Ext4 file system in Linux Environment: Features and Performance Analysis

Borislav Djordjevic, Valentina Timcenko

***Abstract*—**This paper considers the characteristics and behavior of the modern 64-bit ext4 file system under the Linux operating system, kernel version 2.6. It also provides the performance comparison of ext4 file system with earlier ext3 and ext2 file systems. The work involves mathematical analysis of the file system access time with and without journaling option. The performance is measured using the Postmark benchmarking software that simulates the workload of Internet mail server. We have defined three types of workloads, generally dominated by relatively small objects. Test results have shown superiority of modern ext4 file system compared to its predecessors, ext2 and ext3 file systems. Benchmark results are interpreted based on mathematical model of file system access times.

***Keywords*—**Linux, File systems, ext4/ext3/ext2, journaling, inodes, file block allocation, disk performances.

I. Introduction

Ext4 file system is the ext3 file system successor. It is supported on today's most popular Linux distributions (RedHat, Ubuntu, Fedora). In contrast to the 32-bit ext3 file system [1] [2], [3] that has only some features added to its predecessor ext2 and maintains a data structure as in the ext2 file system, the ext4 file system has integrated more substantial changes compared to ext3. Ext4 has improved data structure and enhanced features, which brought more reliability and efficiency. It is a 64-bit, allowing the file size of up to 16 TB [4], [5], [6], [7]. Great efforts that have been put into the process of ext4 development resulted in new features and techniques: extents, journaling check summing, simultaneous allocation of multiple units, delayed allocation, faster fsck (file system check), and online defragmentation of small and large size directories. This way formed folders can have up to 64,000 files.

Manuscript received September 15, 2011: Revised version received Month Day, Year. This paper has been partially financed by Serbian Ministry of Science and Technological Development (Development Projects TR 32025, TR 32037 and III 43002).

B. Djordjevic is with the University of Belgrade, and Mihailo Pupin Institute, Volgina 15, 11060, Belgrade, Serbia (phone: +381112773383; fax: +381112775835; e-mail: bora@impcomputers.com).

V. Timcenko is with the University of Belgrade, and Mihailo Pupin Institute, Volgina 15, 11060, Belgrade, Serbia (phone: +381112774959; fax: +381112575978; e-mail: valentina.timcenko@institutepupin.com).

II. Problem Formulation

The main objective of this study was to notice the new features added to ext4, then examine the performance of modern file system ext4 in comparison of its characteristics and performances with its predecessor, ext3 and ext2 file systems, and finally identify the most dominant new features responsible for resulting differences in the performances. Ext4 file system includes many improvements, especially when comparing to ext3 file system, but being the 64-bit file system, ext4 is much cumbersome. This characteristic opens the possibility to obtain some unexpected results after performing the test procedures. A number of innovations have been added to ext4, and those are in detail explained in this chapter.

A. 64 bit file system

Ext4 is a 64-bit FS, allowing the file size reaches a size of up to 16 TB.

B. Inode size of 256 bytes

Namely, the default size of an index node in ext4 file system is 256 bytes. Index node greater than 128 bytes is required for storing timestamps, and extended attributes (eg, ACL lists) in the index node (thus, the extended attributes that do not fit in the i-node, are stored in separate blocks) [4][8]. Inode numbers in ext4 are 64bit (Figure 1). The lowest four to five bits of the inode number are dedicated to storage of the offset bits within the inode table block. The rest of inode number are for storage of the 32-bit block group number as well as 15-bit

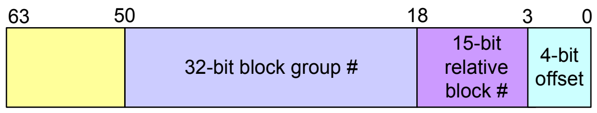

Fig. 1 64nit - inode layout

relative block number within the group. This inode structure permits a dynamic inode table allocation.

Ext4 file system is compatible with its predecessors, ext2 and ext3 file systems. After installing ext2 or ext3 file system, one can change a few options and use it as ext4 file system. Existing data will not be lost because ext4 file system will use new data structures only on newly formed data. Although this feature of ext4 file system is very useful, it is recommended to keep a backup copy of data on additional storage disk space. However, because of the differences in the data structure, there is slight limited compatibility between ext3 and ext4 file system, which in some cases reduces the possibility of using the ext4 file system and activate it as an ext3 file system.

C. *Extents*

One of the major differences between ext3 and ext4 file system is the way that block numbers are stored with data files. Ext3 uses indirect mapping blocks (Fig.2).

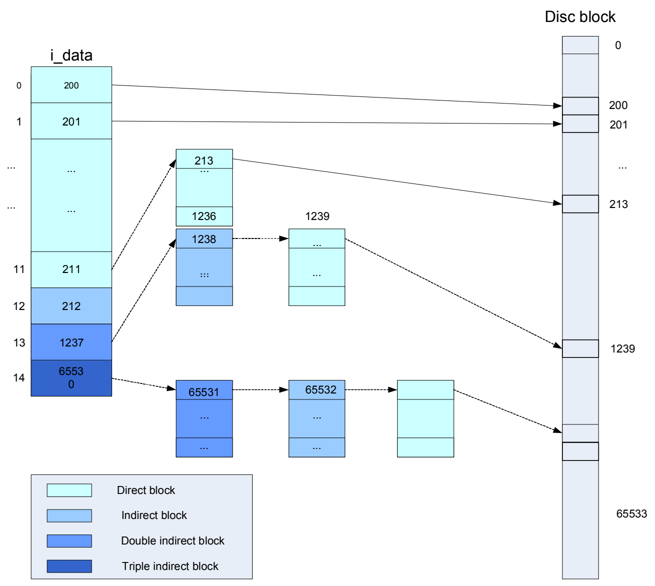

Fig. 2 ext2/ext3 mapping

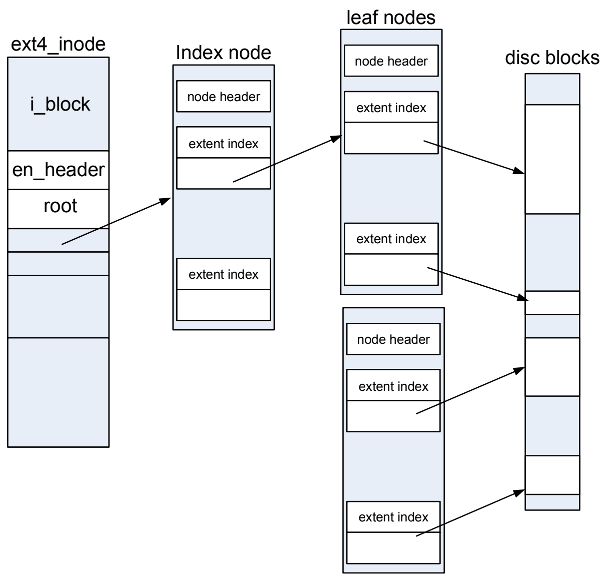

Fig. 3 Map of ext4 extents tree

This is effective when dealing with small and scattered files, but not efficient in dealing with large files, especially when deleting large files. Today, with the growth of the number of files with multimedia content and ever-faster internet, it seems inefficient to use this scheme. To solve this problem, instead of block mapping, the principle of mapping extents is being applied. This way, instead of mapping each block separately, ext4 remembers the number of blocks in extent (descriptor that represents a continuous series of physical blocks) [4][9][10]. Ext4 file system is based on a dynamic allocation of the i-nodes which provides good system performance, robustness and compatibility. To be able to perform successfully with different file sizes, ext4 has inherently implemented dynamic inode allocation's and 64-bit inode number. In addition, unlike ext3, regardless of the file size, ext4 file system ensures better file allocation based on a special block allocator, and specific strategies for different allocation requirements. For smaller size allocation requirements ext4 will try to allocate blocks from a one process group, which will be shared by all allocation requirements generated by the same process. For large size allocation requests, ext4 will allocate the space from i-node group. This way it is assured that the small files are stored continuously, one after another, which reduces fragmentation of large files that are stored in the same continuity (Fig. 3).

D. *Directory scalability*

In order to better support large directories with many entries, the directory indexing feature will be turned on by default in ext4. By default in ext3, directory entries are still stored in a linked list, which is very inefficient for directories with large numbers of entries. The directory indexing feature addresses this scalability issue by storing directory entries in a constant depth Htree data structure, which is a specialized BTree-like structure using 32-bit hashes. The fast lookup time of the HTree significantly improves performance on large directories [4][8]. For directories with more than 10,000 files, improvements were often with a factor of 50 to 100.

E. *Block allocation enhancement*

In order to meet the goal of the increased file system throughput, developers are constantly attempting to reduce file system fragmentation. The main problem is that high fragmentation rates cause greater disk access time affecting overall throughput. It has also impact on increased metadata overhead causing less efficient mapping. Many of new features do take advantage of the existing extents mapping and are aimed at reducing file system fragmentation by improving block allocation techniques.

F. *Persistant preallocation*

The ability to preallocate blocks for a file up-front, without having to initialize those blocks with valid data or zeros is a main advantage applied to some applications, like databases and streaming media servers. Enhancements provided by introduction preallocation have ensured adjacent allocation as far as possible for a file regardless the order and time that the data were written. It also guarantees space allocation for writes within the preallocated size, especially in the case where there is an information of required disc space for specific application. The file system internally interprets the preallocated but not yet initialized portions of the file as zero-filled blocks. This avoids exposing old data for each block until it is explicitly initialized through a subsequent write.

For applications concerning entirely sequential writes, it is possible to make a difference between initialized and uninitialized segments of the file. However, this is not sufficient good enough in the case when applying preallocation to databases and other similar applications. Than, random writes into the preallocated blocks can occur in any order. The file system needs to be able to identify ranges of uninitialized blocks in the middle of the file. Some extent based file systems (XFS, ext4), therefore provide support for marking allocated but uninitialized extents associated with a provided file.

Upon reads, an uninitialized extent is treated just like a hole, so that the VFS returns zero-filled blocks. During writes, the extent must be split into initialized and uninitialized extents, merging the initialized portion with an adjacent initialized extent if contiguous.

G. Delayed and multiple block allocation

Delayed and multiple block allocation can significantly improve file system performance on large I/O. In ext3, during the write operation the block allocator allocates one block at a time, which is inefficient for larger I/O. Since block allocation requests are passed through the VFS layer one at a time, the possibility of file fragmentation is increased due to the fact that the underlying ext3 file system cannot foresee and cluster future requests.

Delayed allocation is a distinguished technique in which block allocations are delayed to page flush time, rather than during the write() operation. This way it is possible to combine many block allocation requests into a single request, reducing possible fragmentation and saving CPU cycles. Another advantage of using delayed allocation is the possibility to avoid unnecessary block allocation for short-lived files.

H. Online defragmentation

Although applying many enhancements introduced into new file systems, there is still a possibility that with age, the used file system can still become quite fragmented. The ext4 online defragmentation tool, e4defrag, can defragment individual files or the entire filesystem, and this way helps avoid file fragmentation caused with file system aging.

I. Reliability

Ext3 is one of the most reliable file systems, therefore ext4 developers are putting much effort into maintaining the reliability of the file system to make it even more reliable [4][11][12][13]. This supposes to make ext4 fields 64-bits in size, but the problem might be to make such large amounts of space actually usable in the real world.

J. e2fsck and unused inode count

In e2fsck, the checking of inodes in pass 1 (phase 1) is by far the most time consuming part of the operation. This operation supposes the following: reading all of the large inode tables from disk, scanning them for valid, invalid, or unused inodes, and then verifying and updating the block and inode allocation bitmaps. Pass 1 scaning can be extremely lengthy so when applying the uninitialized groups and inode table high watermark feature it can be successfuly skipped. This can drastically reduce the total time taken by e2fsck (by 2 to 20 times), depending on how full the file system is.

This feature guarantees that the kernel stores the number of unused inodes at the end of each block group's inode table. As a result, e2fsck can skip not only the operation of reading these blocks from disk, but also scanning them for in-use inodes.

K. Checksumming enhancements

During usual journal operation the commit block is not sent to the disk until the transaction header and all metadata blocks which make up that transaction have been written to disk.

With this two-phase commit, if the commit block has the same transaction number as the header block, it should indicate that the transaction can be replayed at recovery time. If they do not match, the journal recovery is ended. However, there is possibility that this procedure ends wrong and lead to file system corruption.

Metadata checksumming when added into ext4 allows easier corruption detection. The checksum is also integrated into the group descriptors and into journaling. Therefore, in ext3 and ext4 each journal transaction has a header block and commit block.

With journal checksumming, the journal code computes a CRC32 over all of the blocks in the transaction including the header, and the calculated checksum is further written to the commit block of the transaction. If the case that the checksum does not match at journal recovery time, it is an indication of the corruption of one or more metadata blocks in the transaction or unsuccessful writing of the metadata blocks to disk. The concerned transaction, along with later ones, will be discarded as if the computer had crashed slightly earlier and not written a commit clock at all.

There is need for having a two-phase commit for each transaction since the journal checksum in the commit block allows detection of blocks that were not written into the journal. The commit block can be written at the same time as the rest of the blocks in the transaction, which can speed up the file system operation noticeably (as much as 20%, instead of the journal checksum being an overhead).

III. Workload specifications

File-based workloads are designed for testing procedures of file systems, and usually consist of large number of file operations (creation, reading, writing, appending and file deletion). It comprises large number of files and data transactions. Workload can be generated synthetically, as a result of applying benchmark software, or as a result of working with some real data applications.

Workload characterization is a hard research problem [14], [15] as arbitrarily complex patterns can frequently occur. In particular, some authors chose to emphasize support for spatial locality in the form of runs of requests to contiguous data, and temporal locality on the form of bursty arrival patterns. Some authors [15] model three different arrival processes:

- *Constant*: the interarrival time between requests is fixed

- *Poisson*: the interarrival time between requests is independent and exponentially distributed
- *Bursty*: some of the requests arrive sufficiently close to each other so that their interarrival time is less than the service time.

IV. Access time for ext4 and ext3 file systems

In this subchapter we will present the access time value comparison for 64-bit ext4 and its 32-bit predecessors, ext3 and ext2. Expected access time for file system, for specified workload, comprises following components:

$$E[FS\_access] = E[directory\_op] + E[metadata\_op] + E[free\_list\_mngm] + E[direct\_file\_access] \tag{1}$$

Where FS_access represents total time that workload needs to carry out all operations, directory_op represent total time for all operations related to working with directories (search, new object creation and delete of existing objects), metadata_op represents total time needed for performing metadata operations (metadata search operations, metadata cache forming, and metadata objects modifications), free_list_mngm presents total time needed for performing operations with free blocks and inode lists (file expansion/shrinking, creation of new objects or deletion of existing objects), and direct_file_access is the time required for direct file blocks operations (read/write). Directory operations, metadata operations and direct file accesses are cache based accesses and their performances directly depend on cache. Under these conditions, the expected effective access time would be:

$$E[Cache\_Service\_Time] \approx Hit\_prob * \frac{E[request\_size]}{Cache\_Transfer\_Rate} + Miss\_prob \cdot E[Mech\_Service\_Time] \tag{2}$$

Mech_Service_Time is the total time needed to perform disk data transfer.

In the case of cache-miss, performances are strictly dependent on disc characteristics and consist of number of time based components:

$$Mech\_Service\_time = Taccess\_time + Tmedia\_time + TInterface\_time \tag{3}$$

Where *Taccess_time* is total time needed for mechanical components of disk transfer, *Tmedia_time* is total time needed for write/read operstions from disc medium, and *Tinterface_time* is total time needed for read/write operations from disc cache buffer.

$$Taccess\_time = CommandOverhead + SeekTime + SettleTime + RotationalLatency \tag{4}$$

Where *CommandOverhead* time is time required for disc commands decoding, *SeekTime* is time needed for disc servo system positioning, *SettleTime* is time required for disc head stabilization, and *RotationalLatency* is time wasted on disc rotation latency.

There are three dominant components, whose sum can be presented as *Mech Service Time*, which is the service time for a request related to the disk mechanism. These components are: (1) the seek time (*SeekTime*), which is the amount of time needed to move the disk heads to the desired cylinder; (2) the rotational latency (*RotationalLatency*), which is the time that the platter needs to rotate to the desired sector; and (3) the transfer time (*TT*) same as *Tmedia time,* which is the time needed to transfer the data from the disk mechanism to the next higher level. Since these are typically independent variables, we can approximate the expected value of the disk mechanism service time as:

$$E[Mech\_Service\_Time] = E[SeekTime] + E[RotationalLatency] + E[TT[request\_size]] \tag{5}$$

The transfer time (*TT*) is a function of two parameters, the *Transfer_rate,* which is the transfer rate of data off/onto the disk and *E[request_size]*. Function can be approximated as:

$$TT[request\_size] = E[request\_size] / E[Tranfer\_rate] \tag{6}$$

Variations will occur as a result of track and cylinder switches and different track sizes in different zones on the disk.

The *SeekTime* can be approximated as the following function of *dis*, which is the number of cylinders to be travelled:

$$SeekTime[dis] = \begin{Bmatrix} 0 & dis = 0 \\ a + b\sqrt{dis} & 0 \prec dis \le e \\ c + d \cdot dis & dis \succ e \end{Bmatrix} \tag{7}$$

where a, b, c, d, and e are disk -specific parameters.

If we assume that the requests are randomly distributed on the sectors of the given cylinder using a uniform distribution, than:

$$E[Rotational\_latency] = revolution\_time / 2 \tag{8}$$

A. Expected behaviour of ext4/ext3 file systems

Starting from formula (1) which is appropriate for non-journaling file systems, for the case of journaling based file systems, as ext4 and ext3 are, one additional member has to be added, $E[journaling\_time]$, which can increase reliability but can also have negative impact to the global performances:

$$\begin{aligned}&E[FS\_access] = E[directory\_op] + E[metadata\_op]\\&+ E[free\_list\_mngm] +\\&E[direct\_file\_accesses] + E[journaling\_time]\end{aligned} \tag{9}$$

journaling_time is total time needed for performance of journaling operations (metadata writes to the log, and metadata log discharge).

At first glance we could assume that perhaps journaling techniques will decrease file system performances, but the fact is that this statement can not be taken for sure. Instead of math simulated workload, we have applied synthetically generated workload in Postmark benchmark environment.

V. Problem Solution

For purposes of testing the chosen file systems we have used the Postmark Benchmark [16] software. It simulates the Internet Mail server load. Postmark creates large initial set (pool) of randomly generated files, and saves them in any location in the file system. Over this set of files Postmark and the operating system perform operations of creation, reading, registration and deletion of files and determine the time required to perform these operations. The operations are performed randomly in order to provide the credibility of the simulation. Number of files, their size range and number of transactions are fully configurable. With the aim to eliminate the cache effect it is recommendable to create large set of files (at least 10,000) and execution of a large number of transactions over the generated files.

We have presented the results of three different test procedures. The first one, Test1, is based on testing of small files (1K-100K) and it will be a reference when comparing other test results. Next testing procedure, Test2, considers drastically smaller files (1byte-1K) and appreciably increased number of generated files, which will generate high number of metadata operations with ultra small objects. Test3 considers slightly increased size of generated files when comparing to Test1, which implies higher dataflow in workload.

For testing purposes we have chosen disks series HP SAS 10K. These are 3Gb SAS drives, 2.5 inch and 146GB capacity (Table I).

The hardware configuration assumes several basic components, and it is presented in Table II. As for the operating system, it was chosen one of the most popular Linux distributions for the PC architecture, Red Hat Linux Fedora 13 with kernel version 2.6.33.3-85.fc13.

Table I Disc characteristics

HP Invent SAS 10K, 146GB, 2.5", Hot-swap HD	
Capacity	146GB
Interface	SAS plug
Average seek time	4 ms
Full stroke seek	8.1msec
Track-to-track seek	0.2msec
Rotational speed	10,000 rpm
Max. buffer throughput	6Gb/sec

Table II. Hardware configuration.

Server	HP Proliant ML350 G6
RAM	12 GB
Processors	Intel(R)Xeon(R)
CPU Model	Quad-core E5506@2.13GHz
Number of CPU kernels	4
CPU speed	2333MHz
L2 cache	2 x 6 MB
Controllers	
RAID	HP Smart Array P410i SAS
RAID cache memory	256MB
Disc (DualPort)	HP Invent SAS 10K, 146GB,2.5"
Operating system	Linux Fedora 13, kernel- 2.6.33.3-85.fc13
Number of CPU kernels	4
CPU speed	2333MHz

Filesystem is organized in the form of LVM partitions [17], as presented in Table III.

Table III File system layout

Filesystem	Size	Mounted on
LogVol00	130G	/ root FS
LogVol01	5G	/ swap
LogVol02	10G	/ testing FS

Swap is defined as 5GB partition and implemented as a logical group LogVol01. This partition can be found on the testing system by following the path /dev/mapper/VolGroup00-LogVol01. In the logical group LogVol02, 10GB in size we have created empty ext2, ext3, ext4 file system, respectively. It is used for testing purposes.

A. Postmark Test1 (small files)

Files used for the purpose of performing this testing procedure are relatively small, ranging from 1K to 100K. Appropriate Postmark configuration used for this test was: file size range from 1000 to 100000, number of generated files was 4000, and number of performed transactions was 50000. Performance results for each file operation (creation, creation/alone, creation/with transactions, reading, appending, deleting, deleting/alone, deleting/with transactions) are given on the Figures 4a and 4b. Operations delete/alone and create/alone are performed in special benchmark phases, and do not suppose transactions. Though, they do have high values.

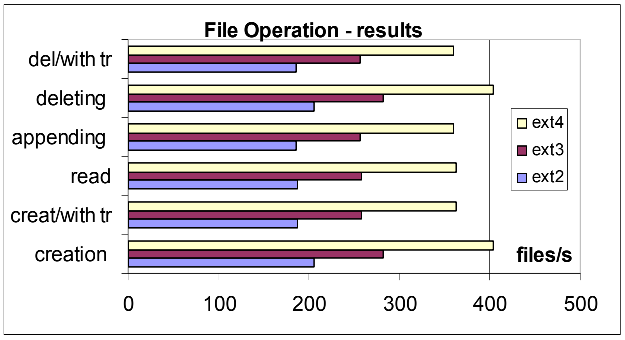

Fig. 4a File operations with transactions

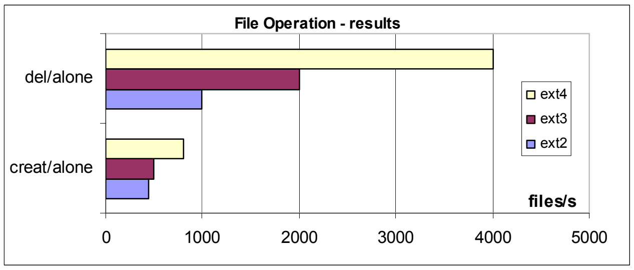

Fig. 4b File operations without transactions

This configuration generates about 1.6GB of read/write data. Obtained test results for data flow are presented in Figure 4c:

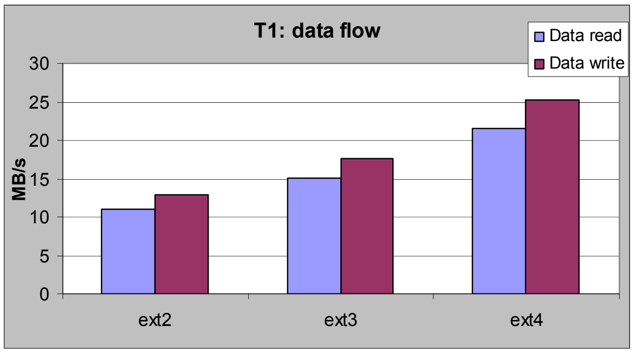

Fig. 4c Test 1 data flow results

In this test ext4 file system has shown superior performances comparing to the performances of its predecessors. Ext4 is more than 40% faster than ext3, and almost twice as fast as ext2. On the other hand, ext3 is more than 35% faster than ext2.

The reasons for this behavior can be found in the fact that ext4 has integrated a number of innovative file allocation techniques, which are extents, enhanced journaling techniques and an improved buffer cache mechanism. The workload for this test is characterized with lower number of files (4000), moderate number of create/delete operations, file size of 1k-100k, and solid amount of reads/writes (1.6GB-r/1.8GB-w). Having into consideration the formula [17], it can be expected that $E[direct\_file\_access]$ component will be dominant. Obtained results have shown better write performances in case of ext4 file system, and the reason for this behaviour are features obtained by applying delayed allocation and multiblock allocators. Besides, the presence of extents and persistent pre-allocation has provided less fragmented files. Write performances are also improved with applied different techniques for journaling enhancement, as journal checksumming and others. This characteristic positively implies the read performances as well. Certain influence to read and write performances, and especially to creation/deletion of files, has feature *Htree* indexing for directories. This performance difference is obvious when considering part of the test deleting/alone, when Postmark creates and than deletes large number of files. It is obvious that ext4 file system among other enhancements, uses *Htree* directories, ext3 doesn't use it regardless of the fact that it has it included into the package, and ext2 even doesn't have this possibility at all.

The obtained results have confirmed that both journaling file systems, ext3 and ext4, have considerably better performances than ext2. To conclude, the journaling techniques combined with the cache mechanism, not only have not slow down the system, as we have expected when thinking about greater robustness of ext4, but have significantly improved performances.

B. Postmark Test2 (ultra small files)

This is also a very intensive test procedure as it involves a large number of very small files, ranging from 1bytes to 1K. Used Postmark configuration was: file size range from 1 to 1000, number of generated files was 30000 and number of performed transactions was 50000. Performance results for each file operation are given on the Figures 5a and 5b.

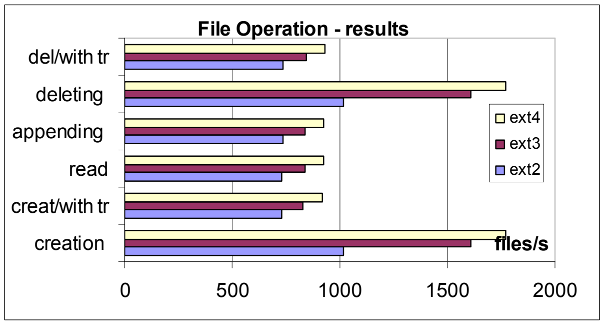

Fig. 5a File operations with transactions

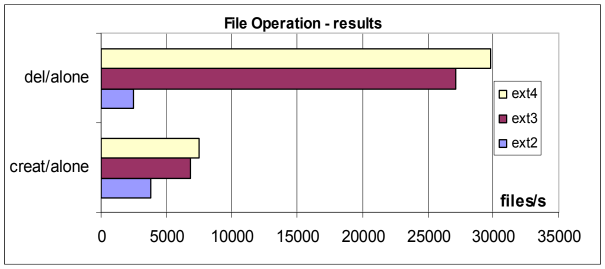

Fig. 5b File operations without transactions

Now, the number of created files is increased to 30000, which brings about 14MB of read data, and 32MB of written data. The testing procedure generates a large number of metadata and I/O requests. Obtained results for data flow are presented in Figure 5c:

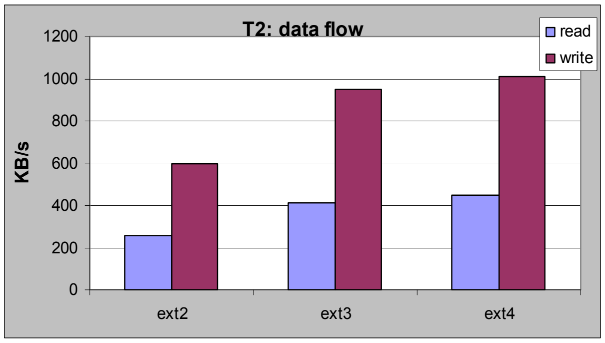

Fig. 5c Test 2 data flow results

In this testing procedure with ultra small files, ext4 file system has also shown better results comparing to its predecessors, although the differences are smaller than in the previous test. Ext4 file system is more than 10% faster than ext3, while it is more than 70% faster than ext2. At the same time, ext3 file system is more than 60% faster than ext2. The workload for this test is characterized with: high number of files (30000), high number of create/delete operations, ultra small file sizes (1byte-1K), and low amount of reads/writes (13.61MB-r/31.35MB-w). Having into consideration the formula [17], it can be expected that the $E[directory\_op] + E[metadata\_op]$ components will be dominant. The main performance difference is noticed when Postmark creates and further deletes files alone. *Htree* indexing for directories has major impact onto performances, and this feature is turned on by default in ext4, while in the case of ext3 directory entries are by default stored in a linked list, making it inefficient when using for directories with large numbers of entries. In this test, ext3 has turned on *Htree* indexing, while ext2 performs as linked list and this fact makes the performance difference even more noticeable. There is also an innovative technique implemented in ext4 based on putting the whole inode into the directory instead of just a directory entry that references a separate inode. This avoids the need to seek to the inode when doing a *readdir*, because the whole inode has been read into memory already in the *readdir* step. If the blocks that make up the directory are efficiently allocated, then reading the directory also does not require any further seeking. To conclude, ext4 read and write performances are slightly better than in the case of ext3/ext2 primarily because of applied techniques delayed allocation and multiblock allocator. As this is the test with ultra small files, techniques as extents and persistent pre-allocation have no impact on performances, and buffer cache mechanism absorbs differences between ext4 and ext3 file systems. This test shows that both journaling FS, ext3 and ext4, are considerably better than ext2, which means journaling techniques combined with the cache mechanism do improve performances.

C. *Postmark Test3 (larger files)*

This is a very intensive test. Files used for the purpose of performing this testing procedure are relatively large, ranging from 1K to 300K.

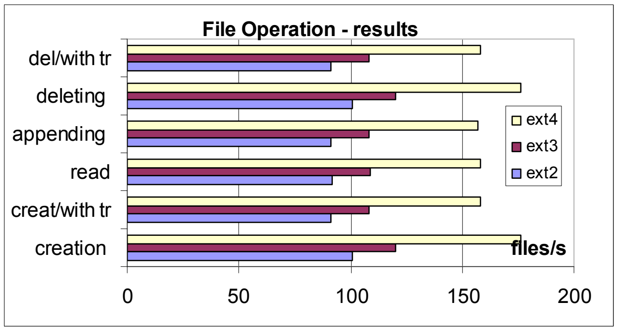

Fig. 6a File operations with transactions

Postmark configuration used for this test was: file size range from 1000 to 300000, number of generated files was 4000, and number of performed transactions was 50000. Performance results for each file operation are given on the Figures 6a and 6b.

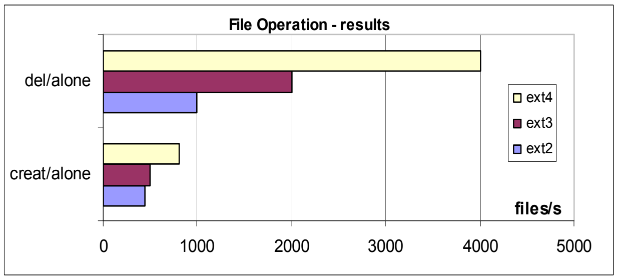

Fig. 6b File operations without transactions

The total amount of data to be read from the disc is 4.7GB, and 5.4GB to be written. Test results for data flow are shown in Figure 6c:

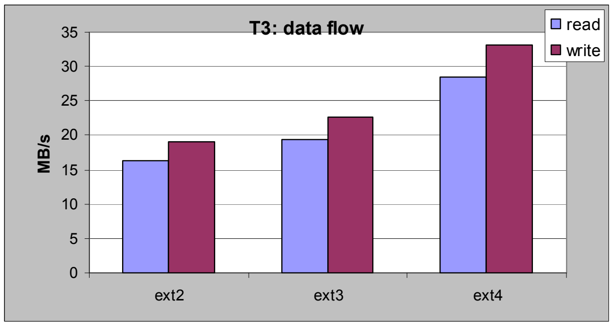

Fig. 6c Test 3 data flow results

In this test, we have implemented larger files, and approved that ext4 file system continues to show superior performance compared to its two predecessors. Ext4 file system is more than 46% faster than ext3, while about 73% faster than ext2. At the same time, ext3 file system is over 18% faster than ext2. The reasons for this behaviour are numerous innovative allocate/search files techniques implemented in ext4, but major impact is on the extents, while journaling techniques and metadata operations during large transfers have less impact on performances. This is the reason for slight differences in performances, especially when comparing ext2 and ext3.

The workload for this test is characterized with lower number of files (4000), moderate number of create/delete operations, file size of 100k-300k, and solid amount of reads/writes(4.6GB-r/5.4GB-w). Having into consideration the formula [17], it can be expected that $E[direct\_file\_accesses]$ component will be dominant. Same as in first test, it is noticeable that the obtained results have shown better write performances in case of ext4 file system, and the reason for this behaviour are features obtained by applying delayed allocation and multiblock allocators, and that the presence of extents and persistent pre-allocation have provided less fragmented files (specially present when considering bigger files – 300K). Write performances are also improved with applied different techniques for journaling enhancement, as journal checksumming and others. This characteristic positively implies the read performances as well. Similar to the situation with the results in the first test, certain influence to read and write performances, and especially to creation/deletion of files, has feature *Htree* indexing for directories. This performance difference is obvious when considering part of the test deleting/alone when Postmark creates and than deletes large number of files. It is obvious that ext4 file system among other enhancements, uses *Htree* directories, ext3 doesn't use it regardless of the fact that it has it included into the package, and ext2 even doesn't have this possibility at all. The obtained results have confirmed that both journaling file systems, ext3 and ext4, have considerably better performances than ext2. To conclude, the journaling techniques combined with the cache mechanism, not only have not slown down the system, as we have expected when thinking about greater robustness of ext4, but have significantly improved performances.

VI. Conclusion

In this paper, we have summarized the results of Postmark testing procedures for three mutually compatible file systems ext2, ext3 and ext4. Obtained results have confirmed majority of our expectations. A new, modern, 64-bit file system ext4 has shown superior characteristics when compared to its two predecessors, ext3 and ext2. Its performances are superior in all three test procedures. A number of innovative techniques for file allocation (extents, persistent pre-allocation, delayed allocation and multiblock allocator), and enhanced *Htree* indexing for larger directories which is always on, improved journaling techniques and buffer cache mechanism. This way ext4 file system has shown superior performances in comparison to its predecessor in difficult conditions, such as working with small files that are mostly used for in our experiments. It can be concluded that ext4 performs better than any of its predecessors, which means that journaling techniques combined with the cache mechanism, not only have not slow down the system, but improved its performance. The obtained results are encouraging for all Linux users to use ext4 file system.

Future work will be focused on testing ext4 file system and its predecessors in RAID configurations, as well as in other benchmark environments such as IOzone, FFSB, Bonnie,, and for FAT and NTFS sile systems [18].

References

[1] M. Seltzer et al., “Journaling versus Soft Updates: Asynchronous Meta-data Protection in File System”, in *USENIX Conference Proceedings*, San Diego, USA, June 2000, pp. 71-84.

[2] Tweedie S., “EXT3, Journaling File system”, presented at the Ottawa Linux Symposium, Ottawa Congress Centre, Ottawa, Ontario, Canada, 20 July, 2000.

[3] Stergious Papadimitrou Konstatinos Terzidis, “Comparative evaluation of the recent Linux and Solaris kernel architectures”, in *Proceedings of the 11th WSEAS International Conference on COMPUTERS*, Agios Nikolaos, Crete Island, Greece, July, 2007, pp 460-463.

[4] 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.

[5] Roderick W. Smith, “Migrating to Ext4”, DeveloperWorks. IBM, 2008 [Online]. Available : www.ibm.com/developerworks/linux/library/l-ext4/

[6] Wikipedia, Ext4 Howto, January 2011, [Online]. Available: ext4.wiki.kernel.org/index.php/Ext4_Howto

[7] First benchmarks of ext4, Oct 2006, [Online]. Available: Linuxinsight: http://www.linuxinsight.com/first_benchmarks_of_the_ext4_file_system.html

[8] G.B.Kim, D.J.Kang, C.S.Park, Y.J.Lee, B.J.Shin, “A Dynamic Bitmap for Huge File System in SANs”, in *Proceedings of the 6th WSEAS International Multiconference onCircuits, Systems, Communications and Computers (CSCC 2002)*, Crete, Greece, 2002, [Online]. Available: www.wseas.us/e-library/conferences/crete2002/papers/444-415.pdf

[9] Jaechun No, “A Design for Hybrid File System”, in *Proceedings of the The 8th WSEAS International Conference on ENVIRONMENT, ECOSYSTEMS and DEVELOPMENT (EED '10),* Vouliagmeni, Athens, Greece, December, 2010, pp. 143-148, ISBN: 978-960-474-260-8 [Online]. Available: www.wseas.us/e-library/conferences/2010/Vouliagmeni/SAM-22.pdf,

[10] Jinsun Suk and Jaechun No, “HybridFS: Integrating NAND Flash-Based SSD and HDD for Hybrid File System”, in *the Proceedings of the 10th WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS (AIC '10),* Taipei, Taiwan, August, 2010, pp 178-185, ISSN: 1792-4626 [Online]. Available: www.wseas.us/e-library/conferences/2010/Taipei/ISTASC-26.pdf,

[11] Jaechun No, “Snapshot-Based Data Recovery Approach”, in *the Proceedings of the the 14th WSEAS International Conference on SYSTEMS*, Corfu Island, Greece July, 2010, pp -160-165, ISBN: 978-1-61804-023-7. [Online]. Available: www.wseas.us/e-library/conferences/2010/Corfu/SYSTEMS-24.pdf.

[12] Gyoung-Bae Kim, Chang-Soo Kim, Bum-Joo Shin, “A 64-bit, Scalable File System for Storage Area Networks”, in *the Proceedings of Proceedings of the 5th WSES International Conference on Circuits, Systems, Communications and Computers (CSCC 2001)* , Rethymno, Greece, July, 2001. [Online]. Available: www.wseas.us/e-library/conferences/crete2001/papers/736.pdf

[13] Ruo Ando, Hideaki Miura*,Yoshiyasu Takefuji, “File system driver filtering against metamorphic viral coding”, in *the Proceedings of the 3rd WSEAS Inernational Conference on INFORMATION SECURITY, HARDWARE/SOFTWARE CODESIGN and COMPUTER NETWORKS (ISCOCO 2004)*, Rio De Janeiro, Brazil, pp217-222. [Online]. Available: www.wseas.us/e-library/conferences/brazil2004/papers/470-247.pdf

[14] Daniel L Martens and Michael J. Katachabaw, “Disk Access Analysis for System Performance Optimization”, in the Proceedings of the 5th WSEAS International Conference on APPLIED COMPUTER

SCIENCE (ACOS '06), Hangzhou, China, April, 2006,. [Online]. Available: www.wseas.us/e-library/conferences/2006hangzhou/papers/531-316.pdf.

[15] Elizabeth Shriver, Arif Merchanty, and John Wilkesy, “An analytic behaviour model for disk drives with readahead caches and request reordering”, in the Proceedings of the SIGMETRICS 98/PERFORMANCE '98, 1998 ACM SIGMETRICS joint international conference on Measurement and modelling of computer systems.

[16] J. Katcher, “PostMark: A New File System Benchmark”, 1997, Technical Report TR3022. Network Appliance Inc.

[17] V. Danen, “Set up Logical Volume Manager in Linux”, Mar 09 2007, TexhRepublic[Online]. Available: http://www.techrepublic.com/article/set-up-logical-volume-manager-in-linux/6166001

[18] Faraz Ahsan et al, “Exploring the Effect of Directory Depth on File Access for FAT and NTFS File Systems”, in the Proceedings of the 8th WSEAS International Conference on SYSTEMS THEORY and SCIENTIFIC COMPUTATION (ISTASC’08), Rhodes, Greece, August, 2008,pp 130-135, ISSN: 1790-2769. [Online]. Available: www.wseas.us/e-library/conferences/2008/rhodes/istasc/istasc18.pdf.

Borislav S. Djordjevic was born in Pirot, Serbia in 1964. He received his B.Sc. Electrical Engineering degree, from University of Belgrade, Serbia, Elektronics specialization in 1989, M. Sc. degree in ICT field in 1992, and Ph.D. degree in ICT field in 2003.. Her research interests are operating systems, communication networks, data protection, disc and filesystem optimization and connecting UNIX with different Operating Systems. She has joined Institute Mihailo Pupin, Belgrade, Serbia – Computer System division in 1989 and now works in research and development as Research Associate. The most important references are:

1. B. Djordjevic, S. Miskovic, “Disk Interface comparison and Operating System file-caching investigation”, Microprocessors and Microsystems, Elsevier Science Volume/Issue 27/4 pp. 181-198 (received 10 July 2002, revised 1 December 2002, accepted 10 January 2003)

2. Dragan Pleskonjic, Nemanja Macek, Borislav Djordjevic, Marko Caric: "Security of Computer Systems and Networks" Book Preview. Comput. Sci. Inf. Syst. 4(1): 77-92 (2007). Volume 4, Number 2, December 2007, Editorial: Marjan Mernik, ISSN: 1820-0214, [Online]. Available: http://www.comsis.org/ComSIS/Vol4No1/BookPreview/Book.htm

3. B. Djordjevic, V. Timcenko, " Ext4 File System Performance Analysis in Linux Environment", 11th WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS (AIC '11), Florence, Italy, August 23-25, 2011

Mr. **Borislav S. Djordjevic** is for many years an IEEE member. He is also member of Serbian ETRAN and TELFOR society.

Valentina V. Timcenko was born in Belgrade, Serbia in 1978. She received her B.Sc. Electrical Engineering degree, from University of Belgrade, Serbia, Telecommunication specialization in 2004, and M. Sc. degree in the same field in 2010. She is currently working for her Ph.D. degree. In 2005, she has received CCNA title from Cisco Systems. Her research interests are communication networks, data protection, disc optimization and connecting UNIX with different Operating Systems. She has joined Institute Mihailo Pupin, Belgrade, Serbia – Telecommunications division in 2004 and now works in research and development as associate researcher. The most important references are:

1. Timčenko V., Stojanović M., Boštjančič Rakas S. MANET Routing Protocols vs. Mobility Models: Performance Analysis and Comparison" // Proceedings of the 9th WSEAS International Conference on Applied Informatics and Communications (AIC '09). – August, 2009. – P. 271–276.
2. Timčenko V., Stojanović M., Boštjančič Rakas S. A Simulation Study of MANET Routing Protocols Using Mobility Models // Computers and Simulation in Modern Science (Vol. III). – WSEAS Press, 2010. – P. 186–196.
3. B. Djordjevic, V. Timcenko, "Ext4 File System Performance Analysis in Linux Environment", 11th WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS (AIC '11), Florence, Italy, August 23-25, 2011

Ms.Valentina Timcenko is for many years an IEEE member. She is also member of Serbian ETRAN and TELFOR society, and received an award for paper published on ETRAN conference in 2008.