensure that all partitions have exactly the same boundaries on the two disks. One partition on each disk is used to create a Btrfs and an Ext4 filesystem while the remaining two partitions on both disks are used to create an Ext4 filesystem with Linux logical volume and a Btrfs volume respectively.

The following table displays the partition layout of the experimental hard disks.

[1] This kernel version is used only for Btrfs Lzo Compression option

[2] This OS is used only for Btrfs defrag test

Disk Partition	Partition Size	Filesystem
sdb1	25G	Ext4
sdc1	25G	Btrfs
sdb2 + sdc2	50G	Ext4 + LVM
sdb3 + sdc3	50G	Btrfs + Volume

Table 3.3: Experimental Hard Disk Partition Layout

3.1.1 Benchmarking Tools

The following subsections discuss the selected benchmarking tools for this project in detail.

1. Iozone is used test the I/O throughput for sequential read, sequential write and re-write, random read, random write and strided read test types. All of the selected test types are executed with record sizes ranging from 64K to 16384K and file sizes ranging from 8192K to 8388608K. These tests are done for a single partition as well as for volumes of both filesystems. Moreover, the I/O throughput of Btrf's compression mount options are also tested using this tool.

2. File and Directory Copy operations are used to perform sequential read and write test by reading and writing an 11GB file and a directory tree having a size of 499MB and 2254 subdirectories.

3. Compression efficiency testing is performed on files with a sizes of 15GB, 6GB and 200MB and also with a directory trees having sizes of 1019MB and 489MB. Compression is performed with the bzip2 software in the Ext4 filesystem. Btrfs transparent compression option efficiency is tested by enabling its compression feature.

4. The Gaussian 09 computational chemistry package is used to benchmark the I/O throughput of both filesystems by executing a task that demands a very large amount of sequential disk I/O, and the achieved input and output levels as well as elapsed time to perform the task are compared for both filesystems.

5. A Perl script is designed to create fragmentation on a freshly created Btrfs file system by randomly creating and deleting files of varied sizes. The number of fragmented files and their percentage out of the total number of files created in the disk partition is recorded, and the Btrfs defragmentation tool is used to defragment the partition. The number of executions of the defragmentation tool and the percentage reduction achieved on each run are recorded until the files in the filesystem is totally defragmented.

Iozone Benchmarking Tool

The Iozone benchmarking tool is used to measure filesystem performance using different test types. It tests file I/O performance for various operations: read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, random write, pread, and specialized tests like mmap, aio_read, aio_write operations [22]. Iozone is a widely-used filesystem benchmarking tool that has been used in a variety of systems. One of its benefit is the number of available options to select metrics that are most appropriate for the specific type of application to be used on the filesystem.

The following are the major metrics that needs to be adjusted while performing the different test types. More information about available parameters and usage can be found by executing the command: man iozone.

- Minimum and maximum file size used for the test
- Minimum and maximum record size used for the test
- Type of test (i.e., random read/write, sequential read/write etc.)

The basic idea behind the Iozone benchmarking tool is breaking up the given file size into chunks of a given record size so that records are written or read in some manner until the given file size is reached. One of the options provided by Iozone is its ability to store the output of the performed test in Microsoft Excel file format which is suitable for performing further analysis on the acquired data. The resulting file contains a tabular form of data where the rows represent the record size (transfer size) and columns indicates size of the file to be tested. The data within each cell is the I/O throughput in Kbytes/sec achieved in performing the specified test type with the corresponding record and file size combination. The following partial output is an example execution of Iozone:-

Iozone sequential read test output

```
# iozone -a -i 0 -i 1 -y 64 -q 16384 -n 8192 -g 8G -b /mnt/parb1/seqred.xls
        64       128       256       512      1024      2048      4096      8192     16384

8192   3737270  3688723  3782110  3653812 3229054 2394606 1893744 1762507

16384  3766316  3794181  3806792  3738857 3192571 2393255 1849445 1667268 1653786

32768  3807088  3844682  3856008  3782986 3264443 2373819 1816514 1622085 1601314

65536  3864617  3895838  3901866  3825782 3303391 2385889 1815048 1605015 1580968
.....
```

The above partial output of the Iozone benchmarking tool displays the achieved I/O throughput for all combinations of record sizes of 64K to 1024K and file sizes of 8192K to 8388608K (8GB). This test is intended to perform sequential read test type on the a selected filesystem. The options -i 0 -i 1 specify the type

of test to be performed. In this example the selected test type is sequential read (-i 1). The option (-i 0) needs to be provided for every test types since Iozone first creates its own file and makes it available for other test types to use it. The other options, -y and -q, allow one to specify the minimum and maximum record size (transfer size). Options -n and -g will enables one to specify the minimum and maximum file sizes to be used for the test.

After acquiring the resulting output file, one can compare the achieved throughput for different record and file size combination for the two filesystems in order to identify the better performer (i.e., which one provides higher throughput per Kbyte).

The test types selected for this project are random read, random write, sequential read, sequential write/re-write and read strided. These tests are selected because they are general file operations that are performed by any application utilizing any kind of filesystem.

- Write: This test measures the performance of writing a new file sequentially.
- Re-write: This test measures the performance of sequentially writing a file that already exists.
- Read: This test measures the performance of sequentially reading an existing file.
- Random Read: This test measures the performance of reading a file with accesses being made to random locations within the file.
- Random Write: This test measures the performance of writing a file with accesses being made to random locations within the file.
- Strided Read: This test measures the performance of reading a file with a Strided access behaviour. An example would be: read at offset x for a length of y Kbytes, then seek n Kbytes, and then read for a length of y Kbytes, then seek n Kbytes and so on. Here the pattern used in this test was to read 4 Kbytes and then seek 200 Kbytes, repeating this pattern thereafter.

Blktrace and SeekWatcher

Blktrace is a block layer I/O tracing mechanism which provides detailed information about request queue operations up to user space. It contains three major components: the kernel patch (only required with kernel versions prior to of 2.6.17) and the blktrace and blkparse utilities. The blktrace utility is responsible for transferring event traces from the kernel and store the results in to a file for further processing or direct formatted output. The second utility, blkparse, is used to format events stored in a file or events that are captured from a live run of blktrace[21].

The following is an example blktrace command:

Blktrace command option

```
1 blktrace -o Ext4.trace -d /dev/sdbx
```

The command line arguments -o and -d specify the name of the output file that stores the events and the device that is going to be traced, respectively.

Blktrace produced detailed block layer information for individual I/O. As a result of this, it is very cumbersome and time consuming to thoroughly analyse its output directly. Seekwatcher is an analysis tool that is capable of graphing the result of blktrace output and help one visualize the I/O patterns and performance easily.

The following is an example seekwatcher command:

Seekwatcher command option

```
1 seekwatcher -t Ext4.trace -o Ext4-Trace.png
```

The command line arguments -t and -o specify the name of the blktrace output file and the name of the graph constructed from the specified event trace output, respectively.

3.1.2 Real Application Benchmarking

Even though synthetic benchmarking tools like Iozone are generally good in providing an overview of the performance of the filesystem under different workloads, the results might not totally reflect the real world scenario. In order to find out how both filesystems perform under a real world situations, large file and directory read/write operations and the Gaussian 09 application are used.

Large File and Directory Read/Write Tests

Files of four different sizes and two directories of different sizes and number of subfolders are used. The test will be performed writing to and reading from both filesystems, in the single disk and volume configurations. The Linux time command is used to report the elapsed time, system time and user time taken to perform the read and write operations.

File copy test

```
1 time cp bigfile /mnt/Btrsing
```

Gaussian 09

Gaussian 09 is an electronic structure programs used by chemists and other scientists worldwide. Starting from the fundamental laws of quantum mechanics, Gaussian 09 predicts the energies, molecular structures, vibrational frequencies and molecular properties of molecules and reactions [24]. This application is used for benchmarking because of its capability in creating tasks that require a lot of disk I/O activity, enabling the performance of a filesystem to be measured in a real life situation. The following is a sample command used to execute a job with this application.

Gaussian 09 task execution

```
1 /usr/bin/time -f "%e %S %U %I %O" g09 test.gjf
```

The above command computes the task named test.gif and displays the elapsed time, system time, user time, and the number of input an output operations acquired while performing the given task by Gaussian. Having the amount of disk I/O operations under both filesystems makes one identify how both filesystms function under a real life situation.

The specific task chosen for this work is a calculation which predicts the total energy for benzene, using the coupled cluster method with single and double substitutions (CCSD) and a large, triple zeta basis with two additional polarization functions on both heavy atoms and hydrogens. A full integral transformation is performed and molecular symmetry is not taken into account. Memory use is limited to 1GB. The scratch files for this calculation will total nearly 20GB, and the I/O access patterns will be sequential.

```
"%Mem=1GB
CCSD/6-311G(df,pd) Trans=Full NoSymmetry
```

3.1.3 Custom made fragmentation tool

As mentioned in the problem statement, one of the research questions that is going to be investigated is the effectiveness of the Btrfs defragmentation tool. In order to examine its effectiveness, one needs to have a file system that is already fragmented. Since the experiment is done on a newly created filesystem, a way to age the filesystem needed to be found in order to be able to measure the performance of Btrfs's defragmentation tool.

For this purpose, a Perl script was prepared to create fragmentation on the filesystem. It operates by filling up the entire partition by creating files with different sizes, deleting some of the newly created files and replacing those files with ones having the same name but a different size. This repeated procedure of creation and deletion of files will result in a fragmented filesystem. The

next stage should be finding out the number of fragmented files and the percentage of fragmented files out of the total number of created files. Therefore, the script provides a percentage of file fragmentation by executing filefrag (a Linux command that will display the number of extents of a file by using the FIEMAP ioctl) for each file created by the script, and then summing up the number of files that have more than one extent.

The second stage of this procedure is executing the Btrfs defrag command and finding out how much effective it is in reducing the percentage of fragmentation that was created by the first stage. By recording the amount of time required by the defrag command to defragment the specified partition and finding out the achieved level of reduction in fragmentation percentage of the file, one can identify the effectiveness of Btrfs's defragmentation tool.

The script requires the starting size of the file which will be used to fill up the partition (the actual file size is varied automatically during the operation) and also the unit of measurement (GB, MB or KB). The file size and units are an optional command line arguments in order to make the script adjustable to different partition sizes.

Here is an example of how this script is run:

script task execution

```
1 ./autofrag -s 32 -u m
```

The -s file specifies the size of the initial file to be created, and -u specifies the unit of measurement of the file to be created. This command specifies an initial file size of 32MB.

3.1.4 Compression test

As it has been specified in the problem statement, Btrfs's built in compression facility will be compared against Ext4 together with the bzip2 compression software. For this test, two metrics have used to measure the compression efficiency: by using space Saving formula[33] (i.e 100 - $\frac{compressedsize*100}{Orginalsize}$) and execution time. Different sized text files and directories having a number of subdirectories and a variety of sizes are tested.

Size	Type
15360MB	File
6144MB	File
200MB	File
1019MB	Directory
489MB	Directory

Table 3.4: Compression Test Files and Directories

The above listed test file sizes are selected to measure the efficiency of the compression technique from large size to small size, relative to the size of the partition, which is 25GB. In addition, the directories also varied in size of sub-directory and file size.

3.2 Benchmarking Environment and Repetition of Experiment

The state of the system during the benchmarking process can have a significant impact on the obtained result of the benchmark. Traeger et al[30] states that some of the major factors that can affect results are cache state, filesystem ageing and non-essential processes running during the benchmarking process.

To avoid cache impact of filestems in the Iozone benchmark test, mounting and unmounting of the tested filesystem is done for every consecutive tests. Similarly for file and directory read/write tests and also for the compression tests, a reboot is performed. Both filesystems are mounted by their respective default mount option except for compression tests, which require enabling Btrfs's compression feature. Moreover, all non-essential process are stopped during the test.

It is also recommended that performing the test with an aged filesystem will results in a representative setting of the real world and produce a better result for the performance evaluation. The best way to age filesystems is running a workload based on system snapshots[30]. This process has two major implementation difficulties for this project experiment. First of all, ageing a filesystem by this method is a time consuming process[30]. More significantly, finding a working system that has been using Btrfs filesystem in a production environment in order to take a snapshot for filesystem aging was not attainable, even though the filesystem is declared as production ready. Since the test environment for both filesystems must be similar, performing the benchmarking on a newly created filesystm is the only possible and optimal option for this particular moment.

Repetitions for the various test were performed as follows:

1. The selected test types of Iozone are repeated 10 times on both single disk and volumes and the average is taken.
2. The Directory and File copy test is repeated 10 times on both single disk and volumes and the average is taken.
3. The Directory and File compression test is repeated 5 times on both single disk and volume and the average time taken to perform the test is taken.
4. Blktrace is run independently for both Iozone and the Directory and File copy tests.

3.3 Package Installation

Btrfs-tools user-space utility package installation

Btrfs is supported in the Linux 2.6.32.5 kernel/ Using only requires loading the btrfs module into the kernel through the modprobe btrfs command. Btrfs has a user space utility package called btrfs-tools that is required to work with it. In addition, this package also contains the btrfs-convert utility which is used to convert an Ext filesystem into a Btrfs filesystem. One can install this utility package by using package manager.

Btrfs utility package installation

```
1 apt-get install btrfs-tools
```

seekwatcher installation

Seekwatcher can be installed by extracting it from the archive and making the file executable. Seekwatcher requires additional package which are needed to generate graphs from the block traces created by the blktrace command on devices. In addition to this, the python and numpy modules are also required. The package and modules can be installed as follows.

Installation of packages required by seekwatcher

```
1 apt-get install python python-matplotlib python-numpy
```

Gaussian 09 installation

Installation of Gaussian 09 requires the C shell(tcsh), a directory to copy the binary files and also editing the .baschrc file under the home directory of the user that will be using the Gaussian application to set the relevant environment variables g09user and GAUSS_SCRDIR. The final step is to run the install script: $g09dir/g09/bsd/install.

The GAUSS_SCRDIR environment variable specifies the location of the scratch directory. It should be located on the single partition corresponding to each filesystem when the benchmark job is run.

Chapter 4

Results

This chapter describes the results obtained from the experiment explained in the previous chapter.

4.1 Iozone Benchmarking Tool Test Results

The following sections present the results obtained from the random read/write, sequential read/write, sequential re-write and strided read test types of Iozone. The first part shows the performance of Ext4 as compared to Btrfs with its default options. The second part shows the comparison of the Btrfs default mount options against Btrfs with its compression feature turned on. The default setting results are the averages of 10 runs and are plotted with error bars equal to twice the standard deviation (in each direction).

4.1.1 Single Disk Sequential Read Test Results

The graphs below show a sequential read operation performance of a single disk on Btrfs and Ext4 filesystems. The performance of both filestems are very much similar, and there is no significant difference on the achieved I/O throughput.

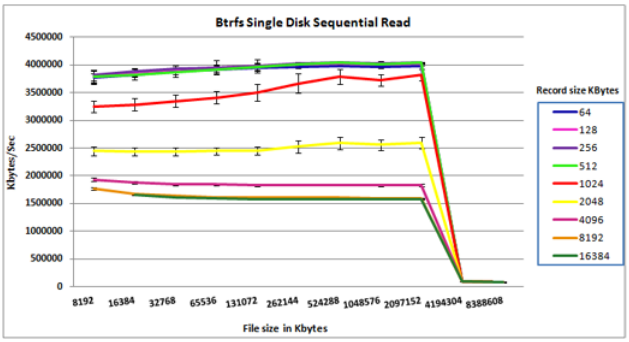

Figure 4.1: Btrfs sequential read I/O throughput (single disk)

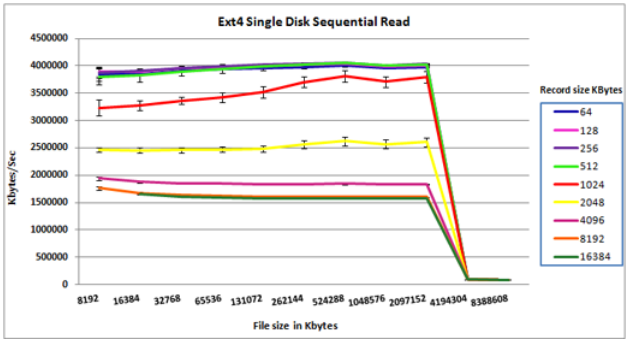

Figure 4.2: Ext4 sequential read I/O throughput (single disk)

4.1.2 Volume Sequential Read Test Results

As can be seen from the graph below, the performance of sequential read operations for both filesystems are very much similar. The difference shown is very an insignificant until the size of the file reached at 4GB, which is the RAM size of the experimental machine.

As can be seen for the graph below and table 4.1 beginning from a file size of 4GB, the attained I/O throughput of the sequential read test on the Btrfs volume shows better performance, nearly double the achieved I/O throughput of the Ext4 volume for the same operation.

Size	FS	64K	128K	256K	512K	1024K	2048K	4096	8192K	16384K
4GB	Btrfs	140953	140451	140889	139651	141917	141150	142120	142189	141681
4GB	Ext4	87154	84065	80521	84285	89092	87829	86299	84336	82526
8GB	Btrfs	121709	121653	122640	122268	121538	121333	122504	121492	122858
8GB	Ext4	79232	77510	73849	80037	77383	73517	80666	77033	74014

Table 4.1: Sequential Read test results for 4GB and 8GB file sizes

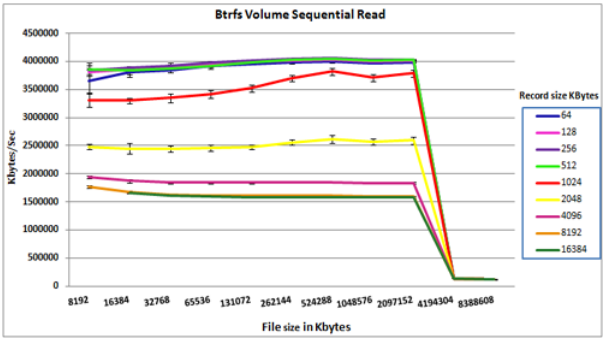

Figure 4.3: Btrfs sequential read I/O throughput (volume)

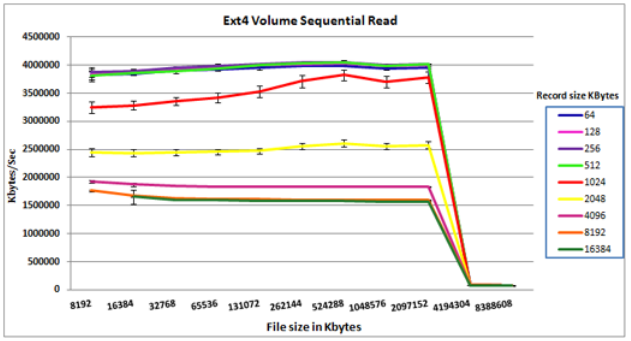

Figure 4.4: Ext4 sequential read I/O throughput (volume)

4.1.3 Single Disk Random Read Test Results

The graphs below depict the achieved throughput of reading a file randomly from a single disk. As it can be seen in the graphs, both filesystems show similar performance for most of file size and record size combinations for this test type. However, when the file size reaches at 4GB, the Ext4 filesystem shows a slight performance improvement over Btrfs . AS it can be seen form Table 4.2 Ext4 performs better for record size $\leq$512K for 4GB and for all record sizes with 8GB files respectively.

File size	FS	64K	128K	256K	512K	1024K	2048K	4096	8192K	16384K
4GB	Btrfs	29862	45610	67587	89618	133351	164270	202062	232706	231864
4GB	Ext4	30583	49100	73035	102941	126021	155998	179979	205764	203142
8GB	Btrfs	8631	13631	19656	23912	33530	46125	63576	66354	79425
8GB	Ext4	10306	17276	26392	41478	46583	57771	72394	80282	86715

Table 4.2: Random Read test results for 4GB and 8GB file sizes(single)

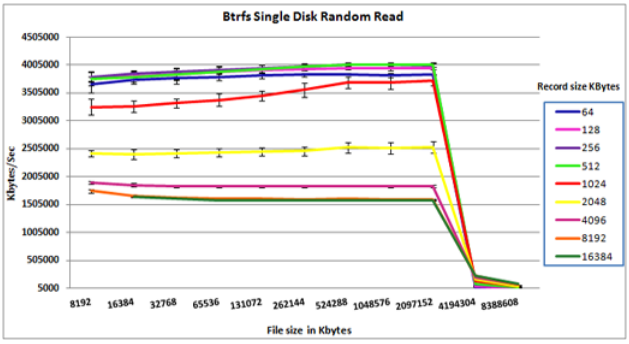

Figure 4.5: Btrfs random read I/O throughput (single disk)

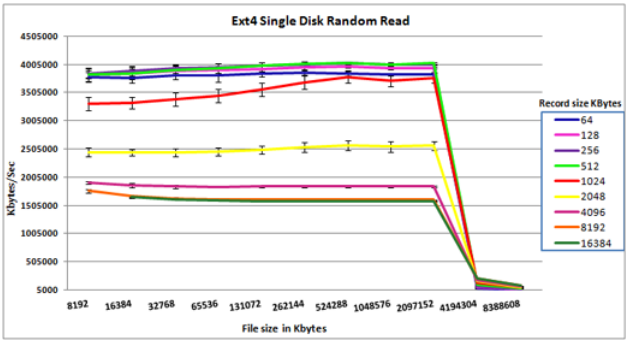

Figure 4.6: Ext4 random read I/O throughput (single disk)

4.1.4 Volume Random Read Test Results

The random read test done on the volume for both filesystems shows similar performance for file sizes less than 4GB. However the performance of Btrfs gets better for the random read test made on the volume with the larger file sizes. Table 4.3 displays that Btrfs performs better for 8GB file as well as for 4GB files with record size $\geq$ 1MB.

File size	FS	64K	128K	256K	512K	1024K	2048K	4096	8192K	16384K
4GB	Btrfs	39367	55799	78822	111113	151220	218362	253405	298121	305337
4GB	Ext4	31000	48682	70946	99353	124468	151550	173513	197291	183999
8GB	Btrfs	10015	15552	23220	31252	39419	62318	81400	92535	113700
8GB	Ext4	10107	16930	25303	40212	44314	53063	68993	73787	76715

Table 4.3: Random Read test results for 4GB and 8GB file sizes(volume)

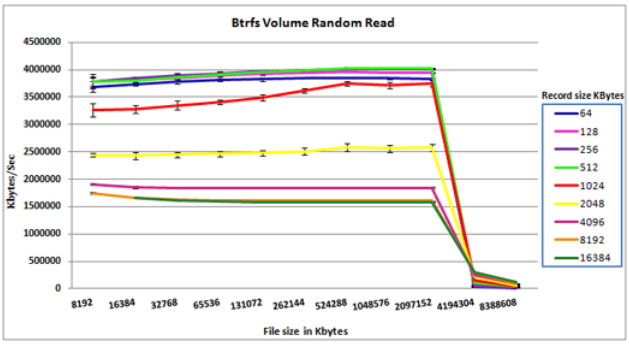

Figure 4.7: Btrfs random read I/O throughput (volume)

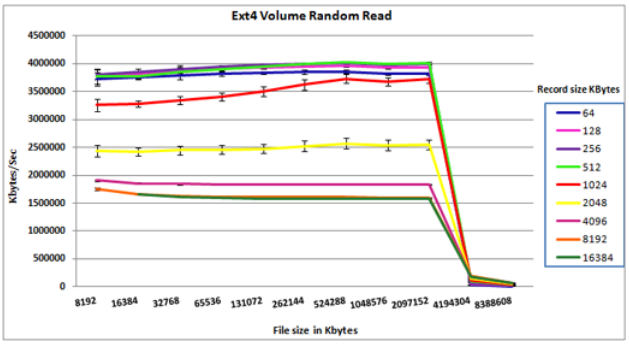

Figure 4.8: Ext4 random read I/O throughput (volume)

4.1.5 Single Strided Read Test Results

The achieved performance of reading smaller files – up to a size of 1GB – in a strided manner is similar to the previously shown sequential read/write and random read/write test results done on a single disk. But when it comes to large file sizes, a different pattern is observed here. For large files with smaller record sizes, Ext4 shows better performance while Btrfs shows improved performance for large file size and record size combinations for the strided read test. As it can be seen from Table 4.4 Btrfs performs better with record size $\geq$1024KB 4GB and 8GB files while Ext4 performs better with record sizes $\leq$512KB for similar file sizes.

File size	FS	64KB	128KB	256KB	512KB	1024KB	2048KB	4096KB	8192KB	16384KB
4GB	Btrfs	45371	51451	59735	62330	114580	158647	181844	208213	229828
4GB	Ext4	60075	62315	87374	110684	168142	142959	167379	185675	199292
8GB	Btrfs	9072	11238	14437	16859	29392	76865	78055	81562	85743
8GB	Ext4	31517	30557	36699	43059	63465	55643	70473	79384	84386

Table 4.4: Strided Read test results for 4GB and 8GB file sizes(single)

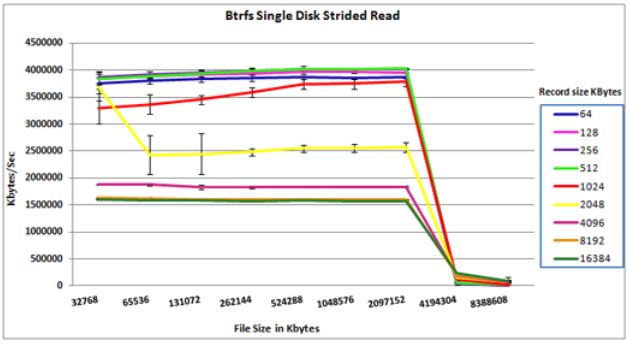

Figure 4.9: Btrfs strided read I/O throughput (single disk)

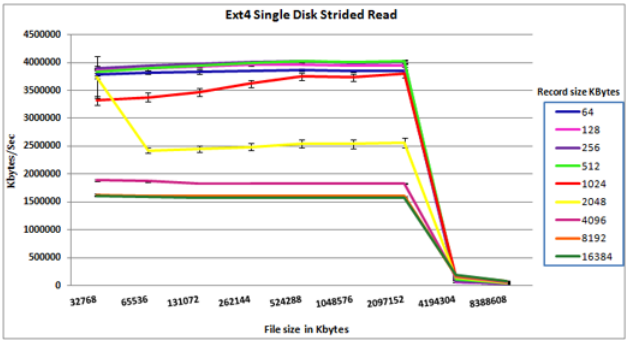

Figure 4.10: Ext4 strided read I/O throughput (single disk)

4.1.6 Volume Strided Read Test Results

The graphs below show the resulting I/O throughput of reading a file in a strided way between the Btrfs and Ext4 Volumes. The achieved performance of test on volumes is similar with that of single disk strided read operation. As it can be seen from the same increment in throughput is shown, with similar patterns exhibited to the single disk strided read test.

File size	FS	64KB	128KB	256KB	512KB	1024KB	2048KB	4096KB	8192KB	16384KB
4GB	Btrfs	58906	58954	68568	82905	154625	208030	231065	277642	311984
4GB	Ext4	57244	82771	92109	103861	159064	139070	157257	177757	181235
8GB	Btrfs	12616	14300	16832	22188	37736	85904	92907	106928	116400
8GB	Ext4	30055	34404	37162	39650	63276	52170	67618	74213	74659

Table 4.5: Strided Read test results for 4GB and 8GB file sizes(volume)

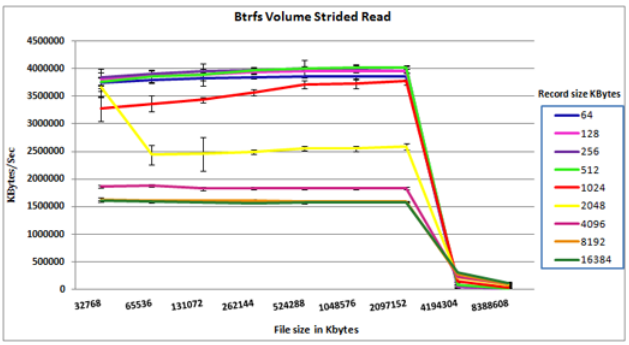

Figure 4.11: Btrfs strided read I/O throughput (volume)

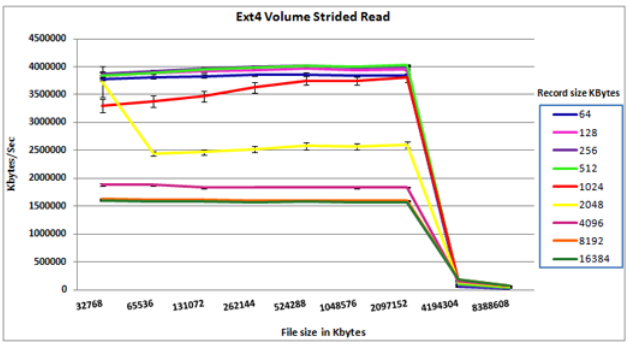

Figure 4.12: Ext4 strided read I/O throughput (volume)

4.1.7 Single Disk Sequential Write Test Results

The graphs below show the achieved throughput of sequentially writing to a file test done on a single disk. For file size less than 1GB Btrfs shows a higher performance as compared with Ext4. In the contrary Ext4 shows better performance with files greater than 1GB. While Ext4 shows a smooth decline of I/O throughput with the increase of file size, Btrfs shows a dramatic decline of performance starting from files with size of 512MB. The effect of buffer cache is exhibited differently for Btrfs and Ext4 i.e 512MB for Btrfs and 1GB for Ext4.

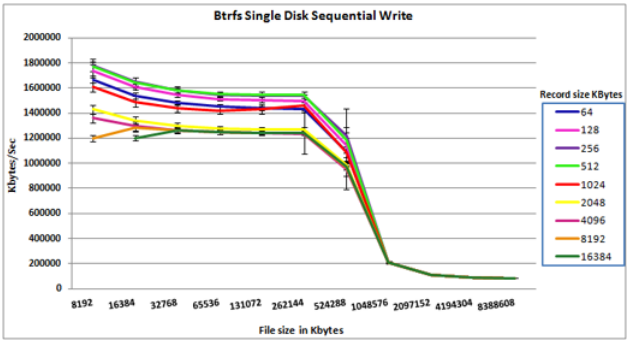

Figure 4.13: Btrfs sequential write I/O throughput (single disk)

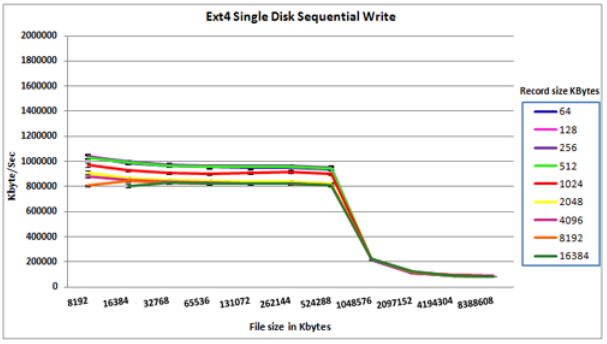

Figure 4.14: Ext4 sequential write I/O throughput (single disk)

4.1.8 Volume Sequential Write Test Results

As it can be seen from the graphs below, Btrfs outperforms Ext4 while writing sequentially writing in to a volume. For Btrfs, better performance is exhibited for all record and file size combinations. However, the error bars on this graph are very large, and so Btrfs performance for any individual test run is quite uncertain.

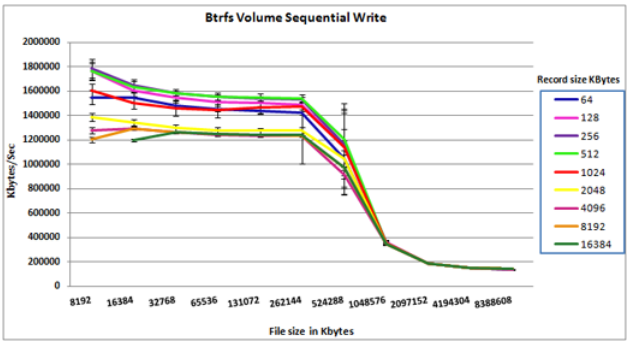

Figure 4.15: Btrfs sequential write I/O throughput (volume)

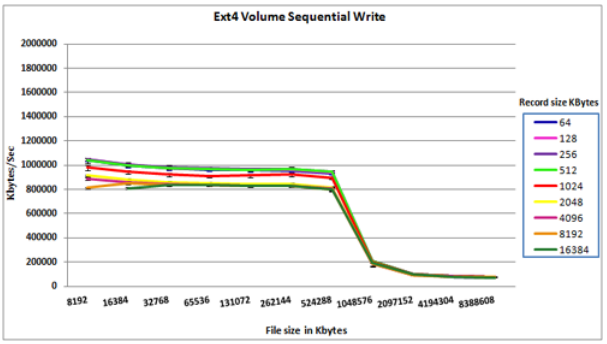

Figure 4.16: Ext4 sequential write I/O throughput (volume)

4.1.9 Single Disk Random Write Test Results

The figure 4.17 and 4.18 below shows the performance of writing to a file in a random manner. Btrfs outperforms Ext4 for all record and file size tests performed. Moreover the gap between the attained throughput of the two filesystems widens with the increase in file size. buffer cache effect comes in to play in the exactly with random write operations, as it has been observed with sequential write operation.

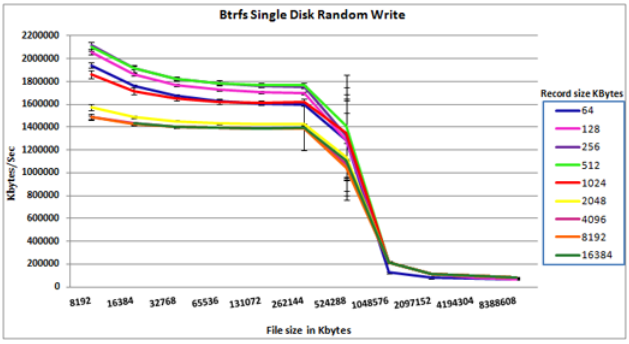

Figure 4.17: Btrfs random write I/O throughput (single disk)

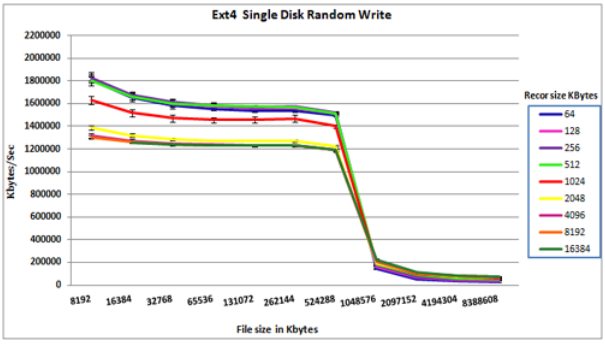

Figure 4.18: Ext4 random write I/O throughput (single disk)

4.1.10 Volume Random Write Test Results

The graph below illustrates that Btrfs performs better for all record and file size combinations in the same way as writing randomly to a single disk. Especially with the increase in file size, the performance gap between two filesytems becomes larger as it has been similarly observed with similar test on a single disk.

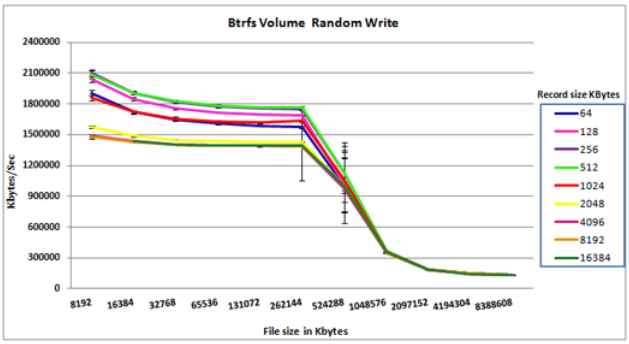

Figure 4.19: Btrfs random write I/O throughput (volume)

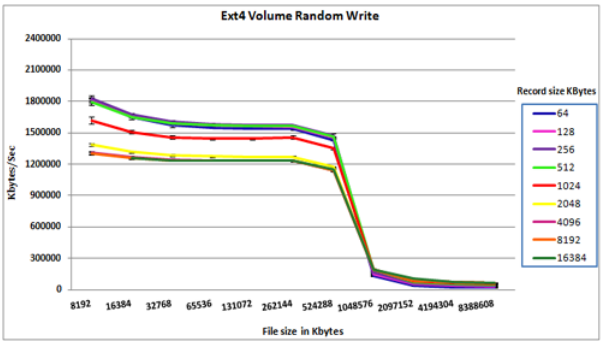

Figure 4.20: Ext4 random write I/O throughput (volume)

4.1.11 Single Disk Sequential Re-write Test Results

As can be seen in the graphs below, Btrfs performs better for small files while Ext4 slightly performs better with large file sizes. The difference exhibited with re-writing sequentially is small compared with the difference that has been observed in the case of writing sequentially.

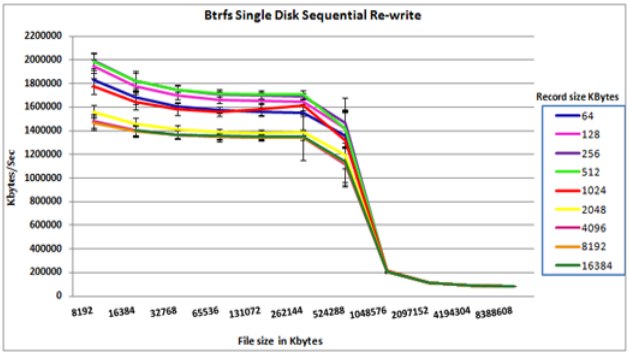

Figure 4.21: Btrfs sequential re-write throughput (single disk)

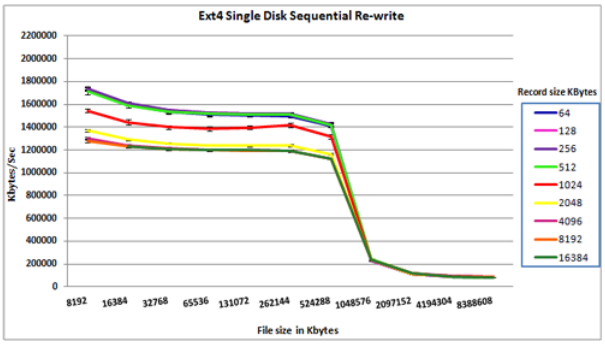

Figure 4.22: Ext4 sequential re-write throughput (single disk)

4.1.12 Volume Sequential Re-write Test Results

The graphs below illustrate the achieved throughput for sequential re-write operations performed on volumes. It shows that Btrfs performs better in all record and file size combinations. Moreover, the difference in the achieved throughput gets larger with the increase in the size of file.

As it can be seen from Table 4.6 the difference on the achived throughput is around 80% on the average for file sizes of 4GB and 8GB.

File size	FS	64KB	128KB	256KB	512KB	1024KB	2048KB	4096KB	8192KB	16384KB
4GB	Btrfs	147347	147428	147484	147093	147249	146794	146682	146680	146714
4GB	Ext4	80464	78050	76596	82690	83147	86126	77368	77898	75972
8GB	Btrfs	133835	133022	133357	133220	132953	132981	132979	132955	132918
8GB	Ext4	76972	75494	71143	76773	75111	71828	77012	73671	72082

Table 4.6: sequential re-write test results for 4GB and 8GB file sizes(Volume)

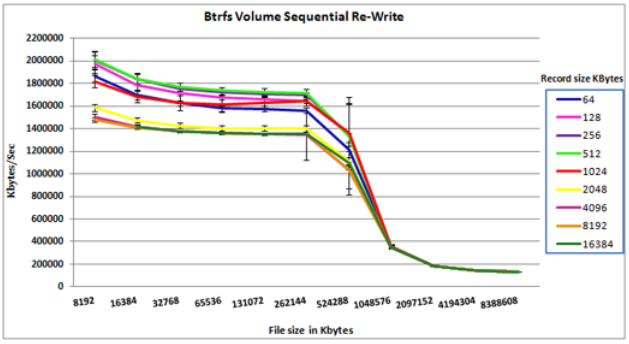

Figure 4.23: Btrfs sequential re-write throughput (volume)

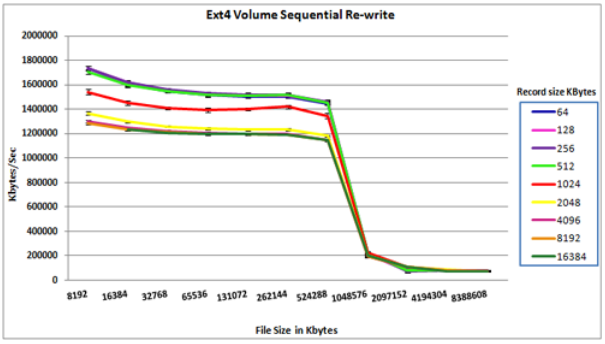

Figure 4.24: Ext4 sequential re-write throughput (volume)

4.2 Iozone Test Results for the Btrfs Compression Feature

For the results presented in this section, results are limited to two record sizes: 128KB and 8192KB.

4.2.1 Compressed vs Default Random Read Performance

As it can be seen from the figure below, the throughput achieved in randomly reading a file wafter enabling the compression feature does not show better performance as compared to the default option for smaller file sizes. But for file sizes that are bigger than 4GB, reading randomly with compression enabled exhibits better performance.

In the same way, enabling the compression feature with Btrfs volumes shows a slight increase in throughput for smaller files and similar higher increases in throughput for the largest file sizes.

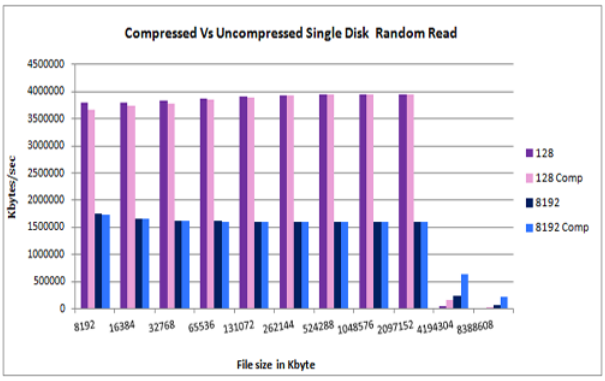

Figure 4.25: Random write I/O throughput (single disk)

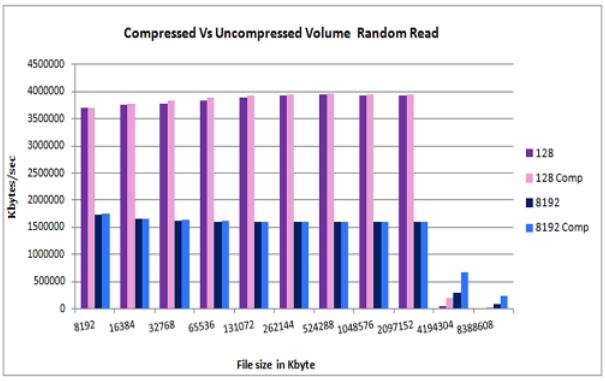

Figure 4.26: Random write I/O throughput (volume)

4.2.2 Compressed vs Default Sequential Read Performance

As it can be seen from the graphs below, the performance of Btrfs with its compression feature starts to outshine the performance of Btrfs with the default options for larger sized files, as it has been similarly shown with randomly reading a file.
However, sequentially reading a file from a Btrfs volume while enabling the compression feature does not show any significant difference in the attained throughput.

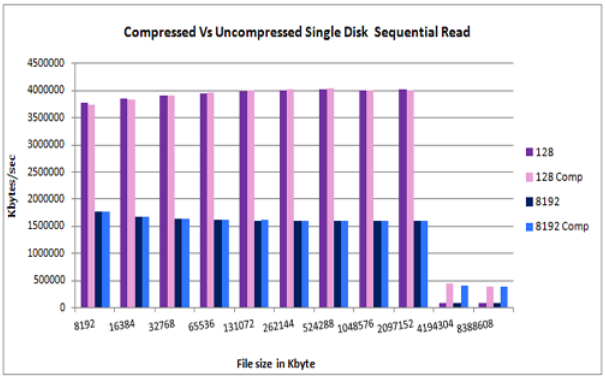

Figure 4.27: Sequential read throughput (single disk)

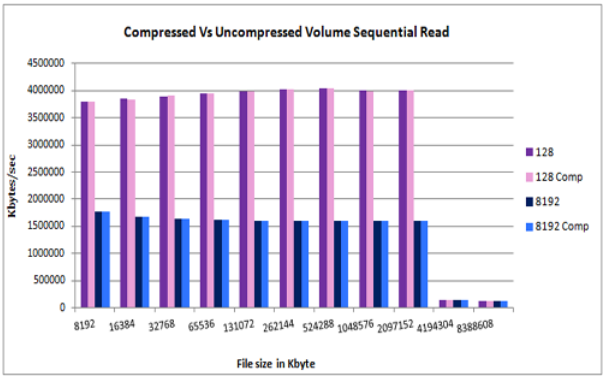

Figure 4.28: Sequential read I/O throughput (volume)

4.2.3 Compressed vs Default Strided Read Performance

Reading a file in a strided manner shows a similar performance difference as has been exhibited with both sequentially and randomly reading a file on a single disk. The figure below shows that the compression feature makes the achieved throughput higher for larger file sizes.
The performance exhibited with Btrfs volume after enabling compression provides higher throughput, as it has been similarly observed for the single disk strided read operations.

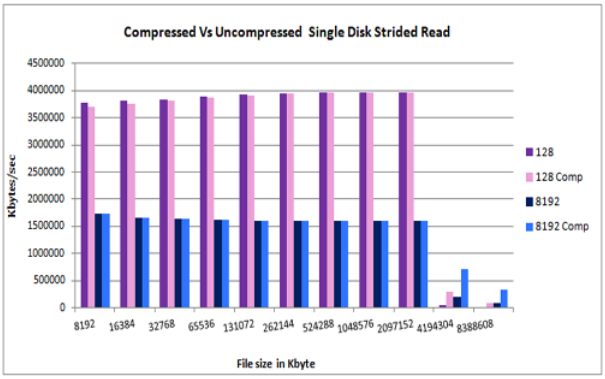

Figure 4.29: Strided Read I/O Throughput (single disk)

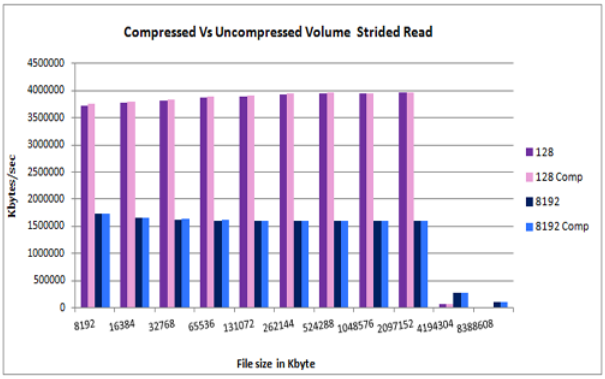

Figure 4.30: Strided Read I/O Throughput (volume)

4.2.4 Compressed vs Default Random Write Performance

As it can be seen in the graphs below, Btrfs without the compression feature enabled shows slightly better throughput for smaller file sizes. However, for bigger file sizes, Btrfs with compression enabled shows better throughput for the single disk random write operation.
Randomly writing to a file after enabling compression feature on Btrfs volume shows much a larger performance improvement for all record and file size combinations.

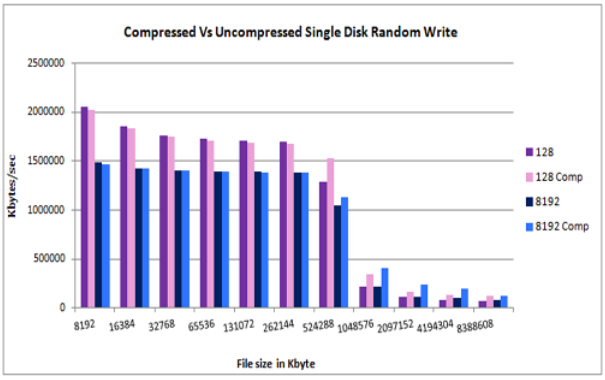

Figure 4.31: Random write I/O throughput (single disk)

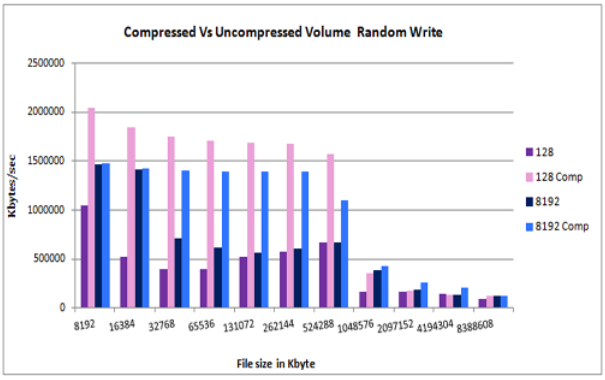

Figure 4.32: Random write I/O throughput (volume)

4.2.5 Compressed vs Default Sequential Write Performance

The graphs below show that sequentially writing to a file in a single disk with compression enabled provides a better performance starting from files of size 1GB. Moreover, the attained throughput difference becomes more bigger with the increment of file sizes.
The difference in throughput for Btrfs volumes with compression feature enabled is not as big as with the one that has been observed with the single disk sequential write operation.

File size	Option	64KB	128KB	256KB	512KB	1024KB	2048KB	4096KB	8192KB	16384KB
4GB	Compress	220350	221315	219533	220631	218369	216508	216710	215677	216657
4GB	Default	90485	91890	91958	92009	91902	92071	91746	91951	92218
8GB	Compress	204102	203502	203828	203489	201853	200304	199714	199648	199794
8GB	Default	83465	85658	85676	85500	85706	85495	85582	85510	85625

Table 4.7: sequential write test results with compression for 4GB and 8GB file sizes(single)

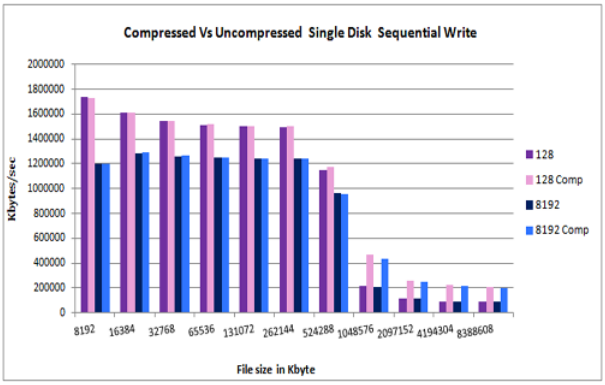

Figure 4.33: Sequential write I/O throughput (single disk)

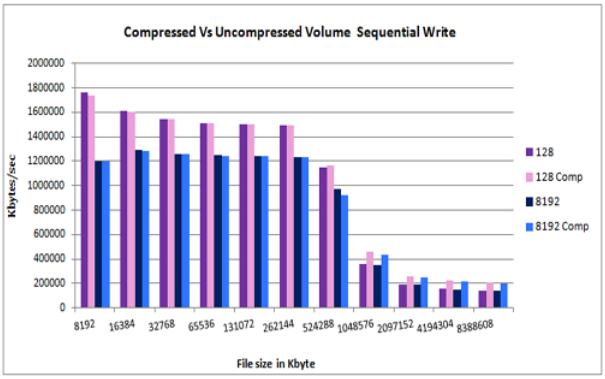

Figure 4.34: Sequential write I/O throughput (volume)

4.2.6 Compressed vs Default Sequential Re-write Performance

The graphs below depict the results for sequentially re-writing to a file. Enabling compression results in better performance for file sizes that are greater than 512MB on a single disk test, while for files with smaller sizes, there is no difference between the compression-enabled and the default performance on this operation.
Sequentially re-writing to a compression enabled Btrfs volume provides better throughput for file sizes greater than 1GB. Similarly, as it has been observed with single disk test results, there is no performance gain in enabling the compression feature on for files with sizes less than 1GB.

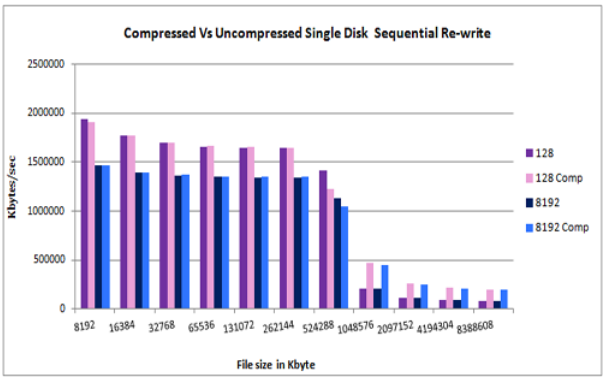

Figure 4.35: Sequential re-write I/O throughput (single disk)

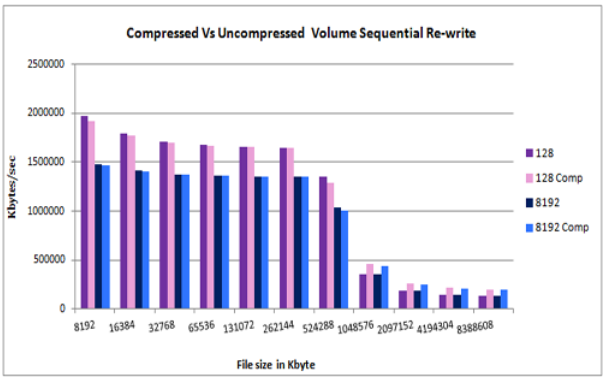

Figure 4.36: Sequential re-write I/O throughput (volume)

4.3 Directory and File Read/Write Test Results

The following section presents the results of performing tests of reading and writing a large file and directory tree. The test is done for Btrfs and Ext4 LVM single disk filesystems and volumes. The tests were performed both with the Btrfs compression option and with the default mount option.

4.3.1 Directory Read and Write Test Results

Figure 4.37 shows the elapsed time required to perform the directory read operation, and it clearly shows that Btrfs performs better in both the single disk and volume environments. In contrast, Ext4 outperforms Btrfs in writing a directory for both the single disk and the volume.

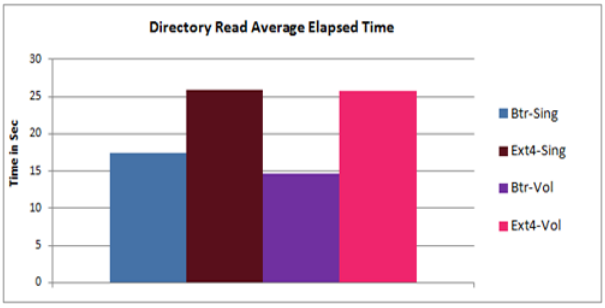

Figure 4.37: Directory Read Average Elapsed Time

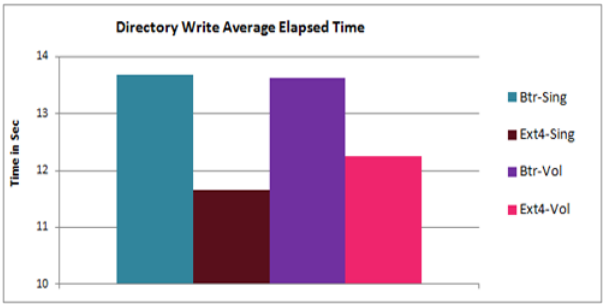

Figure 4.38: Directory Write Average Elapsed Time

4.3.2 File Read and Write Test Results

As can be seen in the graphs below, Btrfs performs better for reading a file from a single disk. However, the performance of reading a file from a volume differs little from the Ext4 LVM volume. For the test of writing a file into a volume, Btrfs outperforms Ext4 whereas for a single disk, Ext4 performs better than Btrfs.

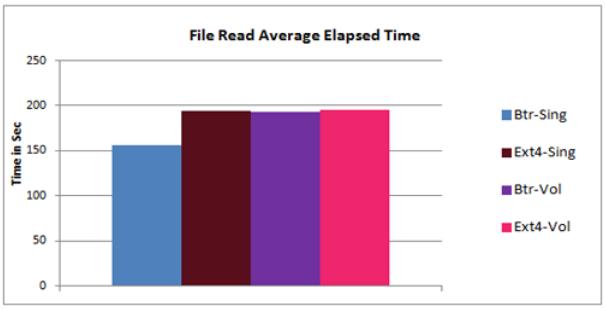

Figure 4.39: File Read Write Average Elapsed Time

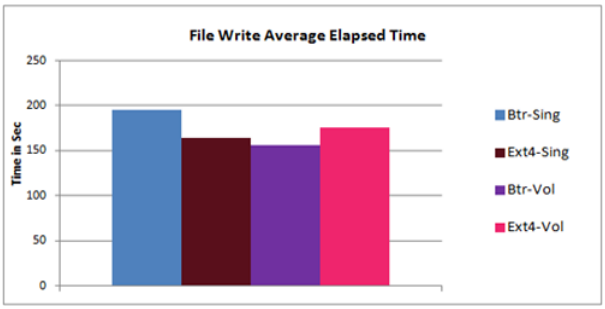

Figure 4.40: File Write Average Elapsed Time

4.3.3 Directory and File Read/Write with the Btrfs Compression Feature

The following section shows the results obtained from file and directory read/write tests while enabling the Btrfs compression option. Since the kernel that comes with Debian Squeeze 6.0 does not support the LZO compression option of Btrfs, the results shown below are obtained from the default compression option (i.e., Zlib).

Compressed vs Default Directory Read/Write Test Results

As it can be seen in the graphs below, enabling the compression option for the directory read test on a single disk results in higher performance compared with the default. However, when the test was performed on the Btrfs volume, this difference did not appear.

In contrast, the test results from reading a directory (Figure 4.41) shows that the performance of writing a directory is much better with the default than with Btrfs compression feature enabled for both single disk and volume.

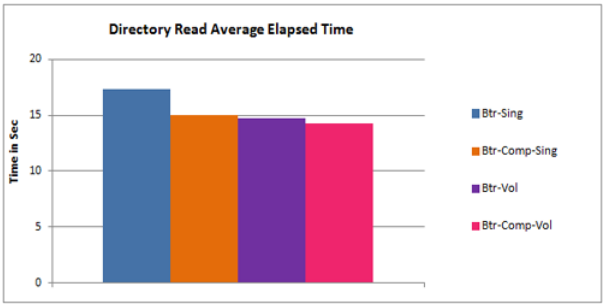

Figure 4.41: Compressed vs Default Directory Read

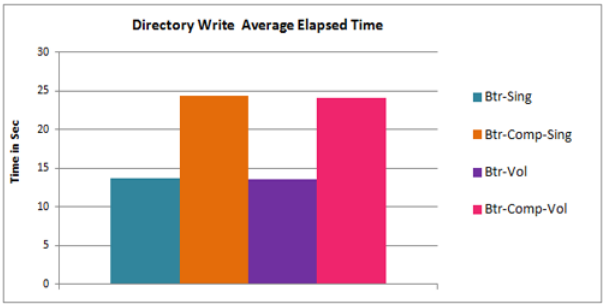

Figure 4.42: Compressed vs Default Directory write

Compressed vs Default File Read/Write Test Results

Figure 4.43 illustrates the file read test results from the compression enabled and default options of Btrfs. Enabling the compression feature worsens the performance of file reading from a single disk as compared with the default. However, the result obtained from file read operation with a volume doesn't show any significant difference from the default option.

In contrast, compression provides higher performance when writing to a file for a single disk. However, the performance of writing to a volume does not show any significant difference as compared to the default one.

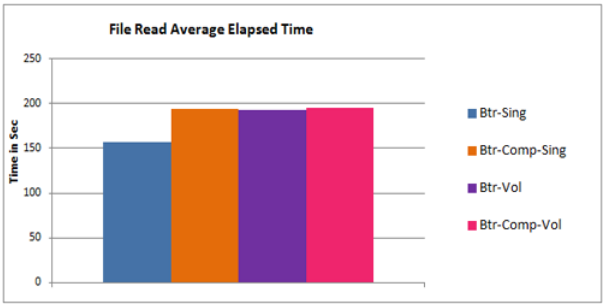

Figure 4.43: Compressed vs Default File Read

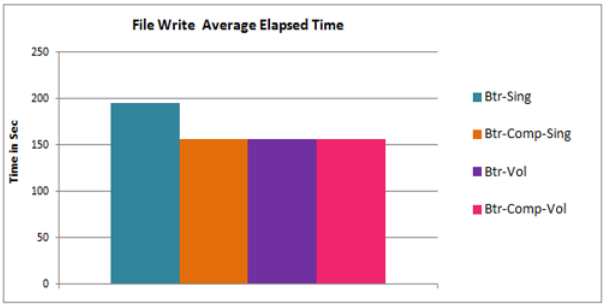

Figure 4.44: Compressed vs Default File Write

4.4 Seekwatcher Test Results

The following sections present the Seekwatcher graphs that are obtained from the block layer tracing done by the blktrace facility. The block I/O traces were taken simultaneously the Iozone tests as well as with directory and file read/write tests.

4.4.1 Iozone Single Disk and Volume Sequential Read

As it can be seen in the graph in Figure 4.45 below, the throughput of Btrfs shows consistency throughout the whole run while Ext4 achieves higher throughput initially and starts to decline after a while. It also shows that the number of seeks performed by Btrfs is much higher than that of Ext4 filesystem for the Iozone sequential read tests performed on a single disk.

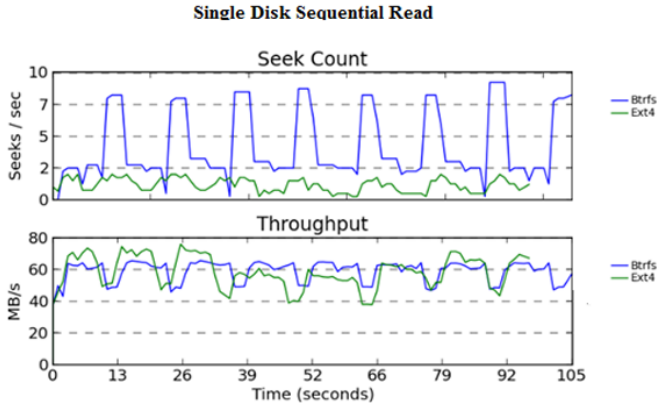

Figure 4.45: Single Disk Sequential Read

For sequential read operations performed on volumes, Ext4 shows higher throughput, and similarly making fewer seeks during the initial stages but increases the rate as the benchmark continues. At the start of the run Ext4, shows lesser throughput performance, but it catches up with Btrs after a little while.

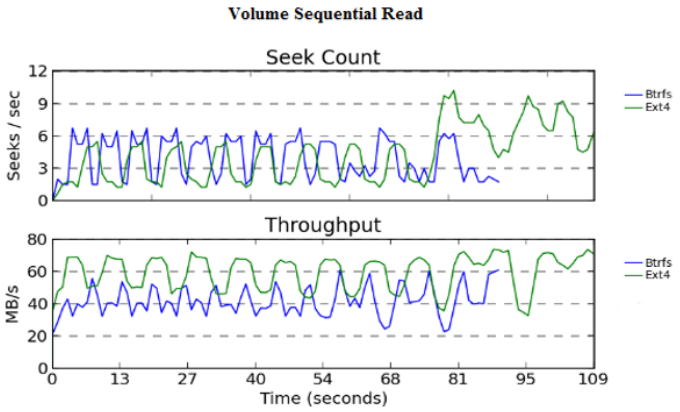

Figure 4.46: Volume Sequential Read

4.4.2 Iozone Single Disk and Volume Strided Read

The results shown in Figure 4.47 displays that both filesystems exhibited similar throughput and disk seek patterns while running this benchmark. Initially Ext4 performs fewer seeks, but after abut 60 seconds, its rate gets higher and appears in a similar state with that of Btrfs.

For strided reading to a volume, Ext4 shows higher throughput and performs fewer seeks than the Btrfs volume.

Figure 4.47: Single Disk Strided Read

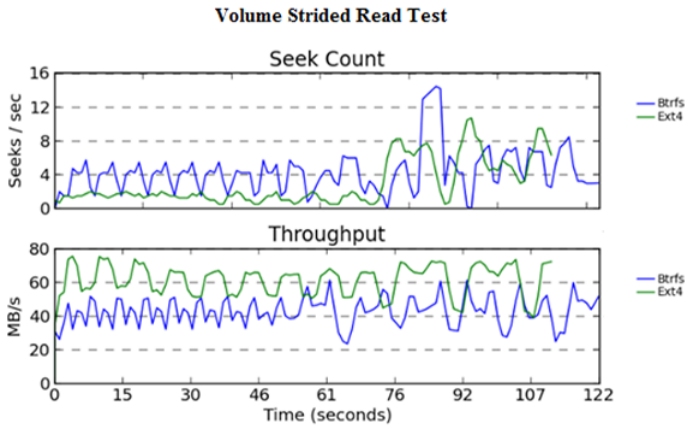

Figure 4.48: Volume Strided Read

4.4.3 Iozone Single Disk and Volume Random Read/Write

As it can be seen in the graph in Figure 4.49 below, even though Ext4 shows better performance initially and at the end of the run, both filesystems shows very much similar throughput most of the benchmarking test run on a single disk. In contrast to the similarity between the achieved throughput, Btrfs perfoms a much larger amount of seeks as compared to Ext4.

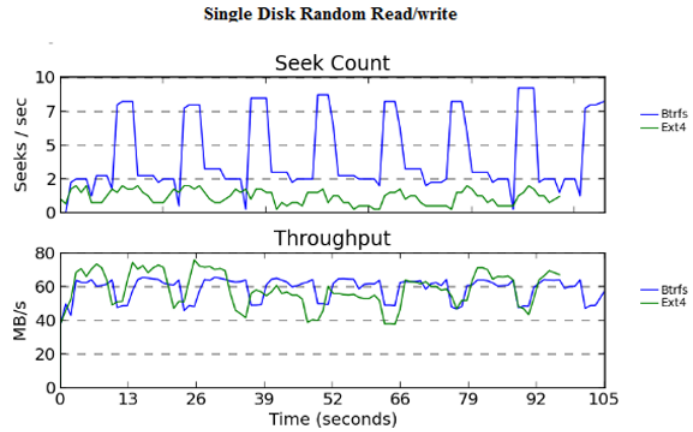

Figure 4.49: Single Disk Random Read/Write

Similarly, in the tests on volumes, Ext4 achieved higher throughput and performed fewer seeks. However, this changes towards the end of the benchmark, and the disk head movement suddenly gets higher and there is an increase in the number of seeks.

Figure 4.50: Volume Random Read/Write

4.4.4 Iozone Single Disk and Volume Sequential Write

For the sequential write benchmarking test run done on a single disk, Ext4 attained similar throughput with Btrfs but with fewer seeks than Btrfs. In contrast, for the case of volumes, Ext4 shows better throughput and a much smaller number of seeks. However, as has been observed with the single disk run of the random read/write test, the number of seek gets higher towards the end of the run for Ext4 LVM volume sequential write operations.

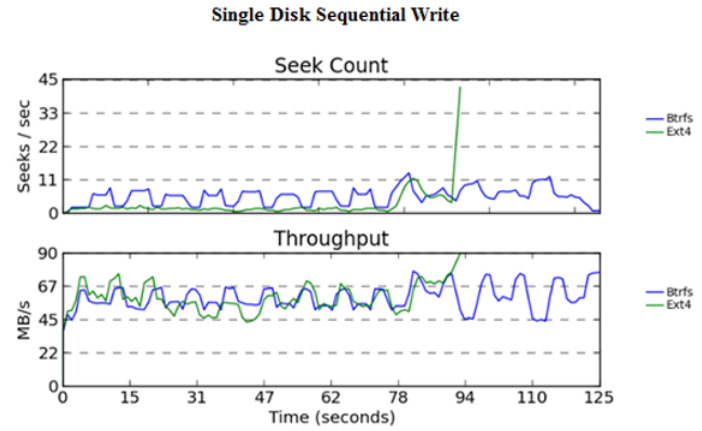

Figure 4.51: Single Disk Sequential Write

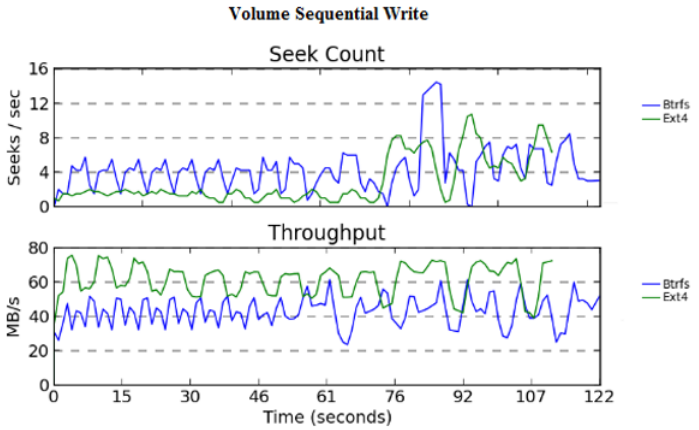

Figure 4.52: Volume Sequential Write

4.4.5 Directory Tree Read/Write Test Results

Figure 4.53 illustrates that Btrfs exhibits a much higher throughput and smaller seek rate in the directory read test run done on a single disk. As can also be seen in Figure 4.54, the number of seeks made by Btrfs is similarly smaller and its achieved throughput is higher for the case of volume.
Although the disk seek rate of Ext4 is much higher for both the volume and single disk cases, Ext4 shows a higher rate of throughput at some stages of the benchmark run.

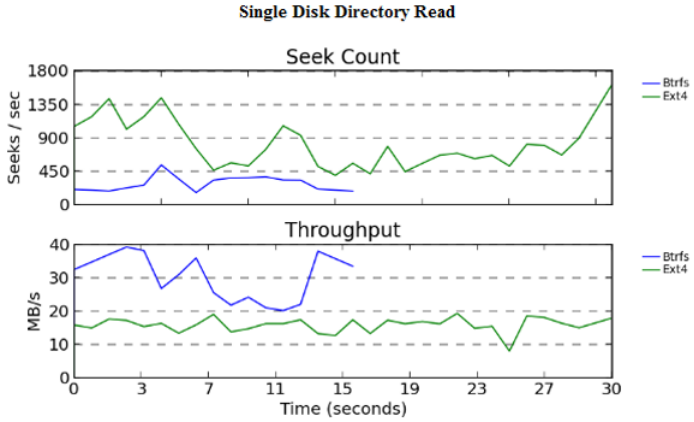

Figure 4.53: Single Disk Directory Read

The figure below shows that the Btrfs seek rate for the directory write operation is too insignificant to be seen in the graph whereas Ext4 displays a higher disk seek pattern. Similarly, the Btrfs throughput is much higher than Ext4, although the performance degrades at one point but then returns to its initial level of high throughput.

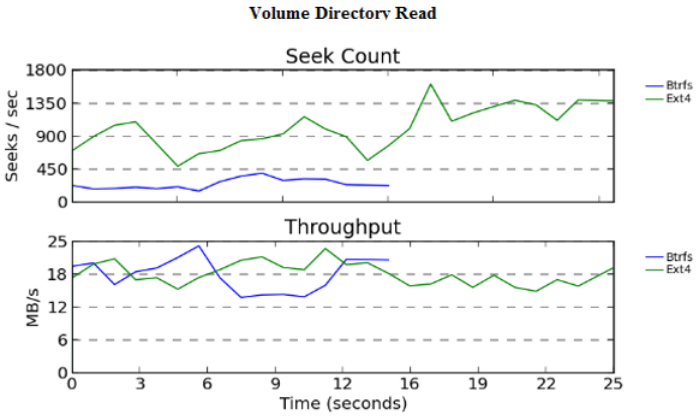

Figure 4.54: Volume Directory Read

In the same way, the throughput achieved for the directory write test run performed on volumes shows that Btrfs achieved higher throughput as compared to the Ext4 filesystem performance, again with almost no seeks.

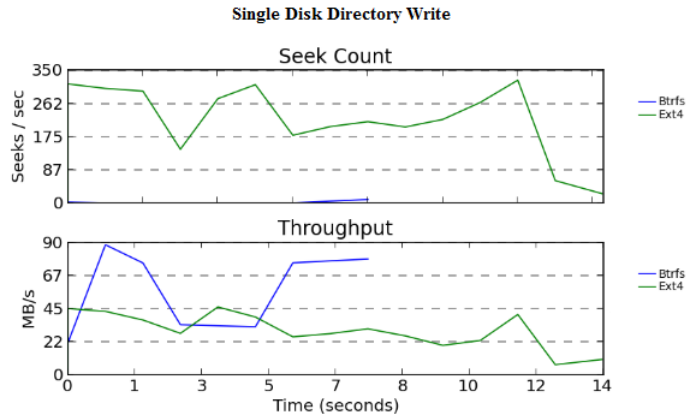

Figure 4.55: Single Disk Directory Write

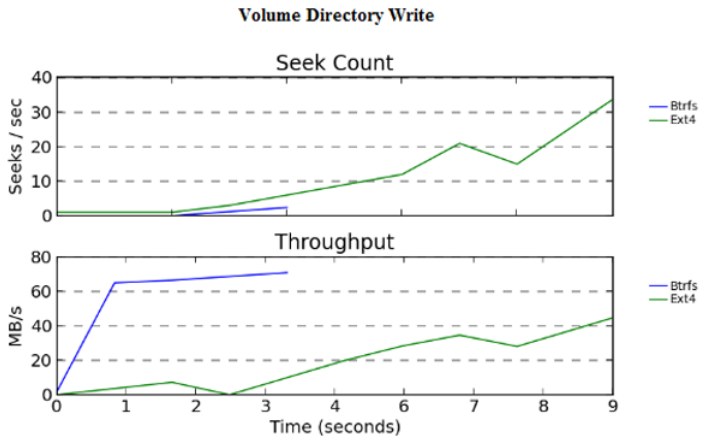

Figure 4.56: Volume Directory Write

4.4.6 11GB File Read/Write Test Results

Figure 4.57 shows that Ext4 achieves similar throughput and fewer seeks in the file read test benchmark run as compared to Btrfs. The disk seeks made by Btrfs are very much more numerous than for Ext4.
For the similar benchmark made on volumes, Figure 4.58 shows that the disk seek rate is similar with the one that is seen with the single disk runs. On the other hand, the achieved throughput for Btrfs is very much lower than the one exhibited on a single disk. Ext4 shows similar higher performance and lower disk seeks for the file read test runs made on volumes.

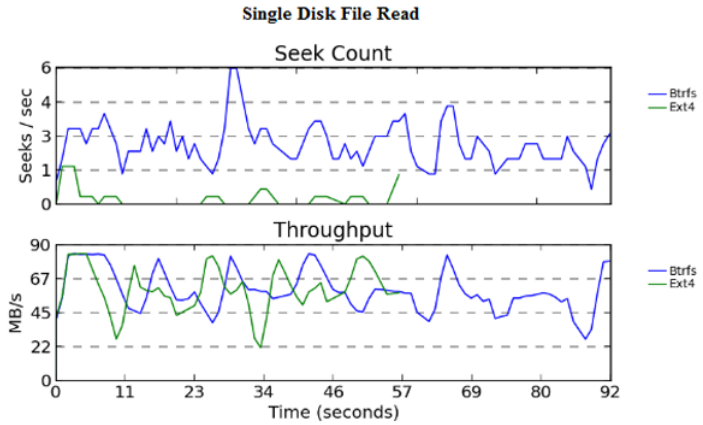

Figure 4.57: Single Disk File Read

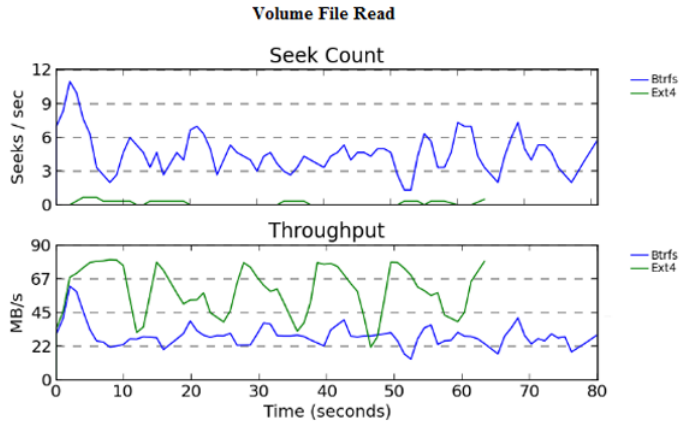

Figure 4.58: Volume File Read

The figure below illustrates that for the file write benchmark, both filesystems show similar disk seek rates and throughput, although Btrfs shows a bit higher throughput at some stage. The similar run made on volumes, shown in Figure 4.60, Ext4 moves the disk head all over the platter. In contrast Btrfs makes only very minimal disk seeks for most of the benchmark run.

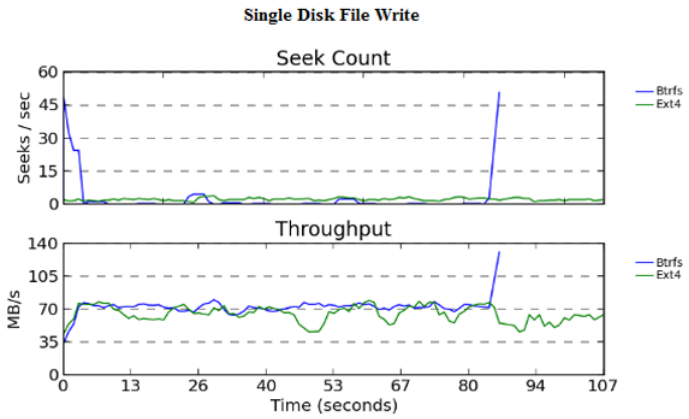

Figure 4.59: Single Disk File Write

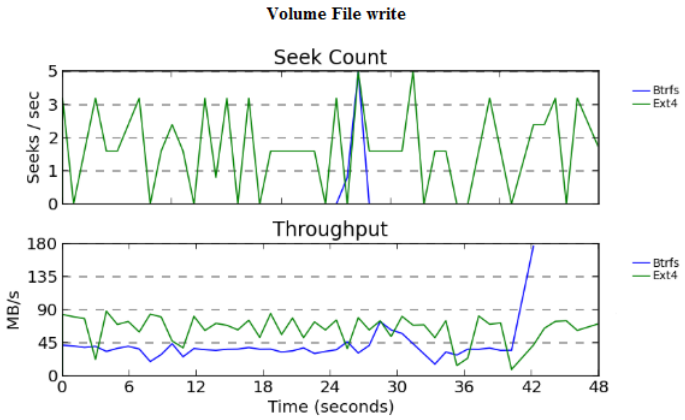

Figure 4.60: Volume File Write

4.5 Computational Chemistry Test Results

Figure 4.61 shows the elapsed and CPU time obtained from Gaussian 09 while performing the same calculation with its scratch files on the two filesystems, in the single disk experimental setup. The graph shows that the Ext4 filesystem has only slightly higher CPU usage – 0.5% – as compared to Btrfs. Nevertheless, the total elapsed time to perform the calculation in Btrfs filesystem is much more higher than the the time taken by Ext4.
As it can be seen in Figure 4.62, Btrfs performs a higher number of I/O operations while executing the Gaussian calculation as compared to the same operation done on Ext4 filesystem.

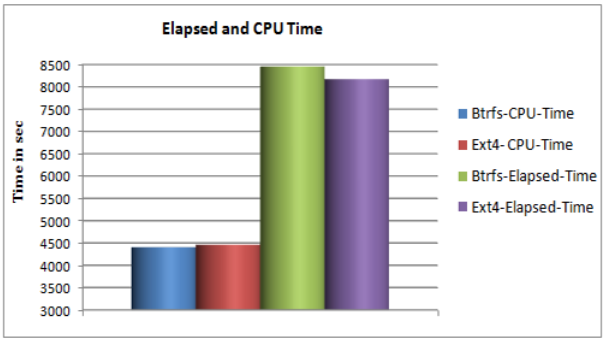

Figure 4.61: Total Elapsed and CPU Time

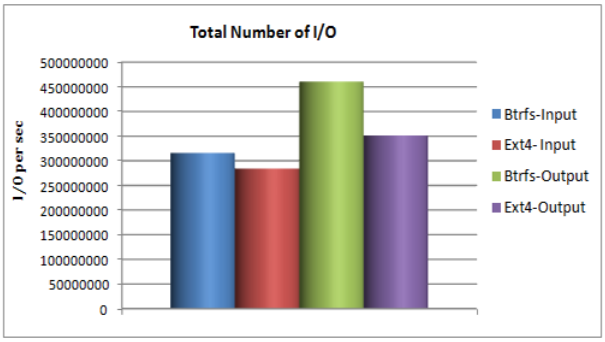

Figure 4.62: Total Number of I/O Operations

4.6 Compression Test Results

The following section shows the test results of file and directory compression with LZO and Zlib compression options of Btrfs and with the bzip2 software with Ext4 filesystem. Since the LZO mount option is not supported with the

default kernel that shipped with Debian Squeeze 6.0, the compression option of LZO is tested with a kernel 2.2.6.38 with the same OS. In addition, the Btrfs compression option was also tested with compress-force mount option.

4.6.1 Compression Times for Files and Directories

As it can be seen from Figure 4.63 and 4.64, the time taken to compress both files and directories is much longer for bzip2 compared with both LZO and Zlib compression features of Btrfs. When comparing the time taken to compress a file between Zlib and LZO, LZO takes lesser time than Zlib to compress a file with a size of 15GB. Similarly LZO takes less time to compress the directory as compared with Zlib.

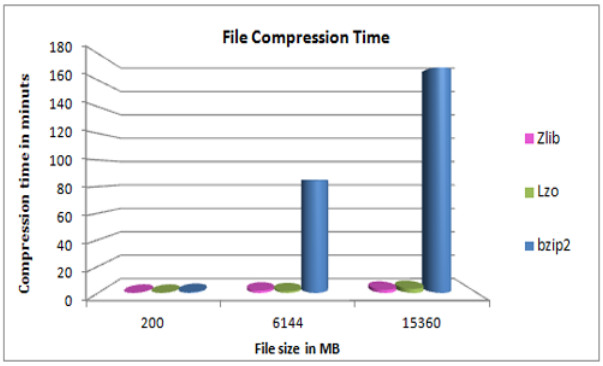

Figure 4.63: File Compression Time

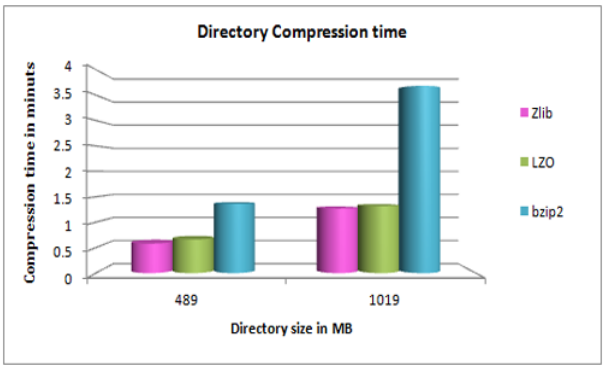

Figure 4.64: Directory Compression Time

4.6.2 Space Reduction Ratios for Files and Directories

Figure 4.67 shows the space reduction gained by the compression mechanisms in compressing files. bzip2 shows more than 90% space reduction while LZO compression shows more than 70% space reduction on compressed files. However, Btrfs Zlib compression only shows a very minimal space reduction for file compression.

For directory compression, bzip2 shows a higher space reduction ration for a directory of size 1019MB containing different file types. In contrast, Zlib shows higher space reduction for directory of 499MB which only contains text files. LZO shows a smaller space reduction for directory compression.

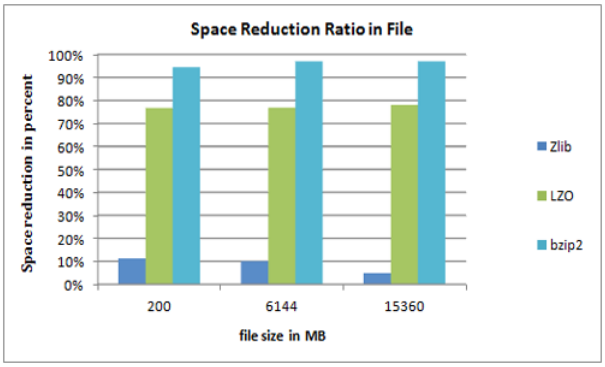

Figure 4.65: Space reduction for file

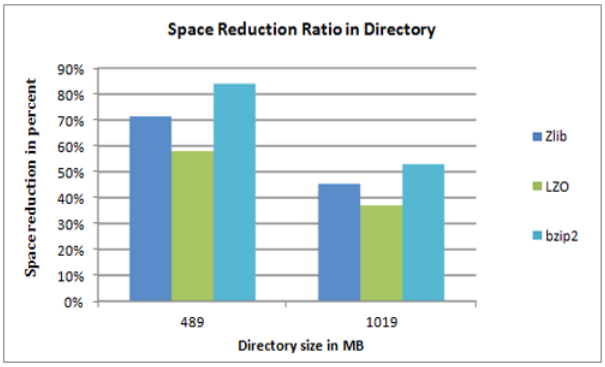

Figure 4.66: Space reduction for directory

4.6.3 Compression Results with compress-force

Using Btrfs's compress-force option forces Btrfs to compress the data no matter how it looks after compression. As it can be seen from Figures 4.67 and 4.68, using this option makes the time taken to compress a bit longer than when the plain compress mount option is given, for both file and directory compressions. However, compress-force results in a big difference on the achieved percentage reduction in size.

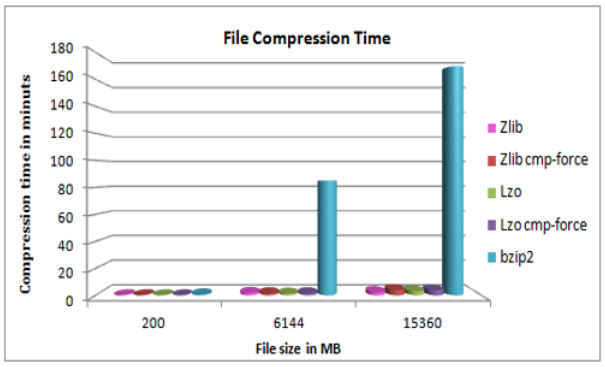

Figure 4.67: File Compression Time with force-compress

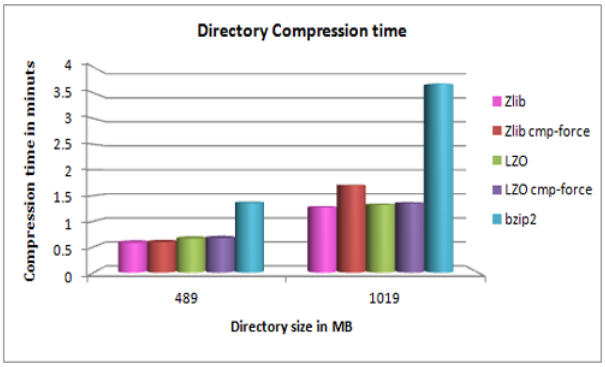

Figure 4.68: Directory Compression Time with force-compress

The reduction of space achieved by the compression option together with compress-force provides a space reduction of more than 80% for both Zlib and LZO file compression. Similarly, it achieves a more than 70% space reduction for the directory with text only files and more than 40% space reduction for the directory containing different file types.

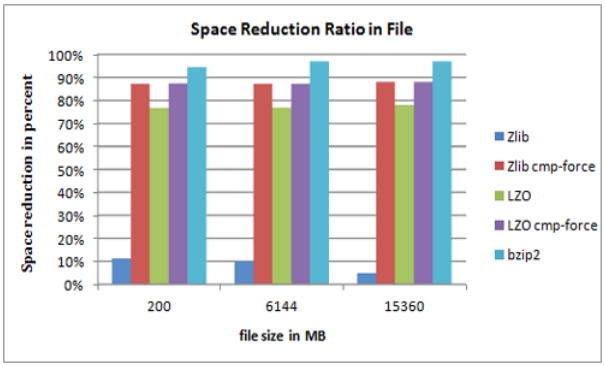

Figure 4.69: Space reduction of files with force-compress

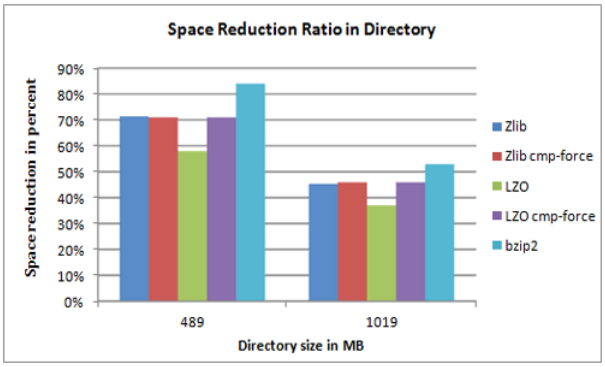

Figure 4.70: Space reduction of directories with force-compress

4.7 Btrfs Defragmentation Tool Results

As it can be seen in Figure 4.71, the time taken by the Btrfs defrag command to defragment a file is very minimal as compared with the total number of fragmentations in the filesystem. The time taken by the defrag command shows a very large time reduction as the number of execution runs of the tool increases.

It can be easily seen that, after the first run, the time required to correct the fragmentation as well as the reduction in the percentage of fragmented file is very dramatic. Moreover, the raw data shows that the time taken to defragment and also the achieved reduction in fragmentation percentage are very consistent across runs.

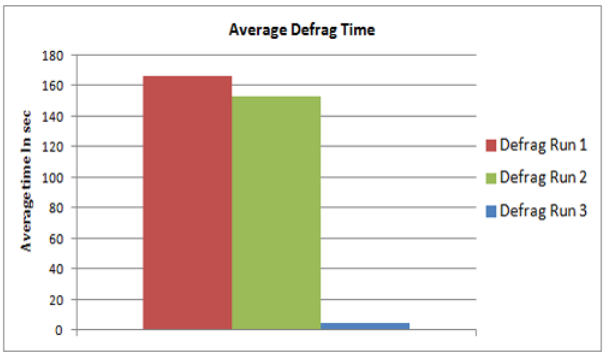

Figure 4.71: Average time for defragmentation

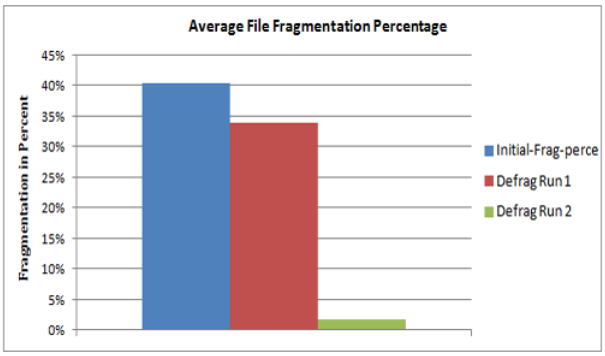

Figure 4.72: Percentage of file fragmentation

Chapter 5

Analysis and Discussion

This chapter presents the results of the various tests described in the previous chapter, and well as discussions of their implications.

5.1 I/O Performance Results

Tables 5.1 and 5.2 summarize the performance differences between the Ext4 and Btrfs filesystems for the single disk and volume tests (respectively). The noted differences in Iozone performance run across all record sizes unless explicitly stated otherwise. Mean performance increases are given with the standard deviation in parentheses.

Significant Performance Differences for Single Disk Tests

Test	Iozone Perf. Advantage	Throughput	Seeks
Sequential Read	no difference	similar	Btrfs many more
Random Read	Ext4: 7.5% (4.3%) for files≥4GB with record size< 512KB	similar	Btrfs many more
	Btrfs: 17% (10.7%) for files ≥4GB with record size>512KB		
Strided Read	Ext4 with record size ≤512KB: 45.9% (6.3%) for files=4GB; 53-71% for files=8GB Btrfs with record size>512KB: 12.5% (11.3%) for files≥4GB	similar	similar
Sequential Write	Btrfs: 55.3% (9.5%) for files <512MB; 20.6% (3.8%) for 512MB files	similar	similar
Sequential Rewrite	Btrfs: 11.6% (3.3%) for files≤256MB Ext4: 11.6% (1.8%) for files=1GB		
Random Write	Btrfs: 11.7% (3.1%) for files ≤512MB; 30.3% (33.7%) for files≥2GB Ext4: 9.7% (3.8%) for files=1GB	similar	Btrfs many more

Table 5.1: Single disk performance differences

Significant Performance Differences for Volume Tests

Test	Iozone Perf. Advantage	Throughput	Seeks
Sequential Read	Btrfs: 62.3% (6.5%) for files ≥4GB	Btrfs ∼33% lower	similar
Random Read	Btrfs: 30.9% (18.1%) for files=4GB and 8GB with record size≥1MB		Btrfs many more
Strided Read	Ext4 with record size up to 1MB: 19.3% (0.1%) for files=4GB; 51.1% (0.1%) or files=8GB Btrfs with record size≥2MB: 53.4% (11.3%) for files≥4GB	Btrfs ∼25% lower	Btrfs slightly more seeks during the first half
Sequential Write e	Btrfs: 54.0% (14.6%) for files≤256MB; 21.7% (4.4%) for files=512MB; 85.2% (7.6%) for files≥1GB	Btrfs ∼33% lower	Btrfs many more seeks (∼3x)
Sequential Re-write	Btrfs: 12.4% (3.7%) for files≤256MB; 83.0% (21.1%) for files≥1GB Ext4: 7.8% (4.4%) for files=512MB		
Random Write	Btrfs: 159.7% (66.7%) for files ≥4GB	Btrfs ∼33% lower	Btrfs many more

Table 5.2: Volume performance differences

5.2 Read vs Write operations

As can be recalled from the results section, Ext4 provides better throughput on random read operations with large files in general, where as Btrfs achieved higher throughput on random write operations for most of file and record size combinations performed done on a single disk. As similarly seen with random write operations, Btrfs also performs better for random read operations with volumes. However, Ext4 perfoms better with 1G and 2GB file sizes.

Similarly, for random write operations Btrfs performs better with smaller and larger file sizes while Ext4 performs better in the middle of the file size range.

In the copy-based sequential I/O read tests, both filesystems perform similarly. In the sequential write test, Btrfs performs better for smaller files while Ext4 performs better for larger files. For similar operations done on volumes, Btrfs performs better for writes operation in general as well as for large files in read operations.

Figure 5.1 shows the overall results of file read/write operations done on a single disk as well as volumes. It can be seen that Btrfs performs better for read operations while Ext4 performs better for file write operations in general. However, as it has been exhibited with Iozone random write test result, Btrfs performs differently when working with a single disk and with volumes. It shows higher performance as compared with Ext4 LVM volumes for the same write operation. On the other hand, read operations done on volumes are very much similar for both filesystems.

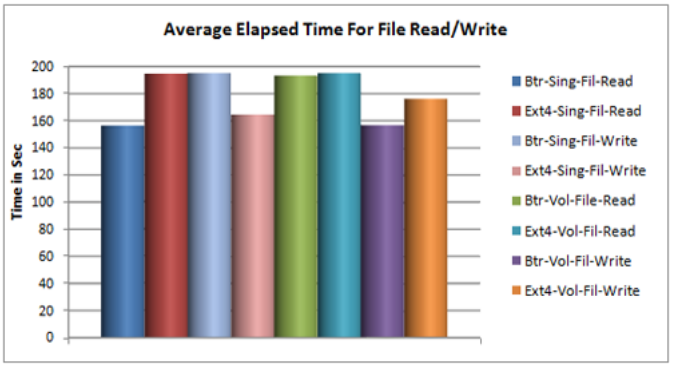

Figure 5.1: File Read/Write Elapsed Time

In addition to the differences exhibited with synthetic workloads as well as the file/directory read/write tests, in the real world application test of the computational chemistry simulation with large sequential I/O operations, Ext4 performs 3.7% faster than that Btrfs while using almost identical amounts of CPU (only a 0.6% usage difference). This is a small but significant difference.

The following subsections discuss these performance results from a variety of perspectives.

5.2.1 Single Disk vs Volume

Using volumes to perform any of the Iozone operations results in higher overhead as compared to the same operation with a single disk for most of the read operations. For example, a random read operation made to a single disk shows a 53% increment on the achieved throughput for Btrfs as compared with the volume. Similarly, the single disk random read operation done on Ext4 on average provides 57% higher performance as compared to using a Linux LVM volume.

The performance overhead of working with volumes increases with the increase of file sizes in general for the Ext4 filesystem. In contrast, Btrfs write tests with Iozone mostly show that, rather than a performance penalty, there is performance gain, especially for large files, while working with volumes.

The directory and file read/write tests indicate that reading a file from a single disk is faster than reading a file from a volume for Btrfs. In contrast, for Ext4 there is no significant difference reading from a single disk or from Linux LVM volumes. In contrast to file reading, directory tree read test results displayed that reading from a volume is faster for Btrfs, whereas reading a file from an Ext4 volume takes almost twice the time needed for doing the same operation on single disk.

Directory tree read tests do not show any significant difference for working with Ext4 single disks or LVM volumes. In contrast, for Btrfs performing directory reads is faster when working with volumes as compared with the single disk. Directory write results do not show any difference when done with both single disk and volumes for Btrfs filesystem. However directory tree writing perfoms better when done with the single disk for Ext4.

Even though there is a difference on performance when working with volumes, especially for the Iozone tests, the block layer I/O tracing results do not show that much difference in disk seeks between single disk and volume. Moreover, the Seekwatcher graphs clearly show that there is not a large difference on the achieved throughput, especially for Ext4. Btrfs shows minimally lower value for the achieved throughput for all tests except for strided read. The difference in performance shows that there is a large performance penalty of about $\sim$30% when with working with Btrfs Volumes.

In contrast to the results of the Iozone test block tracing , both the directory tree and file read/write operations exhibit a high performance degradation while working with volumes. For the file read operation, Btrfs achieved a throughput of up to 80 MB/s; on the other hand, it can only reach up to 20 MB/s for the same task with volumes. Moreover, Btfs shows a higher throughput of up to 70 MB/s for the single disk whereas it achieves 45 MB/s for volumes in the file write operations.

The performance difference exhibited for file read/write operations for Ext4 is small compared with that of Btrfs. Ext4 shows that file read operations results in a $\sim$10% difference while write operation results in $\sim$20% difference of achieved throughput. In contrast, sequential read operation tests don't show any difference on either filesystem when doing the same operations on single disk and volumes for file sizes up to 4G. But for files with size of 8GB, a performance improvement of 9% is exhibited on average for Ext4 filesystems. Strangely, Btrfs shows that for 8GB files, sequential operation done on volumes are 45% better than doing the same operation on a single test.

Strided read tests of Iozone tests made on a single disk do not show any performance gain as compared to volumes in general. However, a similar to the strange result that was exhibited with the sequential read operation for the file size of 8GB also happen for strided read tests. Btrfs shows that for files that are greater than 4GB, volumes have better performance, 27% on the average.

For random write operation done on the single disk, there is no difference exhibited for either filesystem other than Ext4 showing a 5% performance gain on the average for file sizes from 512MB to 1GB.

5.3 Large vs Small File Size

In most cases, Btrfs performs better with large file write operations whereas Ext4 is better for large file read operations. In the random read operation, Ext4 shows higher throughput with 8GB file in combination eith all record sizes for tests made on a single disk. However, the performance with 4GB is a mix, and Ext4 performs better with record size $\leq$256KB while Btrfs performs better with record size $\geq$512KB. The higher performance of Ext4 for large file size is different when reading is made from volumes. Btrfs performs better with file sizes of 4GB and 8GB in combination of record size $\geq$1MB.

In contrast, Btrfs performs better for file with sizes of 4GB and 8GB for random write operations on a single disk. Ext4 performs better with files of 1GB and 2GB in a similar setting. Similarly, for random write done on volumes, Btrfs performs better for both small and large file sizes, except for the file size of 512MB, where Ext4 exhibits better performance.

For sequential read operations, Ext4 outperforms Btrfs for both small and larger file sizes for tests done on a single disk, while Btrfs performs better with larger file sizes for the similar tests done on a volume. Moreover, the results obtained from sequential write operations done on a singe disk shows that Ext4 performs better for large file sizes while Btrfs performs better for small files. However, for the same test type done on volumes, Btrfs shows higher performance with large file sizes.

The results obtained from the sequential re-write test is exactly the same as for the sequential write, for both single disk and volume. However, with the strided read test Btrfs performs better for large file sizes for record size $\leq$512 whereas Ext4 performs better for similar file sizes but with record size $\geq$1MB.

5.4 Strange and Unexpected Results

Tests done with the Iozone benchmarking tool shows some kinds of variation and some results that are difficult to explain. Some of the observed strange scenarios are as follows:

- At times, Btrfs shows very large error bars, for both single disk and volume, for both sequential and random write operations, with and without compression. These occur primarily with file sizes in the range 512MB-1GB.
- Ext4 shows higher performance than Btrfs for the single file size of 512MB in random write,random read and sequential re-write operation done on volumes.
- Ext4 shows higher performance than Btrfs for the single file size of 1GB in random write test done on the single disk.

- Btrfs shows a very large performance difference with write operations done in a volume for large sized files, as compared to the difference shown between Btrfs and Ext4 for a single disk with similar operation.

5.5 Compression feature efficiency

Table 5.3 shows total time taken for compressing files and directories of various different sizes. bzip2 used with Ext4 requires a very long compression time for both the file and the directory While Zlib and LZO took less than 5 minutes and 2 minutes for file and directory compression (respectively), bzip2 took 169 minutes and 3 minutes for file and directory compression (respectively). The difference on the time taken to make compression becomes narrower when compressing the directory. Turning on the forced compression feature does not show a large difference on the time taken to perform the compression as compared with using only the compress mount option, for both Zlib and LZO.

Compression Time in Minutes

	File/Directory size				
Compression Tool	**200**	**6144**	**15360**	**489**	**1019**
Zlib	0.05	1.80	2.43	0.58	1.27
Zlib compress-force	0.05	1.58	4.08	0.59	1.70
LZO	0.04	1.26	3.10	0.66	1.32
LZO compress-force	0.04	1.59	3.82	0.67	1.35
bzip2	0.73	84.77	169.73	1.36	3.68

Table 5.3: Compression Time

As shown in Table 5.4, bzip2 shows a very large space reduction percentage for both file and directory compression. However turning compress-force mount option of of Btrfs allows dramatic space reduction with a very minimal increment on the time required to perform compression. Even though LZO performs better than Zlib for file space reduction, Zlib performs better when compressing directories.

Space Saving Percentage

	File/Directory size				
Compression Tool	**200**	**6144**	**15360**	**489**	**1019**
Zlib	11%	10%	5%	71%	45%
Zlib compress-force	87%	87%	88%	71%	46%
LZO	77%	77%	78%	58%	37%
LZO compress-force	87%	87%	88%	71%	46%
bzip2	95%	97%	97%	84%	53%

Table 5.4: Space Saving Percentage

5.5.1 LZO Compression Test Results of Iozone

For the main Btrfs compression tests described in Chapter 3 and reported in Chapter 4, only the Zlib compression was available for automatic compression and the LZO tests were performed manually. Since then, a new kernel version (2.6.38) was release which includes support for both compression methods in Btrfs, and additional tests have been made with Iozone with LZO compression enabled. This Test results are from the same OS but with the updated kernel.

As can be seen from the graphs below, the LZO compression features provides higher throughput for all types of write operations. However the performance achieved for read operations is not significantly better than for the Zlib compression Option.

LZO compression shows a minimal improvement in throughput for sequential read operations for files with size of 1GB and 2GB, with all given record sizes. However, for file sizes of 4G and 8GB there is no significant improvement on the achieved throughput gain with LZO as compared with Zlib compression feature. Representative results form the full I/O test file size-record size combinaiton range are plotted in Figure 5.2.

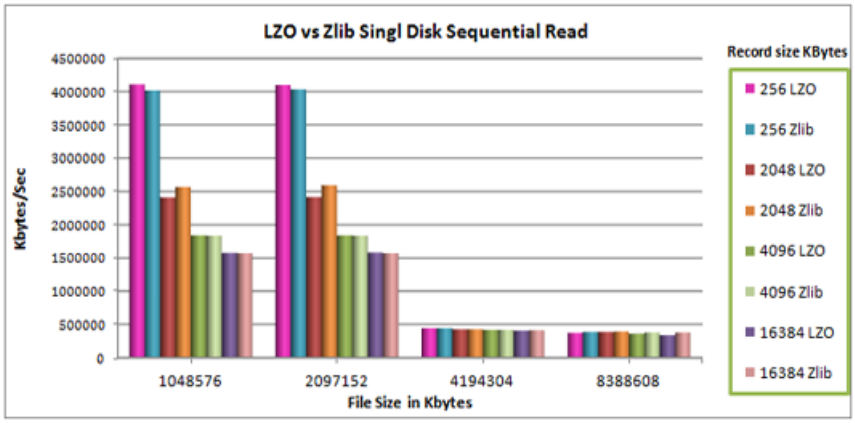

Figure 5.2: LZO and Zlib sequential read I/O throughput

Figure 5.3 shows representative results from the Iozone random read test (single disk). There is no significant difference in achieved throughput between using LZO and Zlib compression features for random read operations although the difference seen on the attained throughput between LZO and Zlib is a bit higher than for the sequential read operation.

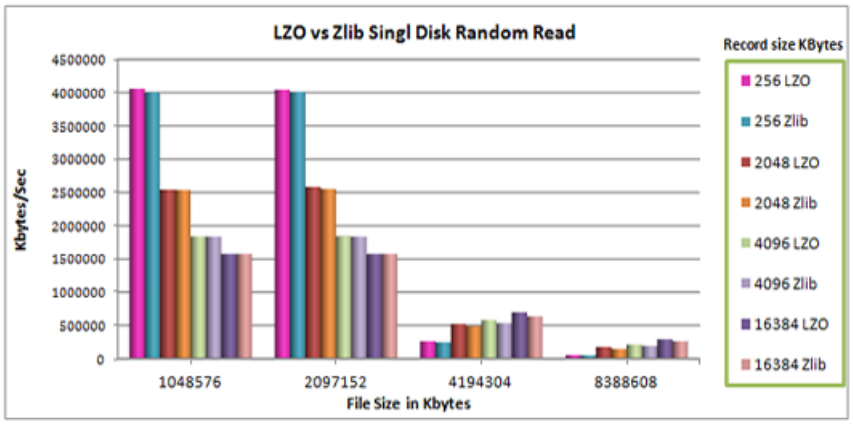

Figure 5.3: LZO and Zlib random read I/O throughput

As similarly seen with sequential and random read operations, the performance improvement gained through using LZO compression is quite small compared with that of the Zlib compression feature for the strided read test, as shown in Figure 5.4.

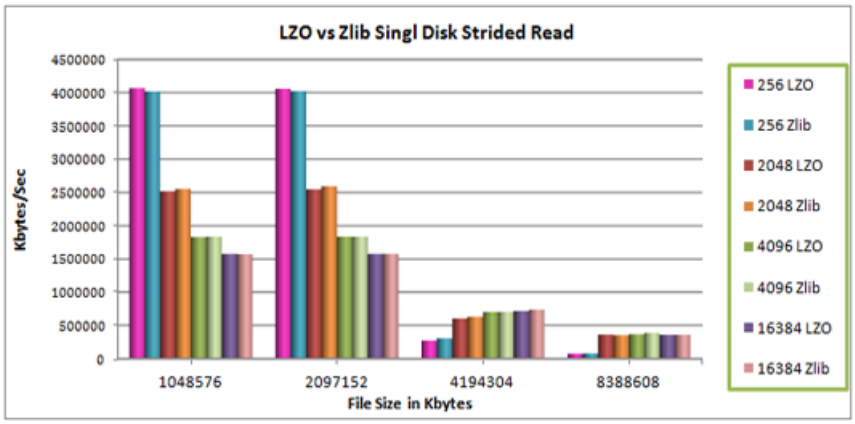

Figure 5.4: Strided read I/O throughput with LZO compression

Turning on the LZO compression feature for sequential write operations results in higher throughput of more than twice that exhibited while using Zlib compression for all record sizes and file sizes of 1GB to 8GB. Figure 5.5 shows representative results.

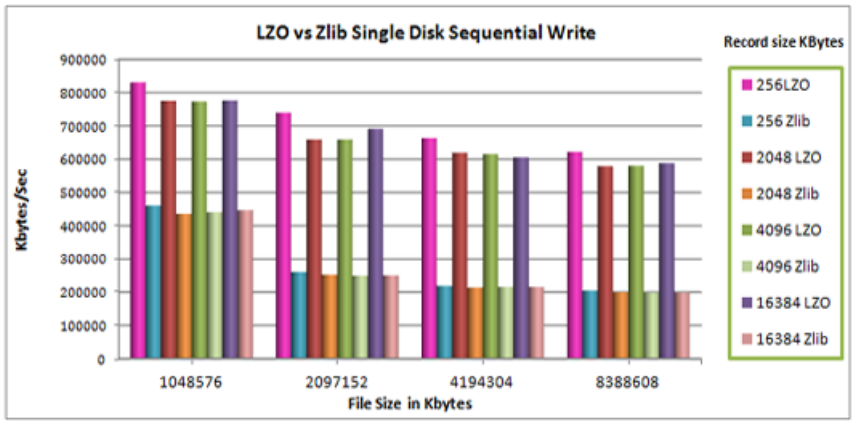

Figure 5.5: LZO and Zlib sequential Write I/O throughput

As can be seen in Figure 5.6, LZO shows higher throughput, more than double that for Zlib, for random write operations with file sizes of 1GB to 2GB. Moreover, the performance gain obtained from LZO compression becomes three times higher for files of size 4GB and 8GB.

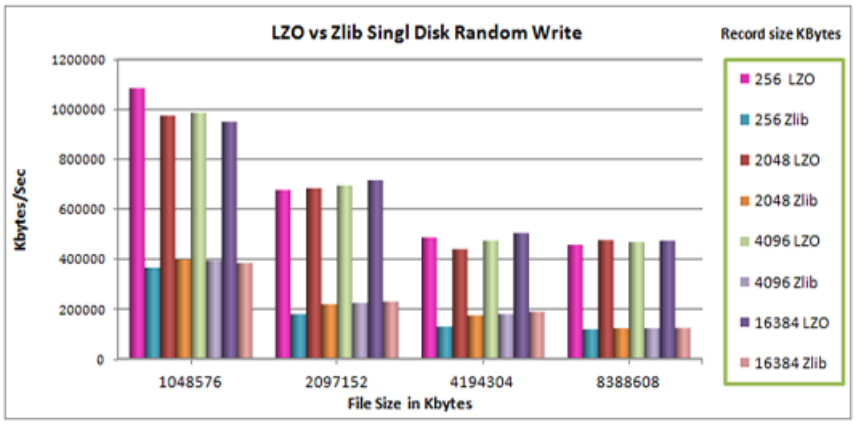

Figure 5.6: Random Write I/O throughput

The performance improvement gained from the LZO compression feature for the sequential re-write test is quite similar to the performance shown with sequential write operation. As it is shown below in figure 5.7 the achieved throughput is doubled with record sizes up to 512KB . More over with record sizes grater than 1MB with file sizes in the range from 1GB to 8GB the achieved throughput goes as higher as three times when compared with that of Zlib.

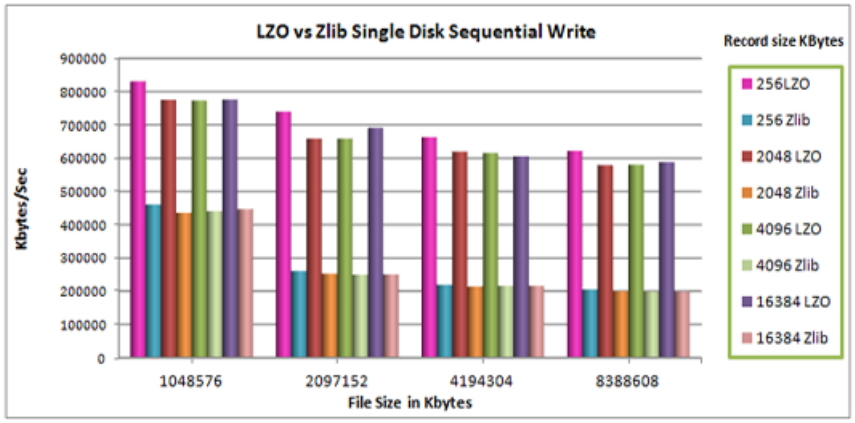

Figure 5.7: LZO and Zlib sequential re-write I/O throughput

5.6 Efficiency of the Btrfs Defragmentation Tool

Measuring the efficiency of the Btrfs defrag utility is performed by using a qualitative measurement, since there is no similar tool with a similar objective for the Ext4 filesystem that is ready for production use. The time required by the defrag tool to achieve defragmentation relative to the level of file fragmentation in the filesystem, and the number of fragmented files in the filesystem, are used as a metrics to evaluate its efficiency.

Table 5.5 reports the time taken to defragment, the fragmentation percentage for the filesystem after the defrag run, and the percentage reduction in fragments achieved during that run (with respect to fragmentation level before the run). Fragmentation decreases dramatically with each defrag run. In its first run, the defragmentation tool corrects an average of 170 fragments out of a total number of 415 initially present, with a standard deviation of 16.6.

	defrag run time (sec)	**FS fragmentation %**	**% Reduction**
defrag run 1	166	34%	16%
defrag run 2	153	2%	96%
defrag run 3	5	0%	100%

Table 5.5: Defragmentation tool efficiency

Chapter 6

Conclusion and Future Work

6.1 Summary of main findings

The overall performance analysis made between Ext4 and Btrfs shows differences in the performance results obtained with the synthetic benchmarking tool and real world application tests. The results obtained from the Iozone benchmarking tool show that the Btrfs filesystem provides better performance for read operations with large sized files in general, with sequential read operations being an exception. Similarly, Btrfs provides higher performance with large file sizes for read operation performed with volumes.

In contrast to read operations, Btrfs provides higher performance for most write operations to small sized files. For random write operations, the results show higher performance with large sized files, as seen with read operations to a single disk.

The results of Seekwatcher show that Ext4 provides similar performance when while working with both read and write operations to a single file. In contrast to the result obtained from Iozone, Seekwatcher shows that Ext4 performs better while working with volumes. In addition to the difference in the achieved throughput between the filesystems, Seekwatcher also shows that Btrfs performs many more seeks, which is one of the factors that can have a large impact on filesystem performance.

Directory tree and large file read/write tests results indicate that Btrfs provides better performance for these operations, whereas Ext4 provides higher performance for both directory and file write operations on a single disk. On the other hand, there is no performance difference exhibited between the filesystems for file reads operation done on a volume. Btrfs provides higher performance for writing a very large file, while Ext4 displays higher perfrmance for writing a directory tree.

The computational chemistry test results indicates that Ext4 provides higher throughout while making fewer I/O operations for this I/O intensive application.

Enabling the default Zlib compression option for Btrfs doesn't result in a large difference when working with small files. However, it provides a higher performance improvement for large files. On the other hand, using the LZO compression feature results in a performance improvement for both small and large file sizes with write operations, but no noticeable difference with read operations.

File and directory compression results shows that bzip2 compression is capable of providing the highest space saving but with the longest compression time. Btrfs transparent compression with compress-force option provides a good amount of space saving with tolerable compression time.

The Btrfs defragmentation tool results shows that the tool is efficient, both in terms of its ability to reduce file fragmentation to a high degree, as well as with its speed in performing the defragmentaion process.

6.2 Evaluation and Future Work

The research in this thesis performed as many possible tests of filesystem functionality as feasible given the timeframe of the work. The following observations can be made about ways that this research could be strengthened even more.

- The Iozone benchmaking tool is good in providing an overall assessment of filesystem performance. However, it sometimes shows discrepancies in the results obtained over multiple runs, most significantly in its write operation tests. This causes its reliability in providing true write performance differences to be somewhat uncertain.
- Implementing tests with both macro and micro benchmarking tools would be helpful in understanding the differences between the filesystems.
- Files used for compression tests were only of one type. The result will be more comprehensive if different file types are tested, since this would help to identify the interactions of file types with compression algorithms
- The Btrfs LZO compression option is tested only in the single disk environment. It should also be tested with volumes.
- Block tracing was not done with the LZO compression tests. It should be performed in order to find out the performance impact of this effective compression feature.

- Experiments should be performed with much larger volumes than were examine here in order to determine to extent to which Btrfs has accomplished its major development goal as a scalable filesystem.
- Btrfs has many additional features that are worth investigating, including such as RAID support and snapshots.
- The two filesystems' performance could also be investigated in a network environment to see if one has an advantage as a file server.
- Virtualization has been implemented widely in most computing environments and investigating how Btrfs performs while used in a virtualized environment would also be interesting and useful.

In conclusion, this thesis has achieved its goal of performing a thorough comparison of the Btrfs and Ext4 filesystems in terms of performance and with respect to some of Btrfs' unique features (compression and defragmentation).

Bibliography

[1] *Lars wirzenius, Jonna Oja , Stephen Stafford. Linux system administration guide, 2004*

[2] *Valerie Aurora . A short history of Btrfs, July 22, 2009.*

[3] *Nathan Willis . Weekend Project: Get started with Btrfs, October 15,2010.*

[4] *Suresh M .Ext4 File system- Feature and setups, October 26, 2009.*

[5] *Michael Hasenstein. LVM HOWTO, 2006* [Online] Available: `http://www.tldp.org/HOWTO/LVM-HOWTO.html`.

[6] *Michael Hasenstein. Logical Volume Management,2001* [Online] Available:`http://www.hasenstein.com/lvm_whitepaper.pdf`

[7] *M. Tim Jones. Anatomy of ext4 February 17,2009* [Online] Available: `http://www.ibm.com/developerworks/linux/library/l-anatomy-ext4/index.html`.

[8] *Amanda Amcpherson. A conversion with Chris Mason on BTRFs: the next generation filesystem for linux ,June 22,2009* [Online] Available: `http://www.linuxfoundation.org/news-media/blogs/browse/2009/06/conversation-chris-mason-btrfs-next-generation-file-system-linux`

[9] *M. Avantika et al., The new ext4 filesystem: current status and future plans, in the Proceedings of the Linux Symposium, Ottawa, Ontario Canada, June 2007.*

[10] *Abraham Silberschatz, Peter Baer Galvin, Greg Gange. Operating Systems Concepts,Seventh Edition ,2004.*

[11] *Timothy Platt. File System Compression in HFS+: Space savings and performance gain?, November 20,2009* [Online] Available: `http://developercoach.com/2009/file-system-compression-in-hfs-space-savings-and-performance-gain/`

[12] *Josef Bacik. Btrfs Swiss Army Knife of Storage ,February, 2012* [Online Available: `https://c59951.ssl.cf2.rackcdn.com/4376-bacik_0.pdf`

[13] *Jeffy B. Layton. Linux Don't need No stinkin' ZFS: Btrfs intro & Benchmarking, April 21, 2009* [Online] Available: `http://www.linux-mag.com/id/7308/2/`

[14] *M. Tim Jones. Anatomy of the Linux file system October 30,2007* [Online] Available: http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/.

[15] *Robert Love. Linux Kernel Development, Third Edition ,2010.*

[16] *AEleen Frisch. Essentials of System Administration, Third Edition ,2002.*

[17] *Theoder Tso. Ext4: The Next Generation of Ext2/3 Filesystem,2007*

[18] *Markus Gattol.Logical Volume Manager* [Online Avialbele: http://www.markusgattol.name/ws/lvm.html

[19] *Paul Krzyzanowski. File system design case studies, October 30,2010* [Online] Available: http://www.cs.rutgers.edu/~pxk/416/notes/12-fs-studies.html

[20] *Chris Mason. Btrfs Design* [Online] Available: http://oss.oracle.com/~mason/btrfs/btrfs-design.html

[21] *Alan D.Brunelle. Blktrace User Guide, February 18,2007* [Online] Available: http://www.cse.unsw.edu.au/~aaronc/iosched/doc/blktrace.html

[22] *Chris Mason.Seekwatcher, February 18,2007* [Online] Available: http://oss.oracle.com/~mason/seekwatcher/

[23] *William D. Norcott .Iozone Filesystem Benchmark, October 28,2006* [Online] Available: http://www.iozone.org/docs/IOzone_msword_98.pdf

[24] *M. J. Frisch et al. Gaussian 09 ,Gaussian Inc. Wallingford CT 2009*[Online] Available: http://www.gaussian.com/g_prod/g09b.htm

[25] *LZO real-time data compression library*[Online] Available: http://www.oberhumer.com/lzo

[26] *Zlib:* [Online] Available: http://www.zlib.net

[27] *bzip2:* [Online] Available: http://www.bzip2.org

[28] *Wasim Ahmad Bhat et al. Benchmarking Criteria for File Systems Benchmarks, International Journal of Engineering Science and Technology (IJEST) , Vol. 3 No. 1, Jan,2011* [Online] Available: http://www.ijest.info/docs/IJEST11-03-01-164.pdf

[29] *Nitin Agrawal, Andrea C. Arpaci-Dusseau,Remzi H. Arpaci-Dusseau. Towards Realistic File-System Benchmarks with CodeMRI Jan,2008* [Online] Available: http://pages.cs.wisc.edu/~nitina/codemri-hotmetrics08.pdf

[30] *Traeger Avishay et al. A Nine Year Study of File System and Storage Benchmarking, ACM Transactions on Storage, Volume 4 No. 2 , May ,2008* [Online] Available: http://www.fsl.cs.sunysb.edu/docs/fsbench/fsbench-tr.pdf

[31] *Jan Kra. Ext4, btrfs, and the others. In Proceeding of Linux-Kongress and OpenSolaris Developer Conference, 2009* [Online] Available: `http://atrey.karlin.mff.cuni.cz/~jack/papers/lk2009-ext4-btrfs.pdf`

[32] *Dominique A. Heger. Workload Dependent Performance Evaluation of the Btrfs and ZFS Filesystems, DHTechnologies, Austin, Texas* [Online] Available: `http://www.dhtusa.com/media/IOPerfCMG09.pdf`

[33] *Data Compression Benchmark and ROI Analysis, White paper, PKWARE,* [Online] Available: `http://www.pkware.com/documents/resources/whitepapers/ROI.pdf`

Appendix A

Average Results of Iozone Benchmarking Tool

A.1 Random Read Single Disk and Volume

Results of Btrfs single and Volume

Average Random Read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3660759	3793487	3794893	3755902	3257346	2428117	1901421	1747390	
16384	3734836	3806428	3847479	3793976	3267264	2411558	1853522	1662051	1649689
32768	3769193	3833850	3879287	3834869	3324706	2424234	1838233	1626482	1605959
65536	3791056	3881352	3922595	3881251	3377229	2436054	1832582	1611965	1586316
131072	3816742	3911477	3955636	3933249	3458760	2455032	1833137	1605656	1579211
262144	3830201	3937234	3983362	3975836	3563088	2467536	1836218	1603326	1576905
524288	3844977	3951101	4008969	4016017	3697750	2528214	1840084	1604703	1575873
1048576	3824438	3944525	4007423	4012959	3695392	2522725	1838427	1602343	1573861
2097152	3836502	3944366	4001479	4019905	3732893	2537317	1838906	1601571	1573946
4194304	29862.2	45610.3	67586.7	89618.4	133350.7	164270	202061.5	232705.5	231864.4
8388608	8631.3	13631.4	19655.9	23911.6	33530.1	46125.3	63575.6	66353.8	79425.3

Average Random Read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3691378	3775683	3779925	3785782	3264434	2441476	1908947	1746167	
16384	3736541	3813090	3844681	3793090	3283364	2429593	1856844	1666404	1655856
32768	3780257	3854341	3891392	3855485	3352356	2447334	1843079	1628860	1607228
65536	3809924	3893770	3932519	3903885	3406188	2461637	1836384	1613273	1590053
131072	3827903	3923818	3970386	3952524	3493436	2479433	1836814	1607349	1580209
262144	3841077	3946395	3995620	3991444	3620726	2510101	1839621	1604930	1577097
524288	3854489	3960349	4015465	4026120	3749108	2578074	1840815	1605389	1576972
1048576	3840598	3951495	4010453	4019857	3714356	2559139	1840931	1604254	1574872
2097152	3836682	3950398	4009764	4023422	3751968	2577049	1842804	1604404	1575163
4194304	39367.2	55799	78822.3	111112.5	151220.1	218362	253405	298120.9	305337.3
8388608	10014.5	15552	23220.3	31252.4	39419.3	62317.6	81399.7	92534.8	113699.6

Results of Ext4 single and Volume

Average Random write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1810813	1828001	1827853	1800848	1631913	1388001	1316467	1303202	
16384	1651603	1667208	1677562	1660991	1516702	1320053	1267611	1263192	1258184
32768	1581844	1599397	1613947	1601636	1470940	1284689	1243626	1240812	1239704
65536	1553122	1573940	1585831	1579713	1454093	1273977	1236189	1233310	1233510
131072	1538633	1561524	1571034	1571704	1456402	1272158	1233151	1230425	1231132
262144	1536392	1558264	1571517	1568730	1469064	1269338	1229685	1230230	1229468
524288	1493802	1510593	1520009	1513798	1400901	1224773	1189288	1188791	1189626
1048576	141406.8	158892.6	176039.9	185868.4	193077.9	195257.2	207600.2	211439.2	221125.4
2097152	53720.9	68446.8	76044.7	87307	97727.6	103013.5	106318.3	108732	111884.2
4194304	31424.9	45563.8	52942.5	59878	72083.6	76665.9	79751.7	81908.1	85295.6
8388608	23625.8	36374.8	45601.4	52023.8	61149.4	66227	70075	72704.4	75398.3

Average Random write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1807695	1821349	1826331	1793057	1620312	1385678	1313814	1302274	
16384	1646901	1666681	1673072	1653638	1508318	1316873	1266270	1259622	1258952
32768	1571931	1590862	1604115	1589461	1456605	1283791	1241073	1236551	1236950
65536	1552265	1570357	1583187	1570841	1445537	1274843	1235485	1233016	1233255
131072	1543947	1563558	1575918	1566736	1446792	1272116	1233983	1232394	1232736
262144	1538316	1560727	1572452	1568413	1456666	1271800	1233548	1231674	1232388
524288	1433678	1452321	1473406	1461492	1353201	1179053	1144774	1143869	1145861
1048576	132465	147926.2	167409.3	187712.4	193719.7	189326.8	190910.5	191118.5	192623.8
2097152	42672.1	58249.9	60462	76568.2	83900.6	90194	88536.5	89279.8	104441.9
4194304	26015	37979.8	45637.4	57887.7	67832.7	71990.3	71396.1	71110.7	73274.7
8388608	21507.4	32663.1	43073.8	51890	59189.5	61436.5	66617.8	68272.3	68710.6

A.2 Sequential Read Single Disk and Volume

Results of Btrfs single and Volume

Average Seq read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3767058	3783324	3813022	3785008	3247426	2449406	1931462	1771131	
16384	3820760	3862635	3878889	3813350	3286758	2438016	1879452	1674934	1663699
32768	3876696	3918416	3932497	3868636	3346714	2437581	1850942	1635354	1612278
65536	3912381	3955209	3955520	3912074	3407994	2447386	1840916	1615474	1591831
131072	3942912	3983392	3985284	3962589	3501115	2457617	1836801	1607555	1581522
262144	3966936	4008690	4029430	4009026	3663068	2526812	1837276	1605184	1578173
524288	3986482	4029288	4049824	4043287	3791844	2589840	1837420	1604828	1577343
1048576	3965496	4012589	4032016	4009828	3726915	2557906	1830393	1600296	1572613
2097152	3986318	4027498	4047504	4042705	3820952	2594569	1833720	1599553	1572215
4194304	93075.4	93297.1	93097.8	93368.1	93397.5	93157.5	93103.3	93036.7	93073.2
8388608	83927.6	83741.8	83420.1	84104.8	84149.2	84141.8	84141.4	84146.6	84139

Average seq read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3647831	3804095	3848417	3855407	3307458	2478755	1945941	1771385	
16384	3801547	3861172	3892706	3837411	3302081	2446978	1873519	1672575	1662179
32768	3844955	3901892	3923709	3866602	3346723	2446696	1851200	1634134	1612978
65536	3913688	3951856	3974926	3919414	3416823	2458916	1844092	1617170	1590797
131072	3951746	3992068	4012765	3977508	3521245	2474730	1840454	1609804	1582178
262144	3975306	4018928	4041318	4021778	3700753	2558834	1843100	1607146	1579426
524288	3993428	4035160	4057102	4050347	3817608	2618462	1842272	1607021	1577825
1048576	3962979	4000335	4021764	4008835	3710422	2567969	1834580	1601006	1572560
2097152	3976336	4015686	4035409	4032832	3789756	2598328	1836459	1600834	1572378
4194304	140952.7	140450.8	140889	139650.8	141916.8	141149.7	142120.2	142189.4	141681.3
8388608	121709.9	121652.8	122639.9	122268.1	121537.5	121333	122504.4	121491.8	122857.6

Results of Ext4 single and Volume

Average seq read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3837730	3887044	3878069	3797588	3235043	2463113	1939823	1764957	
16384	3849296	3899564	3905179	3821783	3274998	2454579	1874810	1676447	1659534
32768	3908701	3951995	3962115	3892393	3360903	2460936	1856114	1635708	1613225
65536	3940367	3982116	3993477	3940026	3423944	2472199	1842979	1617084	1591034
131072	3960295	4002469	4021546	3991117	3524141	2481969	1839686	1608296	1581575
262144	3978997	4018103	4041643	4030490	3704488	2565493	1841008	1605955	1578152
524288	3999983	4041570	4060568	4053235	3807777	2620592	1842073	1605647	1578092
1048576	3954405	3995082	4014623	4005239	3708467	2567082	1834224	1600025	1572543
2097152	3970801	4013022	4033639	4031593	3791301	2604738	1834583	1598855	1570444
4194304	93773.3	93205	92586.5	94024.9	95436.5	94111.6	93523.2	92853.1	91765.6
8388608	86086.3	84818.4	83992	86093.8	84698.5	84233.9	85949.2	84618.1	84499.7

Average seq read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3823233	3823314	3875711	3819052	3254447	2449458	1932707	1767776	
16384	3851824	3892541	3898083	3853830	3285996	2438249	1875442	1674857	1659210
32768	3901549	3936221	3952420	3896553	3357027	2447914	1850699	1635539	1591589
65536	3929618	3975077	3982313	3944283	3426638	2455357	1839607	1615640	1591781
131072	3960153	3998493	4022615	3997530	3534219	2473629	1834968	1608624	1580724
262144	3985501	4019866	4045840	4034781	3713591	2552724	1835819	1605401	1576490
524288	3993913	4037335	4053828	4054800	3823341	2608696	1837485	1605316	1576941
1048576	3935087	3981774	3994492	3991776	3700662	2557503	1828169	1599208	1570233
2097152	3962050	3999827	4022493	4022183	3777372	2577185	1828922	1598958	1570150
4194304	87153.9	84065.2	80520.7	84284.8	89091.7	87828.6	86299	84335.8	82526.3
8388608	79231.6	77509.6	73848.8	80036.7	77382.9	73516.6	80666	77032.8	74014

A.3 Strided Read Single Disk and Volume

Results of Btrfs single and Volume

Average strided read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3703563	3772233	3783089	6066374	5475331	3171423	1917674	1736386	
16384	3747450	3815629	3835893	3787422	5954871	3449360	1878183	1653958	1640179
32768	3752280	3832311	3867939	3831349	3289354	3696104	1883379	1630298	1606493
65536	3801091	3883483	3919569	3887298	3369069	2428423	1879798	1617965	1589005
131072	3834571	3921640	3961715	3943148	3454386	2445746	1832694	1609502	1579403
262144	3847471	3943176	3988712	3986749	3594445	2487909	1835883	1604686	1574981
524288	3863944	3962164	4014798	4021120	3739597	2555108	1838800	1605411	1578183
1048576	3860055	3964931	4019013	4021643	3748759	2559336	1836045	1604552	1576024
2097152	3864455	3960138	4017708	4033588	3784374	2566202	1837245	1601766	1573329
4194304	45371.4	51450.8	59734.5	62330.2	114579.6	158647.4	181843.5	208212.5	229827.8
8388608	9072.1	11238.3	14436.7	16859.3	29392	76865.4	78055.3	81561.5	85742.5

Average Strided read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3707896	3731609	3753794	6027162	5494816	3142403	1904338	1735212	
16384	3720774	3774109	3808425	3730416	5832484	3468528	1879050	1657086	1641524
32768	3744775	3817262	3845277	3768593	3275278	3657490	1868323	1626287	1607264
65536	3791053	3872349	3904613	3865496	3367193	2442814	1880876	1607316	1585673
131072	3820315	3891493	3948033	3890256	3436360	2455081	1828239	1604130	1574587
262144	3838396	3932328	3968325	3968687	3574476	2492477	1831214	1602234	1569228
524288	3850881	3951527	4005442	4006116	3715130	2555201	1834694	1601300	1574285
1048576	3854351	3951968	4006613	4010724	3722962	2556761	1834466	1600166	1572517
2097152	3859122	3962301	4012677	4024071	3783197	2583296	1834054	1599319	1571409
4194304	58905.6	58953.7	68568.4	82904.5	154624.8	208029.5	231065.1	277642	311983.7
8388608	12615.7	14299.6	16831.8	22187.7	37735.9	85903.9	92907.1	106928.4	116399.5

Results of Ext4 single and Volume

Average strided read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3718020	3798979	3824063	6098976	5443438	3134572	1905820	1733287	
16384	3766779	3829237	3856308	3789474	5854639	3403638	1877258	1663563	1647119
32768	3786827	3863192	3899041	3829487	3317254	3737800	1888200	1632077	1607426
65536	3820041	3897765	3940781	3898093	3378668	2425957	1874665	1615834	1587018
131072	3834671	3929476	3974098	3951181	3467877	2446290	1828177	1609128	1579365
262144	3856399	3954867	4000757	3994233	3620128	2488990	1830415	1605110	1574569
524288	3868474	3966760	4019058	4024122	3746776	2544928	1831393	1603742	1576954
1048576	3848062	3950273	4000421	4011445	3731849	2538810	1830831	1602950	1574758
2097152	3851537	3947690	4008791	4025538	3799200	2563052	1831208	1601765	1573797
4194304	60074.7	62314.7	87373.7	110684.4	168142.4	142958.8	167378.6	185675	199291.7
8388608	31516.8	30556.6	36699.3	43059.3	63464.5	55642.6	70473.3	79384.3	84385.9

Average Strided read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3701558	3809741	3799950	6029138	5374642	3151122	1911817	1727591	
16384	3755136	3822974	3791487	3797085	5827629	3507216	1872552	1658658	1642582
32768	3782942	3847385	3878273	3838940	3297402	3733070	1886669	1633646	1603666
65536	3806704	3889090	3926075	3884518	3374574	2443273	1881338	1614206	1587397
131072	3830382	3924377	3969181	3946000	3470943	2466093	1833909	1608512	1577942
262144	3851112	3943330	3996010	3991137	3627620	2514362	1836606	1603427	1573060
524288	3864384	3964651	4013108	4021688	3752707	2578333	1838183	1604354	1576495
1048576	3838666	3939995	3991838	4007097	3746969	2570053	1835786	1601119	1573721
2097152	3844947	3949298	4005900	4024699	3802194	2595496	1836255	1601661	1573323
4194304	57243.7	82771	92109.3	103861.4	159063.9	139069.8	157257	177757.2	181235.4
8388608	30054.8	34404.2	37161.5	39650.4	63275.8	52170	67618.2	74213.1	74658.6

A.4 Sequential Write Single Disk and Volume

Results of Btrfs single and Volume

AverageSequential Write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1667690	1736302	1781362	1770513	1606021	1429038	1358901	1201662	
16384	1540254	1608145	1648966	1641114	1492000	1339213	1294562	1284744	1200785
32768	1478702	1542856	1579832	1580054	1440385	1298710	1263688	1261664	1262118
65536	1449992	1513042	1547498	1551407	1420598	1276970	1246898	1247587	1249120
131072	1438736	1502036	1538184	1543650	1434217	1269619	1238814	1240683	1243578
262144	1431766	1498528	1534793	1545990	1463255	1273007	1236595	1239058	1241828
524288	1091194	1145385	1221343	1185315	1083445	983808.9	951258.1	964778.2	975382.4
1048576	211310.1	212328.1	211801.3	213464.5	213031.1	208298.9	208658.2	210564.3	208682.8
2097152	111032.2	111410.4	111669.9	111252.2	111351	111139.8	110939.9	110642.2	111161.5
4194304	90485.1	91890.4	91957.7	92009.3	91901.7	92070.6	91746.3	91951.4	92218.1
8388608	83465.3	85658	85676.3	85500.3	85706.2	85495	85582.2	85510.3	85624.6

Average Seq Write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1550051	1763063	1787348	1763909	1605624	1385684	1278235	1202731	
16384	1544489	1608193	1646715	1636071	1506402	1342644	1296854	1291758	1201300
32768	1485124	1547839	1583060	1582372	1461278	1299915	1261622	1262404	1262734
65536	1454976	1514734	1551628	1555957	1449477	1280761	1244423	1247854	1250910
131072	1438943	1501218	1540326	1548637	1464842	1277539	1236170	1240295	1242287
262144	1427647	1491363	1532233	1543537	1474964	1278456	1232976	1235155	1239743
524288	1044762	1148943	1162522	1205211	1139097	1045644	914275.6	976014.5	969580.5
1048576	354595.8	358427	356828.8	358021.3	355940.5	350392.4	349657.5	349370.5	348541.5
2097152	188023.4	188630.1	188191.9	188439.3	188661.5	187502	187164.8	186866	186843.4
4194304	151678	151833.3	151993.9	151627.5	151656.6	151323.2	151469.1	151480.2	151511.5
8388608	136875.4	140154	140258.7	140413.7	140267.3	140268.6	140029	140147.5	140156.9

Results of Ext4 single and Volume

Average Sequnetial write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1037150	1040171	1039139	1031636	974557.7	911289.6	877965.5	806428.3	
16384	988174.4	996634.1	999262.6	990997.9	931743.6	869799	849578.6	845870.3	804687.3
32768	966179.6	971512.3	975808.3	967192.4	911272.7	852849.3	836052	833171.9	832012.1
65536	955113.8	961210.5	964937.7	959649.1	902912.7	843216.8	828827.3	826765.5	826762.1
131072	949399.5	957155.7	963356.3	957697.8	905832.1	839176.3	826212.6	824932.9	823975.5
262144	948672.1	957169.7	963377.1	959230.3	913462.1	839701	823576.6	822269.9	822002
524288	939951.6	947407.7	952205.2	947577.4	900344.7	827255.6	812772.9	811933.8	811221
1048576	214174	216707.6	215069	223506.8	221271.7	223552.8	220721.3	221377.2	222069.2
2097152	112111.4	111441.6	111156.5	116867.5	117042.3	114252	117532.7	117386.6	121598.1
4194304	97698.1	91510.5	91196.8	89250.7	94015.5	96540.9	97360.5	90409.1	92704.6
8388608	83757.7	87911.7	83849.1	84635	86720.1	82634.8	86711.4	85432.2	82889.1

Average Seq write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1044260	1050145	1048131	1037532	980701.7	914885.7	884256.6	812533.1	
16384	996755.8	1004462	1006658	1000221	942921.4	879162.4	856852.4	851645.8	809605.3
32768	973354.7	978128.7	983325.4	977712.9	920624.7	859148.9	842234.6	838928	837918.6
65536	963080.8	968659	973030.1	966389.1	911680.1	849432.2	833608.5	832810.3	833828.9
131072	957668.4	965124	967277.2	964034.4	913618.7	844856	831674.4	829765.5	829281.4
262144	956035.4	963969.5	969149.3	964809.7	921314.3	844207.8	829856.8	828644.5	827966.6
524288	933624.4	942071	946081.6	947122.1	896425.7	816391.5	803804.9	800743	801530.1
1048576	205354.3	203888.8	205766.7	205418.6	202750.1	197028.7	185899.4	188658.4	191971.9
2097152	101662.1	100807.3	99362.3	98334.8	98601.5	93630.9	93385.4	94203.4	97681.8
4194304	82090.8	81204.2	79329	83995.9	85618.3	84687.5	81472.3	79578.8	78760.1
8388608	77963.8	77522.3	72501	78886.4	75897.2	72556.6	79029.1	75568.2	73000.3

A.5 Sequential Re-write Single Disk and Volume

Results of Btrfs single and Volume

Average Sequential rewrite

	64	128	256	512	1024	2048	4096	8192	16384
8192	1832425	1946082	1994902	1984628	1779834	1562838	1482995	1466523	
16384	1683566	1775616	1822740	1823744	1647964	1457980	1404296	1394320	1400794
32768	1608207	1696668	1742963	1747702	1583304	1411466	1367888	1365771	1367019
65536	1576281	1660252	1706404	1713162	1561754	1388996	1351965	1350099	1354258
131072	1563427	1649513	1696209	1707701	1579695	1381768	1342883	1345162	1349025
262144	1555088	1643185	1692669	1708791	1611250	1385933	1342700	1344491	1346344
524288	1356391	1416712	1466092	1422281	1318451	1193421	1115452	1136048	1140116
1048576	209604.6	209272.6	210974.1	211613.9	212006.2	209853.3	208192.4	207152.3	207295.4
2097152	110658.8	111665.2	111293.2	111208.9	111338.5	111064.3	110652.7	110798.8	110975.5
4194304	91595.7	90560.8	90168.6	90295.5	90514.9	90305.3	90203.7	90432.1	90019.7
8388608	85467.9	83436.5	83424.6	83376.4	83584.2	83505.7	83387.4	83495.7	83360

Average Seq Re-Write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1867269.60	1971011.20	2009098.00	2012224.30	1818001.80	1585838.90	1500812.40	1481824.90	
16384	1698928.40	1789532.00	1837637.40	1843846.30	1681601.30	1475225.00	1419846.30	1412164.80	1414452.40
32768	1626829.60	1713122.40	1756837.00	1770787.70	1628069.90	1425959.40	1378576.60	1377652.30	1376260.30
65536	1579699.90	1675426.40	1722342.30	1735071.90	1611041.10	1404778.30	1358787.90	1362337.20	1364976.40
131072	1574988.30	1659302.50	1709379.00	1725936.40	1629820.40	1401882.30	1349961.90	1354289.80	1356158.20
262144	1561698.90	1647177.60	1699554.40	1718190.40	1640956.80	1402746.40	1346732.90	1350186.10	1353776.20
524288	1213464.50	1350681.20	1336967.20	1332102.10	1362213.30	1112250.70	1034501.70	1032521.40	1091381.80
1048576	354798.80	357959.20	359377.60	359270.80	358418.00	351810.20	350825.70	349993.90	349817.90
2097152	185140.30	184947.30	184945.50	184915.70	185469.00	184664.90	185099.40	184401.00	184476.00
4194304	147347.00	147428.00	147484.40	147093.00	147249.10	146793.80	146681.80	146680.40	146713.90
8388608	133834.80	133021.90	133356.80	133220.00	132953.10	132980.80	132979.00	132955.30	132917.60

Results of Ext4 single and Volume

Average seq rewrite

	64	128	256	512	1024	2048	4096	8192	16384
8192	1730219	1738986	1732653	1710167	1545101	1372820	1300227	1280690	
16384	1595035	1603530	1609960	1589963	1443399	1290872	1241417	1230155	1229659
32768	1537106	1546861	1551165	1535147	1400525	1258157	1215168	1205516	1206365
65536	1512322	1523901	1528029	1517394	1387720	1240649	1203338	1198235	1196981
131072	1503621	1515291	1521276	1511593	1397718	1235196	1198029	1193882	1196662
262144	1496199	1510309	1517788	1513127	1419381	1235956	1195385	1193503	1194755
524288	1409047	1421863	1428244	1415863	1318909	1161678	1126117	1123149	1125781
1048576	231467.8	230192.6	232955.6	243223	239752.9	239788.1	239884.2	238713.9	238692
2097152	112173.3	110987.5	111732.9	117579.7	115882.7	113307.5	116647.9	116998.8	120723.2
4194304	96926.6	90703.1	90480.5	88625.2	93837.6	95810.4	96567.7	89427.6	91952.6
8388608	83515.3	87581.5	83526	84099.2	86230	82312.8	86259	85187.8	82503.6

Average seq rewrite

	64	128	256	512	1024	2048	4096	8192	16384
8192	1730219	1738986	1732653	1710167	1545101	1372820	1300227	1280690	
16384	1595035	1603530	1609960	1589963	1443399	1290872	1241417	1230155	1229659
32768	1537106	1546861	1551165	1535147	1400525	1258157	1215168	1205516	1206365
65536	1512322	1523901	1528029	1517394	1387720	1240649	1203338	1198235	1196981
131072	1503621	1515291	1521276	1511593	1397718	1235196	1198029	1193882	1196662
262144	1496199	1510309	1517788	1513127	1419381	1235956	1195385	1193503	1194755
524288	1409047	1421863	1428244	1415863	1318909	1161678	1126117	1123149	1125781
1048576	231467.8	230192.6	232955.6	243223	239752.9	239788.1	239884.2	238713.9	238692
2097152	112173.3	110987.5	111732.9	117579.7	115882.7	113307.5	116647.9	116998.8	120723.2
4194304	96926.6	90703.1	90480.5	88625.2	93837.6	95810.4	96567.7	89427.6	91952.6
8388608	83515.3	87581.5	83526	84099.2	86230	82312.8	86259	85187.8	82503.6

A.6 Random Read Single Disk and Volume with Compression

Average Random Read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3618938	3666669	3688586	3662643	3190069	2396566	1836239	1732290	
16384	3660584	3733313	3770834	3714437	3203120	2373488	1822993	1649367	1637334
32768	3702143	3781555	3819895	3781050	3260796	2398292	1822014	1621259	1599921
65536	3773195	3854170	3895683	3855415	3353543	2424810	1822594	1607171	1583549
131072	3809760	3898966	3942370	3915704	3431212	2447328	1825296	1603301	1576768
262144	3833684	3928352	3978772	3967850	3539100	2473641	1829041	1601577	1574750
524288	3854840	3953050	4005981	4010832	3664392	2523124	1831817	1601741	1574092
1048576	3839210	3944329	4000484	4007830	3668150	2521228	1831846	1601139	1573605
2097152	3835944	3949034	4008526	4024076	3735512	2547689	1832867	1600967	1573713
4194304	96622.5	168838.5	245464	324479.7	417841.7	504289.7	544961.2	626751.5	662714
8388608	17655.6	31892.2	51018.4	76148.5	104818.8	146611.9	189469.9	225666	263963.5

Average Random write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1887947	2029607	2099734	2086407	1848125	1567218	1477385	1472985	
16384	1711173	1835677	1904151	1899973	1705764	1483403	1425540	1422727	1424043
32768	1629537	1748016	1810285	1816943	1644287	1445013	1401119	1399796	1400206
65536	1590290	1705822	1771351	1780514	1619436	1429627	1391072	1391002	1393407
131072	1567783	1685698	1750400	1762916	1611880	1424299	1387430	1388297	1390644
262144	1556659	1676146	1741240	1760619	1616930	1421109	1386430	1387867	1391365
524288	1486290	1531062	1643193	1578598	1392488	1245925	1136296	1132489	1072600
1048576	306545	345762.2	374025.3	397450.5	409360.8	413243	411787.3	406518.7	404324.1
2097152	157365	166876	182386.5	197256.4	212240.9	229757.7	235942.3	241336.3	243298.7
4194304	128915	133189.7	133792.1	140245.7	154526.3	181650.2	186332.7	194252.5	199113.8
8388608	105017	119661.4	121167.4	122348	122527.5	122679.9	122819.6	122782	124182.6

A.7 Sequential Read Single Disk and Volume with Compression

Average Seq read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3684678	3745114	3755488	3718574	3236768	2443394	1941574	1771986	
16384	3789927	3830224	3856781	3812216	3274358	2446278	1882903	1679271	1664071
32768	3873068	3915067	3934080	3878315	3352904	2457513	1857632	1637400	1614804
65536	3920884	3961269	3983586	3936068	3430100	2470540	1847328	1619326	1592486
131072	3954871	3999003	4016254	3989740	3542606	2489616	1843622	1610933	1582855
262144	3977880	4023229	4044795	4030242	3725751	2583882	1843886	1608668	1579857
524288	3995567	4039094	4060096	4053648	3831086	2642065	1845632	1607845	1577791
1048576	3959304	3999975	4017803	4002902	3708000	2586043	1837200	1601218	1571912
2097152	3970119	4011393	4032604	4026342	3773127	2600865	1836931	1599911	1570766
4194304	442862.8	445328.2	444173.5	440825.5	439610.6	428196.9	418408.4	416169.5	415190
8388608	390533.8	392243.2	389533.4	390095.6	392522.7	390983	381874.8	380419.5	379527.9

Average sequential read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3705192	3805534	3868693	3808891	3295567	2482854	1933070	1767709	
16384	3790024	3845226	3863171	3812582	3288981	2443639	1874457	1674050	1660329
32768	3865389	3907185	3921422	3868292	3352596	2443634	1847851	1635516	1612310
65536	3921031	3957791	3973667	3915669	3421410	2458043	1839077	1616023	1590510
131072	3953512	3992862	4012316	3970663	3519702	2472084	1835595	1608665	1581049
262144	3975654	4018400	4038626	4015584	3683846	2538219	1836005	1606105	1577488
524288	3995976	4034812	4054195	4048599	3811162	2607803	1838826	1606149	1577330
1048576	3956423	3993751	4011078	3996414	3694135	2558899	1829393	1599887	1571047
2097152	3967199	4008921	4025903	4021474	3762024	2574732	1831124	1599146	1570805
4194304	141985.2	140699.6	142094.5	142544.1	141688.2	141700.9	141399.3	140619.1	142072
8388608	121480.8	121894.3	121873.1	122169.4	121582.8	120900.2	121678.2	121655.4	121492.4

A.8 Strided Read Single Disk and Volume with Compression

Average Strided read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3623557	3702790	3770411	6044521	5426864	3072809	1899479	1735816	
16384	3695750	3764483	3793328	3733656	5794066	3319805	1869912	1655424	1645919
32768	3753050	3821151	3854807	3810311	3288278	3640597	1875384	1632653	1608782
65536	3801496	3878890	3921019	3881921	3366209	2432048	1875999	1614451	1589413
131072	3826303	3918009	3961607	3941310	3454518	2443506	1831569	1608263	1581259
262144	3848158	3942480	3993479	3989593	3597971	2485298	1833419	1603852	1574017
524288	3861745	3964378	4015946	4022725	3738230	2546862	1837157	1604073	1576068
1048576	3863427	3966131	4018832	4030391	3763930	2554327	1836579	1604148	1576427
2097152	3863114	3963814	4022403	4040534	3831952	2588744	1836099	1602806	1575829
4194304	319350.7	292145.4	307498.4	316656.8	523159.5	633001.1	698622.3	702785.6	734426.7
8388608	89850.3	89305.5	81224.8	88145.2	162211.9	355383.3	390035.1	333231.2	358574.2

Average seqtrided read

	64	128	256	512	1024	2048	4096	8192	16384
8192	3676022	3756839	3763765	6047704	5393192	3109030	1913849	1733720	
16384	3735812	3797471	3822246	3757646	5952588	3455135	1871459	1660292	1645828
32768	3761598	3830976	3858716	3814765	3294294	3719965	1879995	1633403	1608899
65536	3808117	3884111	3919983	3877703	3364452	2441500	1876300	1617191	1589594
131072	3837459	3919468	3958051	3934640	3450718	2462876	1829631	1610097	1581424
262144	3853737	3942869	3988420	3982689	3590990	2501725	1832443	1605347	1576536
524288	3870370	3960386	4014079	4018366	3731630	2568101	1834374	1604764	1577415
1048576	3860366	3956166	4012044	4015770	3731440	2566234	1833781	1604147	1574976
2097152	3864011	3961392	4018828	4032484	3792515	2587550	1833656	1602935	1575071
4194304	59718.2	58972.1	69557.1	82751.8	154012.2	209206.8	233176.1	276280.6	312478.7
8388608	12567.7	14330.9	17039.5	22846.4	37084.4	87765.2	93657.3	108284	117789.8

A.9 Random Write Single Disk and Volume with Compression

Average Random write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1887947	2029607	2099734	2086407	1848125	1567218	1477385	1472985	
16384	1711173	1835677	1904151	1899973	1705764	1483403	1425540	1422727	1424043
32768	1629537	1748016	1810285	1816943	1644287	1445013	1401119	1399796	1400206
65536	1590290	1705822	1771351	1780514	1619436	1429627	1391072	1391002	1393407
131072	1567783	1685698	1750400	1762916	1611880	1424299	1387430	1388297	1390644
262144	1556659	1676146	1741240	1760619	1616930	1421109	1386430	1387867	1391365
524288	1486290	1531062	1643193	1578598	1392488	1245925	1136296	1132489	1072600
1048576	306545	345762.2	374025.3	397450.5	409360.8	413243	411787.3	406518.7	404324.1
2097152	157365	166876	182386.5	197256.4	212240.9	229757.7	235942.3	241336.3	243298.7
4194304	128915	133189.7	133792.1	140245.7	154526.3	181650.2	186332.7	194252.5	199113.8
8388608	105017	119661.4	121167.4	122348	122527.5	122679.9	122819.6	122782	124182.6

Average random Write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1901466	2047671	2111579	2105591	1858013	1578589	1490103	1480224	
16384	1716146	1842457	1909314	1917900	1716237	1489488	1431936	1431245	1436733
32768	1638601	1754426	1819197	1830178	1653134	1453186	1406896	1406729	1407322
65536	1599070	1710343	1776432	1789337	1628952	1434407	1395779	1397196	1397923
131072	1576002	1692762	1756744	1774767	1624532	1428804	1391818	1393675	1395986
262144	1562715	1682462	1749049	1770270	1636045	1426216	1388466	1390811	1394474
524288	1480544	1574615	1601371	1538290	1400015	1161549	1114724	1097055	1047510
1048576	329733.2	354116.9	392133.7	417730.9	431963.2	441140.7	440934.7	429008.5	418837.6
2097152	170714.5	173048.6	187905.6	205733.4	224345.2	245495.7	253830.7	260798.3	262036.1
4194304	140490.8	135292.1	135964.6	141708.7	159851.7	193263.4	198794.1	208000	214078.6
8388608	131312.7	124147.1	122457.5	122227.3	122657.7	122577.9	123182.5	123928.2	123477.5

A.10 Sequential Write Single Disk and Volume with Compression

Averagesequential rewrite

	64	128	256	512	1024	2048	4096	8192	16384
8192	1799993	1908662	1963322	1956525	1749207	1546011	1475900	1466107	
16384	1671626	1770906	1819461	1816669	1637432	1458378	1407340	1399012	1403835
32768	1610085	1699148	1746882	1747887	1584643	1415072	1374467	1369506	1370331
65536	1581368	1666771	1714840	1719636	1566787	1394527	1356647	1355782	1358174
131072	1564900	1653490	1700743	1712189	1581634	1389598	1350478	1350118	1353486
262144	1556089	1645983	1692375	1711547	1615593	1391284	1345880	1347664	1350853
524288	1201125	1227533	1328506	1302361	1216546	1059204	1024683	1043004	1047008
1048576	459050.2	463522.4	460244.3	467235.1	453816.3	445670.9	444580.9	444272.3	445998
2097152	253310.4	258048.2	255149	256173	254244.1	251145.5	249997.1	251849.8	250692.3
4194304	213476.6	214212.2	212921.8	213603.3	211682	209633.2	210009.2	209381.1	209755.8
8388608	197849.9	198743.8	198110.8	198473.2	196783.9	195587.5	194115.6	195059	194613.3

Average Seq Write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1566267.30	1742024.00	1780054.40	1739443.50	1568796.20	1372348.30	1273202.20	1196483.50	
16384	1540073.90	1605293.30	1640528.60	1625622.20	1484928.60	1336164.00	1293068.10	1287245.00	1202504.00
32768	1476558.20	1541503.50	1576472.90	1574097.60	1439611.80	1294668.60	1262216.10	1259761.30	1261652.60
65536	1450535.20	1513865.60	1547518.50	1549257.70	1423504.60	1272619.30	1245041.50	1245658.10	1248470.00
131072	1438796.10	1502099.70	1539203.30	1544154.50	1449160.70	1271716.80	1237253.00	1239175.70	1241592.10
262144	1429377.30	1492965.90	1532644.60	1540209.20	1468808.90	1273437.60	1233508.20	1235340.20	1238352.80
524288	1097314.80	1162120.90	1185616.50	1183158.90	1116529.40	983563.60	946937.80	923460.00	960860.00
1048576	458666.50	459154.80	463127.20	460503.60	453942.00	443502.40	435793.20	436140.20	437023.10
2097152	255593.10	257660.00	258260.70	257950.20	254443.00	252626.90	250757.80	249151.20	253656.00
4194304	218836.70	219691.70	219459.10	220464.80	218592.40	216067.90	216048.80	215401.60	215236.40
8388608	203083.00	203645.30	203290.80	202855.00	201901.30	200187.50	199346.20	199466.00	200192.20

A.11 Sequential Re-write Single Disk and Volume with Compression

Average Sequential write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1659309	1731771	1767535	1763490	1589870	1422504	1357570	1201473	
16384	1541850	1609039	1642824	1638782	1488929	1342364	1295581	1288869	1203844
32768	1482439	1546127	1581850	1578901	1441553	1301284	1267589	1263544	1264106
65536	1459156	1517348	1551807	1552297	1423388	1280351	1249711	1249680	1251350
131072	1444359	1505941	1538730	1545773	1434351	1274830	1242362	1243192	1246265
262144	1436152	1499498	1534338	1544863	1462035	1275698	1238448	1240459	1243240
524288	1130873	1178551	1170101	1205568	1097198	993039.8	937908.3	954516	967467
1048576	459245.6	468219.7	461887.6	462696.4	460334.1	437803.1	435780.8	437362.6	444072.3
2097152	256852.4	259621	259591.2	261825	256574	253902.9	252932.1	252466.7	251097.9
4194304	220349.5	221315.3	219532.8	220630.9	218369.1	216507.5	216709.6	215676.6	216656.8
8388608	204101.8	203502.2	203828.4	203489.2	201853.2	200304.2	199714	199647.8	199793.9

Average randum Write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1901466	2047671	2111579	2105591	1858013	1578589	1490103	1480224	
16384	1716146	1842457	1909314	1917900	1716237	1489488	1431936	1431245	1436733
32768	1638601	1754426	1819197	1830178	1653134	1453186	1406896	1406729	1407322
65536	1599070	1710343	1776432	1789337	1628952	1434407	1395779	1397196	1397923
131072	1576002	1692762	1756744	1774767	1624532	1428804	1391818	1393675	1395986
262144	1562715	1682462	1749049	1770270	1636045	1426216	1388466	1390811	1394474
524288	1480544	1574615	1601371	1538290	1400015	1161549	1114724	1097055	1047510
1048576	329733.2	354116.9	392133.7	417730.9	431963.2	441140.7	440934.7	429008.5	418837.6
2097152	170714.5	173048.6	187905.6	205733.4	224345.2	245495.7	253830.7	260798.3	262036.1
4194304	140490.8	135292.1	135964.6	141708.7	159851.7	193263.4	198794.1	208000	214078.6
8388608	131312.7	124147.1	122457.5	122227.3	122657.7	122577.9	123182.5	123928.2	123477.5

A.12 Random Read Single Disk with LZO Compression

	Average Rand Read								
	64	128	256	512	1024	2048	4096	8192	16384
8192	3774788.00	3864500.33	3892170.33	3755324.67	3252440.33	2483192.67	1907545.00	1751149.00	
16384	3788505.00	3861096.00	3885237.00	3791645.00	3269139.33	2462367.33	1865641.67	1665853.67	1640516.00
32768	3820641.00	3894171.00	3927097.00	3841215.67	3296698.67	2517176.33	1869552.67	1636566.00	1608476.67
65536	3848001.00	3937368.67	3984561.00	3916896.33	3372706.67	2498764.33	1869801.33	1621293.00	1592501.33
131072	3865408.00	3968463.00	4017771.00	3973555.33	3469190.33	2513767.67	1860388.00	1614125.00	1580923.67
262144	3879368.33	3982960.67	4044002.33	4024253.00	3599230.00	2536704.67	1847012.67	1609742.33	1577726.00
524288	3890754.00	4000071.00	4056615.67	4059120.33	3714019.00	2577047.33	1839335.00	1607718.33	1577300.67
1048576	3881248.67	3995546.00	4061562.00	4064358.00	3736917.67	2540271.67	1839842.00	1607657.00	1576103.00
2097152	3866644.00	3983917.67	4045124.67	4063770.00	3811722.67	2584520.00	1842259.67	1608981.33	1578607.00
4194304	92436.00	166483.67	259677.00	352633.67	444672.33	524778.00	579829.67	656380.67	698920.67
8388608	18343.67	33122.00	54539.33	85243.67	129517.33	176183.67	209296.00	242308.67	292931.33

A.13 Sequential Read Single Disk with LZO Compression

	Average Rand Read								
	64	128	256	512	1024	2048	4096	8192	16384
8192	3774788.00	3864500.33	3892170.33	3755324.67	3252440.33	2483192.67	1907545.00	1751149.00	
16384	3788505.00	3861096.00	3885237.00	3791645.00	3269139.33	2462367.33	1865641.67	1665853.67	1640516.00
32768	3820641.00	3894171.00	3927097.00	3841215.67	3296698.67	2517176.33	1869552.67	1636566.00	1608476.67
65536	3848001.00	3937368.67	3984561.00	3916896.33	3372706.67	2498764.33	1869801.33	1621293.00	1592501.33
131072	3865408.00	3968463.00	4017771.00	3973555.33	3469190.33	2513767.67	1860388.00	1614125.00	1580923.67
262144	3879368.33	3982960.67	4044002.33	4024253.00	3599230.00	2536704.67	1847012.67	1609742.33	1577726.00
524288	3890754.00	4000071.00	4056615.67	4059120.33	3714019.00	2577047.33	1839335.00	1607718.33	1577300.67
1048576	3881248.67	3995546.00	4061562.00	4064358.00	3736917.67	2540271.67	1839842.00	1607657.00	1576103.00
2097152	3866644.00	3983917.67	4045124.67	4063770.00	3811722.67	2584520.00	1842259.67	1608981.33	1578607.00
4194304	92436.00	166483.67	259677.00	352633.67	444672.33	524778.00	579829.67	656380.67	698920.67
8388608	18343.67	33122.00	54539.33	85243.67	129517.33	176183.67	209296.00	242308.67	292931.33

A.14 Strided Read Single Disk with LZO Compression

Average Strided Read									
	64	128	256	512	1024	2048	4096	8192	16384
8192	3810513.67	3882117.33	3866456.33	6109413.00	5388462.00	2985651.00	1975542.67	1747199.33	
16384	3807091.00	3866666.67	3904041.67	3794685.33	6005324.00	3208061.00	1912314.33	1683500.00	1648007.33
32768	3839427.33	3911671.00	3950666.33	3883045.33	3368908.33	3559062.67	1900828.00	1645444.33	1617464.67
65536	3862654.33	3947501.00	3988702.67	3926245.33	3364403.33	2514884.00	1916022.00	1621147.33	1593342.67
131072	3880626.33	3979635.67	4023763.00	3987664.67	3431829.00	2474523.00	1866763.00	1619748.33	1589412.67
262144	3907610.33	4004870.33	4053373.00	4042523.67	3617759.00	2482812.67	1840320.33	1612309.67	1583638.33
524288	3915023.33	4015065.00	4070369.33	4067616.33	3761345.67	2519300.33	1835619.33	1609746.33	1577040.67
1048576	3908790.00	4014184.67	4067959.00	4076900.00	3773387.00	2519885.00	1830892.67	1609058.00	1576904.67
2097152	3905655.67	4002498.33	4058295.00	4070334.67	3848876.67	2548174.33	1833204.00	1609327.67	1578819.33
4194304	364106.00	300987.00	271719.33	256732.67	531679.00	608154.67	702972.00	684880.33	716709.33
8388608	82980.67	84125.67	76088.67	74924.33	155121.67	363565.00	370312.67	329554.67	360303.67

A.15 Random Write Single Disk with LZO Compression

Average Rand wri									
	64	128	256	512	1024	2048	4096	8192	16384
8192	1831478.33	1991266.67	2074185.67	2054358.00	1829473.00	1587076.00	1492752.00	1483542.33	
16384	1655362.67	1800386.33	1882189.33	1872068.33	1697907.33	1500572.00	1434143.00	1425681.33	1427730.67
32768	1578114.00	1712611.67	1785388.33	1787450.33	1626609.33	1478027.00	1403696.33	1403783.67	1396165.00
65536	1542196.33	1668111.67	1746992.67	1752141.67	1604856.00	1447115.67	1395432.33	1388407.00	1391521.00
131072	1519872.67	1652574.33	1724668.00	1739408.67	1599593.00	1436553.00	1389478.33	1386787.33	1387620.33
262144	1510610.67	1641373.00	1720611.00	1738989.00	1608871.00	1426762.33	1382580.67	1384278.00	1386365.67
524288	1351980.67	1478413.00	1503608.33	1517525.00	1452923.00	1313151.67	1231029.67	1151153.33	1220339.33
1048576	976345.00	1056552.33	1085439.67	1069314.67	1058364.00	975227.67	986977.33	989386.33	950846.67
2097152	522483.33	669207.67	676865.00	686748.33	688456.33	683986.33	693620.33	702902.00	716459.00
4194304	431337.67	549611.67	486747.67	453040.33	445235.00	439181.67	472718.00	586170.67	504252.00
8388608	380321.67	494986.33	457488.00	461113.67	431004.67	475785.00	467232.67	466684.00	473332.00

A.16 Sequential Write Single Disk with LZO Compression Test Result

Average Seq Write									
	64	128	256	512	1024	2048	4096	8192	16384
8192	1566267.30	1742024.00	1780054.40	1739443.50	1568796.20	1372348.30	1273202.20	1196483.50	
16384	1540073.90	1605293.30	1640528.60	1625622.20	1484928.60	1336164.00	1293068.10	1287245.00	1202504.00
32768	1476558.20	1541503.50	1576472.90	1574097.60	1439611.80	1294668.60	1262216.10	1259761.30	1261652.60
65536	1450535.20	1513865.60	1547518.50	1549257.70	1423504.60	1272619.30	1245041.50	1245658.10	1248470.00
131072	1438796.10	1502099.70	1539203.30	1544154.50	1449160.70	1271716.80	1237253.00	1239175.70	1241592.10
262144	1429377.30	1492965.90	1532644.60	1540209.20	1468808.90	1273437.60	1233508.20	1235340.20	1238352.80
524288	1097314.80	1162120.90	1185616.50	1183158.90	1116529.40	983563.60	946937.80	923460.00	960860.00
1048576	458666.50	459154.80	463127.20	460503.60	453942.00	443502.40	435793.20	436140.20	437023.10
2097152	255593.10	257660.00	258260.70	257950.20	254443.00	252626.90	250757.80	249151.20	253656.00
4194304	218836.70	219691.70	219459.10	220464.80	218592.40	216067.90	216048.80	215401.60	215236.40
8388608	203083.00	203645.30	203290.80	202855.00	201901.30	200187.50	199346.20	199466.00	200192.20

A.17 Sequential Re-write Single Disk with LZO Compression Test Result

Average Seq Re-Write

	64	128	256	512	1024	2048	4096	8192	16384
8192	1810602.80	1917705.70	1969381.90	1955748.70	1757936.60	1548775.80	1485899.20	1471965.70	
16384	1680477.40	1773873.70	1825874.50	1826760.20	1648312.80	1460605.50	1411115.00	1402420.90	1409838.80
32768	1615167.90	1702917.40	1751215.30	1756232.00	1592133.10	1413954.80	1375879.60	1371009.60	1373267.20
65536	1583046.80	1671079.70	1717619.80	1728220.60	1575897.40	1393133.50	1358249.40	1358606.30	1360501.20
131072	1572400.80	1657511.50	1707891.50	1719943.90	1608330.20	1392700.10	1350260.50	1351076.00	1353440.20
262144	1558469.70	1647591.70	1698723.70	1714566.90	1631298.20	1394747.60	1346307.40	1347272.20	1350670.20
524288	1197935.60	1285313.00	1297247.20	1329698.40	1227782.00	1059739.60	1032223.80	1010371.00	1011431.00
1048576	456185.50	460069.70	461348.00	461720.20	460612.20	448716.00	437544.80	439039.30	436783.00
2097152	255590.30	254342.40	254466.50	256826.00	255465.50	250856.50	250217.00	252202.80	249249.50
4194304	210890.80	213474.50	213281.50	213026.40	211253.30	209029.70	209428.90	208055.40	208646.20
8388608	197391.40	198384.10	198661.70	198436.40	196585.00	195035.50	195053.50	194523.10	193749.00

Appendix B

Results of defrag tool run

Total Fragmentation in No.	Total Fragmentation in %	Time taken for defrag Tool Run 1	Fragmentation reduction in %	Time taken for defrag Tool Run 2	Fragmentation reduction in %	Time taken for defrag Tool Run 3	Fragmentation reduction in %
139	33.33%	125	27.55%	118	0.48%	2	100.00%
167	39.76%	161	32.14%	142	2.14%	6	100.00%
152	36.19%	162	28.83%	142	1.30%	4	100.00%
159	37.86%	150	32.14%	144	1.67%	4	100.00%
184	43.81%	183	37.38%	168	1.90%	6	100.00%
192	45.71%	192	39.53%	178	0.44%	2	100.00%
149	35.48%	137	28.81%	134	1.90%	5	100.00%
154	36.67%	145	30.00%	135	1.67%	3	100.00%
178	42.38%	180	35.48%	162	1.67%	5	100.00%
183	43.57%	183	37.62%	166	2.14%	6	100.00%
197	46.90%	203	39.29%	180	2.62%	8	100.00%
167	39.76%	159	33.81%	156	1.90%	5	100.00%
174	41.43%	169	34.29%	150	1.90%	5	100.00%
178	42.38%	176	36.19%	164	1.43%	4	100.00%
171	40.71%	168	35.00%	157	1.67%	5	100.00%

Appendix C

Script

C.1 Fragmentaion Tool Script

```
#!/usr/bin/perl
use Getopt::Std;
use strict "vars";
use feature ":5.10";
my $opt_string ='hs:m:';
getopts("$opt_string",\my %opt) or usage() and exit 1;
my @disk_free;
my @df_out;
my @size_list;
my $nb_fl;
my $path_dir;
my $bs_sz;
my $avl_spc;
my @fl_to_del;
my %seen;
my @crt_file;
my $numeric_val;
my $unit_val;
my $debug=1;
# show help message if -h option is given
 if ($opt{'h'}){
    usage();
    exit 0;
}
my $fl_sz = $opt{'s'};
my $ms_unit = $opt{'m'};
die " File size to be created is not provided\n" unless $fl_sz;
die " Measurement unit of file to be created is not provided\n" unless $ms_unit;

# convert user input to upper case
$ms_unit = uc($ms_unit);
```

```
# check the measurement unit entered is correct for this case k or m or g only
if (($ms_unit ne "K") && ($ms_unit ne "M") && ($ms_unit ne "G")){
    print "measurement unit in terms of K or M or G\n";
    exit 1;
}
# express block size of given file in terms of k , m or g
if (($fl_sz) && ($ms_unit)){
    if ($ms_unit eq "K"){
$bs_sz = 1 ."K";
    }elsif($ms_unit eq "M"){
$bs_sz = 1 ."M";
    }elsif ($ms_unit eq "G"){
$bs_sz = 1 ."G";
    }
}
# get the amount of available space on the given partition to fill it up
$path_dir = `pwd`;
@disk_free = `df -h $path_dir`;
# get the row that specifies avialble disk space from btrfs fi df command
foreach my $vals (@disk_free){
    @df_out = split(/\s+/,$vals);
}
$avl_spc = $df_out[3];
if ($avl_spc =~/(\d+)(\w+)/){
$numeric_val = $1;
$unit_val = $2;
    }
# Convert given file measurement unit into measurement unit of disk partion
if(($ms_unit eq "k") && ($unit_val eq "M" || $unit_val eq "MB")){
    $numeric_val = $numeric_val * 1024;
}elsif(($ms_unit eq "k") && ($unit_val eq "G" || $unit_val eq "GB")){
    $numeric_val = $numeric_val * 1048576;
}elsif(($ms_unit eq "M") && ($unit_val eq "K" || $unit_val eq "KB")){
    $fl_sz = $fl_sz * 1024;
}elsif(($ms_unit eq "M") && ($unit_val eq "G" || $unit_val eq "GB")){
    $numeric_val = $numeric_val * 1024;
}elsif(($ms_unit eq "G") && ($unit_val eq "K" || $unit_val eq "KB")){
    $fl_sz = $fl_sz * 1048576;
}
# check if there is enough space to create the required file
if ($fl_sz > $numeric_val){
    print " There is no enough to create the requested file size\n";
    exit 1;
}
# get the total disk size and subtract some value reverser working space
$nb_fl = int(($numeric_val - 128)/$fl_sz);
 # create a file with various size based on given size and
```

```
 @size_list = (($fl_sz - 2),($fl_sz - 1),$fl_sz,($fl_sz - 2),$fl_sz);
 # create the required files with varying size
 for(my $val = 0; $val < ($#size_list +1); $val++){
      if(@fl_to_del){
  for (my $i = 0; $i < ($#fl_to_del +1); $i++){
      my $nm = $fl_to_del[$i];
      system("dd if=/dev/zero of=$nm bs=$bs_sz count=$size_list[$val]");
      push(@crt_file,$nm);
         }
      }else{
     for (my $i = 1; $i <= $nb_fl; $i++){
       system("dd if=/dev/zero of=$i bs=$bs_sz count=$size_list[$val]");
       push(@crt_file,$i);
        }
      }
# generate random file names to delete
    @fl_to_del=();
    %seen =();
    my $haf_fl = int($nb_fl/2);
    print " files are going to be selected \n";
    for (1..$haf_fl){
my $rn_gen = int (rand($nb_fl)) +1;
redo if $seen{$rn_gen}++;
push(@fl_to_del,$rn_gen);
}
       # delete selected files
       for (my $dl =0; $dl < ($#fl_to_del +1); $dl++){
         my $delfle = $fl_to_del[$dl];
         system("rm $delfle");
        }
   # update the list of created file to only hold remaining files in the partition
    @crt_file =();
     open(FL,"ls $path_dir |");
       while ( my $line =<FL>){
    chomp($line);
    if($line =~ /^\d+$/){
        push (@crt_file,$line) ;
          }
       }
       close(FL);
}
sub usage {
print " Usage:\n";
print "-s size of file to be created\n";
print "-m measurement unit of file to be created\n";
print "./Script [-s][-m]\n";
}
```

C.2 Fragmentaion percentage report

```
#!/usr/bin/perl
use strict;

my $tot_no_fl = 0;
my $tot_no_ext = 0;
my $no_frag_fl = 0;

my $path = `pwd`;
chomp($path);
open (FL, "ls $path |");

  foreach my $file (<FL>)
  {
      my $flfrg_com = `filefrag $file`;
      if ($flfrg_com =~/.*:\s+(\d+).*/)
      {
  my $no_ext = $1;
  $tot_no_ext += $no_ext;
  if ($no_ext > 1)
  {
      $no_frag_fl++;
  }
  $tot_no_fl++;
      }
  }
close (FL);

my $frag_perc = sprintf("%.2f",(($no_frag_fl/$tot_no_fl) * 100));
print " $no_frag_fl non contigious files $frag_perc"."% \n";
```

C.3 defrag tool automation script

```
#!/usr/bin/perl
use strict;

my $tot_no_fl = 0;
my $tot_no_ext = 0;
my $no_frag_fl = 0;

my $path = `pwd`;
chomp($path);
open (FL, "ls $path |");

   foreach my $file (<FL>)
   {
      if ($file =~ /^\d+$/)
        {
   my $flfrg_com = `filefrag $file`;
   if ($flfrg_com =~/.*:\s+(\d+).*/)
   {
       my $no_ext = $1;
       if ($no_ext > 1)
       {
     system("btrfs fi defrag $file");
           }
       }
        }
    }
close (FL);
system("sync");
```

Appendix D

Computational Chemistry Test Input File

g09 application input file

```
%nproc=1
%mem=1gb
# ccsd/6-311G(df,pd) tran=full test nosymm

BENZENE d6h, rhf/6-31g* structure

0,1
C
H,1,RCH
C,1,RCC,2,120.00000
H,3,RCH,1,120.00000,2,0.00000,0
C,3,RCC,1,120.00000,4,180.00000,0
H,5,RCH,3,120.00000,4,0.00000,0
C,5,RCC,3,120.00000,6,180.00000,0
H,7,RCH,5,120.00000,6,0.00000,0
C,7,RCC,5,120.00000,8,180.00000,0
H,9,RCH,7,120.00000,8,0.00000,0
C,9,RCC,7,120.00000,10,180.00000,0
H,11,RCH,9,120.00000,10,0.00000,0

RCH=1.07560
RCC=1.38618
```