IJCSI International Journal of Computer Science Issues, Volume 13, Issue 6, November 2016
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.20943/01201606.6167

Sext3: The Secure Ext3

Muhammad Raza[1], Ke Zhou[2] and Basheer Riskhan[3]

[1, 2, 3] School of Computer Science, Huazhong University of Science and Technology, Wuhan, Hubei, P.R. China.

Abstract

Harboring bug may leads system crashes, file system bugs hence corrupt file systems and cause data loss, regardless of the existence of journals and similar defensive techniques. While consistency checkers such as fsck can detect this corruption and restore a damaged image to a usable state, they are generally created as an afterthought, to be run only at rare intervals. But issue is that the fsck is the utility for checking errors in Ext3 file system. In our opinion file system must has its own checker for silent killer known as harboring bug. So Sext3 has this ability and does not need third party checking utility for harboring bug. These modifications in system file allow Sext3 to check system in real time scenario. If we use third party utility the system needs to invoke that utility every time when system is writing data to any field. But in our modification of Ext3 there is no need to invoke the application because the checker is the part of file system. In our proposed solution high memory consumption is only drawback. In addition, we demonstrate the Sext3 performance and competitively with Ext3.

Keywords: *Sext3, Ext3, fsck*

1. Introduction

File system is the fundamental building block of any Operating System. Such as NTFS is for Microsoft Windows and EXT3 is for Linux. To make system intact it is necessary to guaranty data integrity for personal or enterprise computation. It will be challenging to restore the data once it becomes corrupted. Before implementing any file system engineers judge file system's robustness and reliability.

It is necessity to make your data dependable. Once your data lost or corrupted (mistake made by human or by machine) it is very difficult to restore data or make data available in future [22]. In the war between errors and developers must make sure that there developed file system must be robust and reliable enough to prevent ad hoc or permanent data corruption or data losses.

There are many factors that are involve to corrupt the data such as unexpressed shutdown, bug in data of system file. Conflicts between file system and device drivers are also a fact that can easily crash the system [13]. Not only file system data play an important role in data corruption, hardware failure is also a critical issue in overall system crash. Hardware may include primary memory (RAM) or may be secondary memory (HDD) [1, 4, 5, 20]. So meta-data and file system must be robust and fault tolerant and can prevent the propagation of bug and errors that leads system permanent data lost.

From many years scientists are working on file system to make it reliable and robust enough to resist against corruption. Such as journaling [8], copy-on-write [21] and soft update [17] having the ability to prevent system corruption and unexpressed crashes. Another value able source that can prevent and remove errors form code of system and prevent the corruption in it known as bug finding tools [14, 19]. Hardware failures can be detected by different checksums [6, 9].

Most of the solution can reduce the chance of error in system and has the ability to repair it. But unfortunately these solutions cannot protect system from all the faults, errors and bugs. The error that comes silently when copying or update the date in system file. These types of error are silent killer of files system integrity. This type of error is known as harboring error. It comes in the system very silently and cannot be detected when it copy itself in the system. System file seem correct and error free. It can stay in system silently from days to month in system. When system needs file or execute the file then only administrator notice this error. There are my factors involved that can corrupt the file system such as undeclared shutdown, bug in file system or corruption in device drivers can tear down meta-data integrity.

Dealing with this harboring bug and building a well-organized checker for storage systems involves a new approach that works for the file-system checker as more than a postscript. Thus, we propose the Sext3 file system, a modified version of Ext3 which sets harboring bug free handling of inconsistency as a standard plan objective, providing direct system support for the file-system checker in its implementation. Our measurements show that Sext3 has the ability to stop harboring bug before copying itself in system, scans the file system significantly accurate than Ext3, nearing the sequential peak of disk bandwidth. Moreover, its ability is robust to file system aging, making it possible to estimate the error removal and recovery of

system beforehand and thus helping the system administrator make better decisions about running the checker. We also hope that, surprisingly, Sext3 can improve ordinary I/O performance in some cases.

2. Related Work

As all we know that without any file system machine cannot work in papers researchers explained the architecture and design of a new file system named as XFSand it is designed for Silicon Graphics' IRIX operating system. It is an all-purpose file system that can be used on workstations and servers. The center theme of the paper is on the methods used by XFS to scale capability and presentation for the support very large file systems. So that large file system support having mechanisms for managing large files, large numbers of files, large directories, and very high performance of I/O [2]. Many other researchers also explained the theme of work-in-progress to plan and employ a transactional metadata journal for the Linux ext2fs file system. They reviewed the crisis of recovering file systems after the file crash, and explain a plan intended to augment ext2fs's speed and dependability of crash revival by adding a transactional journal to the file system [3].

EXPLODE is a system that creates it simple to methodically make sure real storage systems for errors. This system takes user-written, potentially system-specific checkers and uses these values to drive a storage scheme into difficult corner cases that include crash recovery errors. EXPLODE uses a novel version of ideas from model checking, a complete, long-lasting formal verification method, that creates its checking more systematic and more effective than a pure testing approach while being just as lightweight [10]. Corner-case model checking system is to find serious errors in file systems. Model checking is a prescribed verification method tuned for finding corner-case errors by expansively exploring the state spaces defined by a system. File systems have two self-motivations that make them attractive for such an approach. So if their errors are some of the most serious, since they can obliterate continual data and lead to unrecoverable corruption [11]. To compare and contrast the plan attitude and performance of two computer system families one is the IBM S/360 and it has evolution to the present zSeries line, and the HP (old name as Tandem) NonStop1 Server. Both of the systems have a great and long history. The obligation for the original S/360 line was for very high ease of use; the obligation for the nonStop platform was for single fault tolerance against unplanned outages. There were and still are many resemblances in the design attitudes of the two lines, that including the use of superfluous components and wide error checking. The primary dissimilarity is that the S/360 zSeries center of focus has been on localized retry and restore to keep processors performances as long as possible, on the other hand NonStop developers have supported systems on a loosely coupled multiprocessor blueprint that maintains a "fail-fast" attitude that implemented from side to side a mixture of hardware and software, with workload being actively taken over by another resource when one be unsuccessful [12]. Another study done by researcher shows effects of disk and memory corruption on file system data integrity. Their analysis focuses on Sun's ZFS, a contemporary commercial contribution with many dependability instruments. From side to side cautious and methodical fault inoculation they showed that ZFS is strong to a wide range of disk faults. They further showed that ZFS is less elastic to memory corruption that can leads to corrupt data being came again to applications or system crashes. Their analysis exposes the significance of allowing for both memory and disk in the building of truly robust and strong file and storage systems [15]. By rising performance of CPUs and memories will be wasted if not coordinated by a similar routine boost in HO. As the ability of Single big Expensive Disk has full-grown rapidly, the routine upgrading of system has been self-effacing. Redundant Arrays of Inexpensive Disks (RAID), that is based on the magnetic disk technology developed and promoted for personal computers and that offered an attractive alternative to SLED, that showed potential improvements of an arrange of scale in performance, dependability, power use, and scalability. In this research, researchers introduced five levels of RAIDs, and gave their relative cost, performance and compares RAID to an IBM 3380 and a Fujitsu Super Eagle [16]. The paper [18] showed methods that mechanically take out such checking sequence from the source code it, before the programmer, thus keep away need for a priori information of system rules. The main beliefs are facts implied by code dereference of a pointer, p, and implies a belief that p is non-null, a callto "tmlock (1)" entail that 1 was locked and so on.

3. Extended Motivation

Before we implicate a good and robust file system checker, we have to see back what type of approaches are in practice in the system. First we discuss the file system overall check and its repair function with its abilities on the system that is widely used as an open source known as EXT3 file system.

Ext3 and Ext4 are the default Linux file system. Ext4 is the new version of Ext3 and adds some field in Ext3 mechanism, but the basic structures are the same. The metadata is accumulated throughout the file system, and

the metadata which is linked with a file are stored close to it. The entire area is divided into numerous block groups, and block groups holds several blocks. A block group is used to store file metadata and file content.

3.1 Super Block

The super block in Ext3has 1024 bytes and it is located in the start of boot of file system. It can store first two sectors of the boot code if necessary to store. Typically backup copies are stored in the first file of the block of every block group. It has the most basic information about file system, just like block size, number of block bitmap and inode bitmap for the block group.

3.2 Block Group Descriptor Table

It includes a collection descriptor data structure for each block group. The group descriptor stores the address of block bitmap and inode bitmap for the block group.

3.3 Data Bitmaps

The data bitmap is also recognized as block bitmap. This data/block bitmap directs the share status of the blocks in the group. The inode bitmap controls the allotment status of the inodes in the group.

3.4 Inode Tables

Inode table includes the inodes that illustrates the files in the group.

3.5 Inodes

Each inode matches to one file and it stores file's main metadata, for example file's size, rights, and temporal information. It size is typically 128 bytes and it is allocated to each file and directory.

3.6 Inode Allocation

If a fresh inode is for a directory, Ext3 attempts to put it in a collection that has not been used much. Using some amount of free inodes and blocks in the superblock, Ext3 computes the standard free inodes and blocks per group. Ext3 searches every of the group and uses the primary one whose free inodes and blocks are fewer than the standard. The default size of blocks is 4KB, and the size is known in the superblock. When a block s owed to an inode, Ext3 will try to assign in the same group as the inode and use first available plan.

3.7 Indexing and Directories

An Ext3 is regular file except it has a special type value. The content that located to the directories is a list of directory entry data structure, which explains file name and inode address. The directory and its length entry vary from 1 to 255 bytes. There are two fields in the directory entry

- Name length the length of the file name.
- Record length the length of this directory entry Directory entry allocation.

4. Design and Implementation of Sext3

In this section we will discuss the design and implementation of our proposed model Sext3 (secure Ext3). While our system of Sext3 is based on Ext3, Sext3 improves meta-data's density for checking meta-data before writing it on the drive. In other words we can say that this Sext3 has the ability to remove the chance of occurrence of harboring bug in meta-data of data bitmap field in Sext3. The Harboring Bug in data bitmap causes silent Metadata corruption leads Ext3 to miss indirect block and revert the behavior of FSCK to traditional error checking utility. Sext3 significantly reduce the harboring bug in Sext3 and insure harboring bug free system to data bitmap filed writing of meta-data.

4.1 Goals

We expect Sext3 to meet the following criteria

4.1.1 Backups

The Sext3 will make the backup of last state of the data bitmap filed. Through this backup system it can easily roll back the erroneous meta-data. So this ability of scan and repair of meta-data through backup should be supreme concern for file designer because no one wants erroneous meta-data when he needs it.

4.1.2 Ability of communication

It has the additional ability of commutation. In other word we can say that it is the indirect communication bridge between system and data bitmap filed. No data can be written on to the data bitmap field until unless system sends permanents list to data bitmap checker. It will send report if it found any mismatch between the parameter list and the data of update.

4.1.3 Real time checking for harboring bug

Thanks to its real time checking ability of meta-data and getting permanents list before updating the field of data bitmap it should remove the chance of occurring of harboring bug in meta-data while system is writing it in the field of data bitmap of Sext3. If it found any harboring

bug in update it will immediately block and roll back the update and resume the last state of data bitmap and through the ability of commutation it will communicate the presence of harboring bug in its update.

4.2 Competitive file system performance

Repairability can come at the expense of responsiveness and through put, as these are not critical in environment of scientific research. As we know that security never comes for free, this is the tradeoff between repairability and responsiveness and through put.

4.3 Sext3 File System

Sext3 is developed atop Ext3, the upcoming default file system for many popular Linux distributions. Sext3 inherits most of the mechanisms used in Ext3, except one the data bitmap checker block structure. This section details these new features and gives a basic overview of our implementation.

4.4 Sext3 layout

To reduce the chance of Harboring bug in phase one of file update, Sext3 introduced a new filed that is known as data bitmap checker. Through this checker the system cannot update the system directly. As we early mention the system must send critical parameters to data bitmap checker and by the help of this parameters this data bitmap checker checks the upcoming update in real time scenario.
The difference between Sext3 and Ext3 is the removal of harboring bug from system at the time of update in other words we can say Ext3 has the ability to secure the data bitmap field in real time update coming from system.

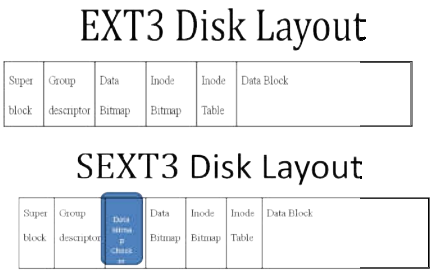

Figure 1 Ext3 and Sext3 Disk Layout Comparison.

5. Methodology of Data Bitmap Checker

Suppose system wants to update Metadata to the data bitmap. In normal Ext3, system can access directly the data bitmap field. But in our proposed solution the data cannot be updated by direct access of data bitmap field.

First it must send some parameters to data bitmap checker. Parameters include all checksums, date and update mode.
When data bitmap checker gets all information about update process from system and parameters, than it will allow system to update the data bitmap field in Ext3. When system is updating Metadata in the field of data bitmap, in this mean time the data bitmap checker will monitor the updating parameters whether this update contain any harboring bug.

If data bitmap checker found any harboring bug in the process of update, immediately data bitmap checker will stop the process.

It will do two things

- First roll back every update done by system.
- Second it will alert system.

In this alert the data bitmap checker will give two options; in first option data bitmap checker will ask to system to update parameters and resend if system wants to make no change in upcoming update. In second option data bitmap checker will ask system to remove harboring bug from upcoming update and make changes as per parameter sent in last communication.

We can take examples of system update without and with bitmap checker in Ext3 file system of Linux operating system. Suppose system is updating the field of data bitmap in Ext3 file system and there is nothing that can check the harboring bug. If system sending data that contain harboring bug, normal checksums cannot check the data for this type of bug. This bug can only be noticed when system needs to execute file for any critical work.
So our proposed solution has the ability to prevent this bug before get penetrate in system file. Suppose we have three things

- **System** it is the source of data.
- **Data bitmap** this is the field of Sext3 that will get the update from system.
- **Data bitmap checker** it is the field of Sext3 that has the ability to check the upcoming data for harboring bug.

In first stage, system will send parameters to data bitmap checker and after this sending the system will send data towards data bitmap field. In this process of updating data bitmap field the data bitmap checker continually monitor for any mismatch between data and parameters. If no mismatch found the data bitmap allow system to finalize the update. And also update date itself for backup.
There is another condition if system sends parameters to checker and data to data bitmap. In the process of update if data bitmap checker found any harboring bug in data or in parameters it will immediately stop that updating process

and alert system about it. In this alert checker will say to modify parameters as per data or changer data as per parameters. When system is taking any action on the checker alert, in this mean the data bitmap checker roll back all the update occurred.

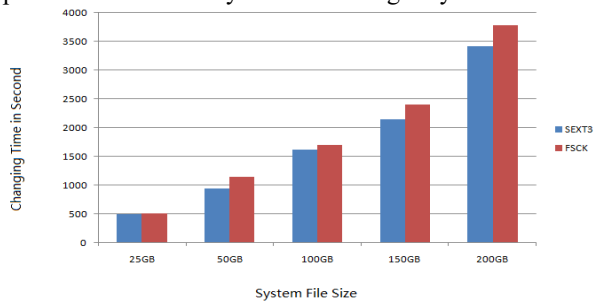

Figure 2 This graph shows sext3 and fsck execution time for different size of file systems

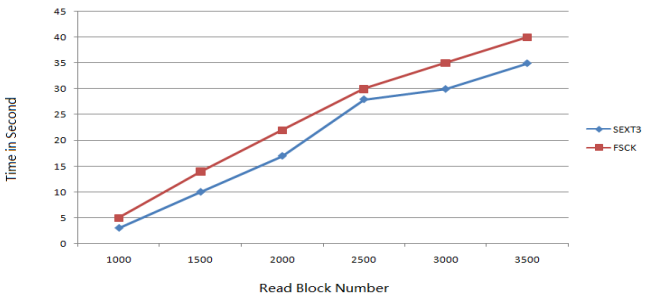

Figure 3 This graph shows the time to access each inode in block while scanning

6. Sext3 performance Analysis

First we observed how Sext3 and fsck executes as the file system increases in size. So we initialize the file system image through creating single directory on one block group, each of which holds a number of files with sizes chosen consistently from 1 to 512 block (4 KB 2 MB); and then we creates files until it contains 25.6 MB or 20% of the block group size. To boost the size of the file system to the preferred amount, then randomly create new files (4 KB 2 MB) or add one to 256 blocks of data to existing files. We show our results in Figure 3.

To verify check controls the scan time, Figure 3 further defined by the total time and by the amount spent in each phase. During this phase, fsck and Sext3 scans all inodes and their corresponding indirect blocks, which include the largest portion of the file-system's metadata. In addition, since fsck has to execute this scan again if it detects multiply-claimed blocks created by harboring bug, the actual total time may be even longer in the presence of errors.

To better recognize the I/O performance during this phase, we calculate the time fsck spends reading each character block during the 150 GB experiment. We show our results in Figure 3, which displays the collective time spent reading indirect and inode blocks. Accesses of indirect blocks overpoweringly dominate I/O time.

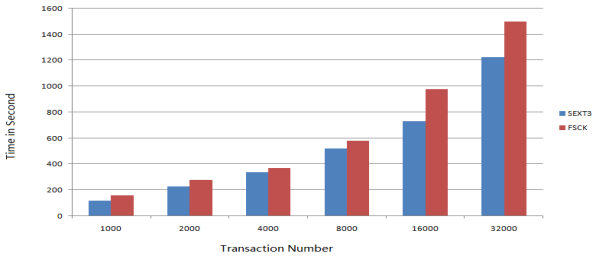

Figure 4 This graph shows postmark performance of the system

Finally, we use Postmark to compare Sext3 and fsck. We use the default settings for Filebench and invoke Postmark with 32000 files between 4 KB and 4 MB in size. A figure 4 shows our results. In most cases, Sext3 performs nearly identically to ext3. In this case, Sext3 performs 5% better than ext3. Given these performance measurements, we can conclude that Sext3 performs competitively with fsck in most cases, exceeding Sext3 in its ability to handle random checking of harboring bug, and performing slightly worse in terms of memory cost.

6. Limitation

It is still prototype and has some limitations as it is very simple. It has new field so it must take some space on disk. Our proposed solution works on physical data rather than logical data. It has two abilities first is backup and second is rollback so these abilities also consume physical space. It also put some delay in updating process from system to data bitmap field. In conclusion, memory consumption is the potential drawback of our checker. In future work, these can likely be addressed by using more sophisticated checksum for each file. In figure 5 we can observe that little bit more memory is consumed by the Sext3 system as compared with fsck file checker.

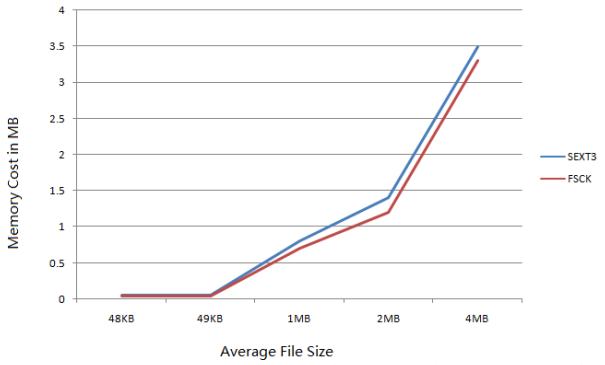

Figure 5 This graph shows cost of memory in system per file in different sizes

7. Conclusion

While the file system checker in Ext3 is the ultimate solution to check file system for errors in the system. But most of the time it fails to checks the harboring bug in the file system just like data bitmap field in Ext3 file system. Our proposed file system is powerful enough to check and remove this type of silent killer of data bitmap in Ext3 but it is still academic model needs sufficient testing in lab before implemented in real environment. It has the ability to remove the bug in real time scanning system. We believe that it has the ability to remove the harboring bug before bug destroys the entire file system of data bitmap on execution. Why we wait until the execution of the data we have to take measure against this bug before it create unchallengeable progress. In other words we can say that this data bitmap checker is proactive in nature that will not allow this bug to take place on disk.

References

[1]H.Reiser. ReiserFS. www.namesys.com, 2004.
[2]Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS File System. In Proceedings of the USENIX Annual Technical Conference (USENIX '96), San Diego,California, January 1996.
[3]S. C. Tweedie. Journaling the Linux ext2fs File System. In The Fourth Annual Linux Expo, Durham, North Carolina, May 1998.
[4]D. Hitz, J. Lau, and M. Malcolm. File System Design for an NFS File Server Appliance. In Proceedings of the USENIX Winter Technical Conference (USENIX Winter '94), San Francisco, California, January 1994.
[5]M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. ACM Transactions on Computer Systems, 10(1) 26–52, February 1992.
[6]Sun Microsystems. ZFS The last word in file systems. www.sun. com/2004-0914/feature/, 2006.
[7]Wikipedia. Btrfs. en.wikipedia.org/wiki/Btrfs, 2009.
[8]R. Ganger and Y. N. Patt. Metadata Update Performance in File Systems. In Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI '94), pages 49–60, Monterey, California, November 1994.
[9]D. Engler and M. Musuvathi. Static Analysis versus Software Model Checking for Bug Finding. In 5th International Conference Verification, Model Checking and Abstract Interpretation (VMCAI '04), Venice, Italy, January 2004.
[10]J. Yang, C. Sar, and D. Engler. EXPLODE A Lightweight, General System for Finding Serious Storage System Errors. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06), Seattle, Washington, November 2006.
[11]Yang, P. Twohey, D. Engler, and M. Musuvathi. Using Model Checking to Find Serious File System Errors. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04), San Francisco, California, December 2004.
[12]W. Bartlett and L. Spainhower. Commercial Fault Tolerance A Tale of Two Systems. IEEE Transactions on Dependable and Secure Computing, 1(1) 87–96, January 2004.
[13]Bonwick and B.Moore. ZFS The Last Word in File Systems. http //opensolaris.org/os/ community/zfs/docs/zfs last.pdf.
[14]C. A. Stein, J. H. Howard, and M. I. Seltzer. Unifying File System Protection. In Proceedings of the USENIX Annual Technical Conference (USENIX '01), Boston, Massachusetts, June 2001.
[15] Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, End-to-end Data Integrity for File Systems A ZFS Case Study Proceedings of the 8th Conference on File and Storage Technologies (FAST '10), San Jose, CA, February 2010.
[16]D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In Proceedings of the 1988 ACM SIGMOD Conference on the Management of Data (SIGMOD '88), pages 109–116, Chicago, Illinois, June 1988.
[17]K. Keeton and J. Wilkes. Automating data dependability. In Proceedings of the 10th ACM-SIGOPS European Workshop, pages 93–100, Saint-Emilion, France, September 2002.
[18]D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as Deviant Behavior A General Approach to Inferring Errors in Systems Code. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), pages 57–72, Banff, Canada, October 2001.
[19]M. M. Swift, B. N. Bershad, and H. M. Levy. Improving the Reliability of Commodity Operating Systems. In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, New York, October 2003.
[20]D. Anderson, J. Dykes, and E. Riedel. More Than an Interface SCSI vs. ATA. In Proceedings of the 2nd USENIX Symposium on File and Storage Technologies (FAST '03), San Francisco, California, April 2003.
[21]L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An Analysis of Latent Sector Errors in Disk Drives. In Proceedings of the 2007 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '07), San Diego, California, June 2007.
[22]L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C Arpaci-Dusseau, and R. H. Arpaci-Dusseau. An Analysis of Data Corruption in the Storage Stack. In Proceedings of the

6th USENIX Symposium on File and Storage Technologies (FAST '08), pages 223–238, San Jose, California, February 2008.

Raza Muhammad: He completed his Bachelor Degree BS(CS) for Shah Abdul Latif University Kharipur Pakistan in computer science in 2004 and move to Karachi for higher studies. In 2007 he completed his Master Degree MS(CS) in computer science from Pakistan Air Force-Karachi Institute of Economics and Technology Karachi Pakistan. Now he is pursuing his PhD in computer science from Huazhong University of Science and Technology Wuhan P.R. China. In his academic career he worked as lecturer and visiting lecturer in various universities and government institutes. He also contributed in research papers as main author and as second author. His area of interests are Big Data, Linux Kernel programming, Wireless network and security System of Computation.

Zhou Ke: He completed Bachelor in Computer Peripheral from HUST Wuhan China in 1992 and Master in Computer Architecture HUST Wuhan China in 1996 and perused his Doctor in Computer Architecture HUST Wuhan China and completed in 2000. Nowadays he is serving HUST as Professor in Wuhan National Laboratory for Optoelectronics (WNLO). He also served as Visiting Scholar Cranfield University U.K. 2005-03-31-2006-03-31. He got honor and award in A Heterogeneous Unified Storage System for GIS Grid Super Computing 2006 storage challenge finalist Award,2006.

Basheer Riskhan: He received primary and secondary education at St. John's College, Jaffna and Zahira National College, Puttalam. He earned his Bachelor degree in Computer Science from Bharathidasan University, India in 2002 and Master degree in Education from National Institute of Education, Sri Lanka in 2012. At present he completed his PhD in Computer Science from Huazhong University of Science and Technology Wuhan, P.R. China. Basheer Riskhan joined National College of Education in Sri Lanka as a Lecture in 2005 and working as a Sri Lanka Teacher Educator Service (SLTES) officer in Sri Lanka. Before that he served as Software Engineer in several private companies. His research interests are in the field of Virtualization, Cloud Computing, Kernel Programming and Big Data.