International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 02 Issue: 09 | Dec-2015 www.irjet.net p-ISSN: 2395-0072

A secure and flexible storage system using multiple servers for combining data in IoT

A.sravanthi[1], P.Vijayalaxmi[2]

[1] *Asst professor ,CSE , Gokaraju Rangaraju Institute of Engineering and Technology, Telangana, India*
[2] *Asst professor, CSE , Gokaraju Rangaraju Institute of Engineering and Technology, Telangana, India*

Abstract – Now a days, there is fast development of silicon chips and networking technologies, computers, devices and networking have become highly cost, incurring the introduction, develop-ment and deployment of the Internet of Things (IoT). The tiny identifying devices and wearables in IoT have transformed daily life in human society, as they generate, process and store the amount of data increasing at exponential rate all over the world. Due to great uasge on data mining and analytics activ-ities in IoT, secure and scalable mass storage systems are highly demanded for aggregate data in efficient processing. In this paper, we propose such a secure and scalable IoT storage system based on revised se-cret sharing scheme with support of scalability, flexibility and reliability at both data and system levels. Shamir's secret sharing scheme is applied to achieve data security without complex key management associated with traditional cryptographic algorithms. The original secret sharing scheme is revised to uti-lize all coefficients in polynomials for larger data capacity at data level. Flexible data insert and delete operations are supported. Moreover, a distributed IoT storage infrastructure is deployed to provide scala-bility and reliability at system level. Multiple IoT storage servers are aggregated for large storage capacity whereas individual servers can join and leave freely for flexibility at system level. Experimental results have demonstrated the feasibility and benefits of the proposed system as well as tangible performance gains.

Key Words: Security,Scalability,Reliability,Storage,Big Data,IoT

1. Introduction

In current computer world, the development of multicore pro-cessors has led to increase in the amount of computational power available on a single die. In parallel, the evolution of networking technologies has contributed with the fast development of web technologies and data centers.

Big Data" is a name for large and growing sets of data. "How fast it is growing" and "how complicated it is" are the major con-cerns. Enterprises know that the data in their storage systems is an excellent source of insights [1]. Due to a relatively recent phe-nomenon of a fast increase in data scattered across computing nodes in distributed systems or devices in the Internet of Things (IoT), mass storage systems are also often required for aggregated Big Data [2]. In addition, the computational power of individual IoT devices and wearables such as sensors and RFID tags as well as their connecting network bandwidth is insufficient for distributed data mining and analytics. In order to take advantage of IoT servers' computing capacity, widely spread data should be collected and fused in distributed storage servers. Then, closely connected IoT

Eg 2: It is reported that X increase with Y [45].

1.1 Sub Heading 1

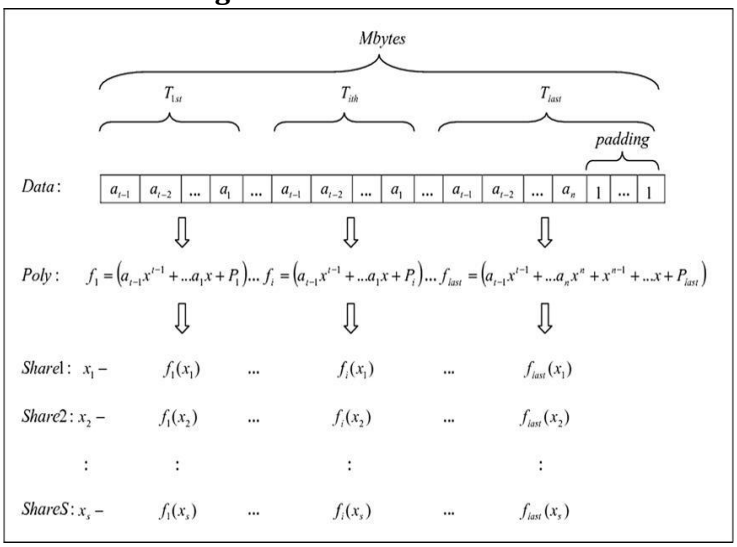


Fig. 1. Share generation procedure

2. Multi-coefficient secret sharing scheme with internal padding

Shamir's secret sharing scheme is redesigned to contain more data with support of internal padding. Data can be transformed into shares and distributed amongst multiple participants. In the Data Restoration phase, the original data could be reconstructed from certain number of shares.

2.1. Share generation

Initially, all the original data are divided into blocks according to a default threshold or a number provided by users. Assume there are M bytes in data, and the threshold is T . Then, each block will contain $(T - 1)$ bytes. If there are fewer than $(T - 1)$ bytes in a block, a padding string (consisted with all 1's) will be appended to make sure the

block size is $T - 1$ bytes.

For each $(T - 1)$-byte block, a $(T - 1)$th degree polynomial $f(x)$ is built. Each of the last $(T - 1)$ coefficients is assigned one byte data item from the block. The first coefficient a_0 is used to indi-cate the padding size for the corresponding polynomial. The value of a_0 should be the block's padding size plus one, because a_0 it-self occupies one byte in this block. Since only one byte is used for padding size, the maximum size of padding is 256. This indicates that the limit of threshold T is also 256. Such a design is sufficient for secret sharing since normally polynomial degree cannot reach this high. Otherwise, the computation overhead will be significant. To generate shares, we assign a unique value x to each participant or device server. In this paper, the value of x is a natural number starting from 1. The actual share for each participant consists of the unique value x and the corresponding polynomial values for all data blocks. The general procedure is illustrated in Fig. 1.

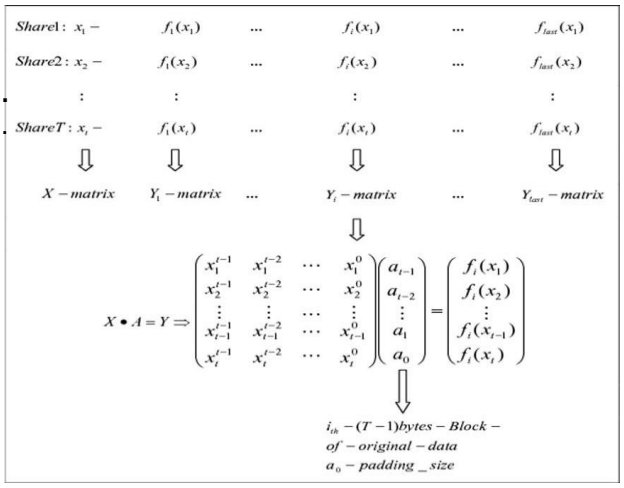


Fig. 2. Data retrieval procedure.

2.2. Data retrieval

Shares collected from multiple devices are used to reconstruct the original data. The general steps of Data Restoration are shown in Fig. 2.

With T shares, T equations can be created and aligned to form a multiplication of two matrices, the coefficient matrix A and the value of x matrix X. The result matrix Y is the matrix of $f(x)$ val-ues, i.e. $X \times A = Y$. The matrix X is constructed by powers of x as shown in Fig. 2. This is the reason that Shamir's secret sharing [5] is selected over Blakley one [7] where matrix X consists of arbitrary values (not powers of x). Therefore, the share size in Shamir's ver-sion is much smaller than the one in Blakley's. Matrix Y is built by the $f(x)$ values extracted from the corresponding bytes in shares. For example, the first matrix Y is

$(f_1(x_1), f_1(x_2), f_1(x_3), \ldots, f_1(x_{t-1}), f_1(x_t))^T$.

Then, Gauss–Jordan-Elimination (GJE) is applied to calculate the coefficient matrix A in Fig. 2.

After matrix A has been calculated, the corresponding data block is retrieved by extracting coefficients one after another. The coefficient a_0 is the padding size to indicate how many bytes in this vector should not be treated as padding stuffs. For every matrix Y, such a matrix A should be calculated for the data in that block. Eventually, all data will be recovered.

Since large numbers can cause accuracy problem, finite field GF(2^8) [8] is chosen to represent numbers (in 8 bits each). Addition and multiplication can be handled easily and properly. Division can be split into two steps: calculation of the divisor's multiplicative in-verse and then the multiplication. Traditionally, extended Euclid's algorithm is used for multiplicative inverses. Since there are only 256 possible values for coefficients and $f(x)$, their multiplicative inverse values can be calculated in advance and saved in a table. This pre-calculated table approach can achieve significant perfor-mance gains.

3. Post-generation data operations

Once data shares are generated, the corresponding contents of the original data are not exposed anymore. Since shares change the representation of the original data, encryption effect is achieved. After sharing generation, data operations such as data insertion and deletion might still need to manipulate partial or whole data. If the whole data needs to be modified, going through data restora-tion and share re-generation processes is reasonable. But if only partial data need to be modified, re-generating all shares is highly inefficient. Revised Shamir's secret sharing scheme with internal padding reduces the data management granularity to block level. The threshold or block size defines the actual data modification units since all coefficients in polynomials are used to hold data and have to be dealt together. In this section, insert operation is taken as an example. Other operations are similar.

3.1. Positioning

For efficient data operations, the precise location and size of affected data are critical. These can be calculated based on the padding sizes in all blocks. To speed up this process further, one metadata, called Padding Record (PR), is maintained for all blocks' padding sizes. PR only maintains information for those blocks con-taining significant padding, i.e., the padding size is greater than one. With PR, it is not necessary to restore blocks in sequence to fig-ure out all previous padding sizes and the exact position for mod-ification can be calculated quickly. Only related blocks and their corresponding polynomials are adjusted.

For a particular file, according to its PR, threshold (block size) and data offset (input), the first block and its corresponding bytes in shares can be detected. Initially, set $data_{pos}$ = 0. Then the PR is scanned sequentially and block data sizes are continuously added. If a block is not in PR, its default *paddingSize* will be one. Otherwise,

3.2. Insert operation

To insert a new data item into a file, it is not necessary to regen-erate all data shares. Only those bytes related to the affected block in shares will be changed.

First of all, the positioning process with PR has to be applied to detect the affected block and exact byte position. The pin-pointed block might contain padding if it has been touched by post-generation operations before. If this is the case and the padding size is larger enough for the newly inserted data, only the polynomial values for this block will be regenerated and re-distributed. PR will be updated accordingly. If the original block does not contain any padding or the existing padding space is not enough for the new data, extra blocks are required. Some data bytes will be inserted into the current block to fill up the padding size whereas others will be used to build up new blocks. The original next block will not be touched, but pushed backwards since new blocks will be inserted before that.

As a library call, the output of this function contains the share numbers for updating, the affected block number, a breaking flag to indicate if new blocks are added, and the new blocks' values for related share bytes. The output format is as: ShareNum − Af-fectedBlock + BreakFlag + NewBlocks'Values. PR will be updated as well. entries where blocks 2 and 14 contain significant padding bytes. The output indicates block 7 will be changed and five more blocks will be added. Affected and newly generated bytes for the associ-ated shares are provided. PR maintains two

4. Infrastructure of the secure and scalable IoT storage system

In the proposed secure and scalable IoT storage system, there are mainly four components: client, dispatcher, peer managers and regular peers. The client is the original file holder; the dispatcher maintains the references (IP addresses) of peer managers who are in charge of peer groups; regular peers (or storage devices) are aligned up in columns to form peer groups.

Operations in the proposed storage system can be categorized into three layers:

- File saving and restoration: Files are transformed into shares based on the scaled secret sharing scheme. Since shares look to-tally different from original data blocks, security protection for data in-transit and at-rest is provided. Shares can be converted back to the data in the original files.
- Connection setup and data transfer: Connections are estab-lished between the client and the dispatcher for peer managers' IP address search, between the client and peer managers for share distribution and gathering, and between peers for share replication.
- Share replication: Peer mangers determine which peers in the groups are selected for share replication to achieve high avail-ability. Regular peers can leave and join their groups on the fly. It is possible that peer managers can exchange their peer nodes as

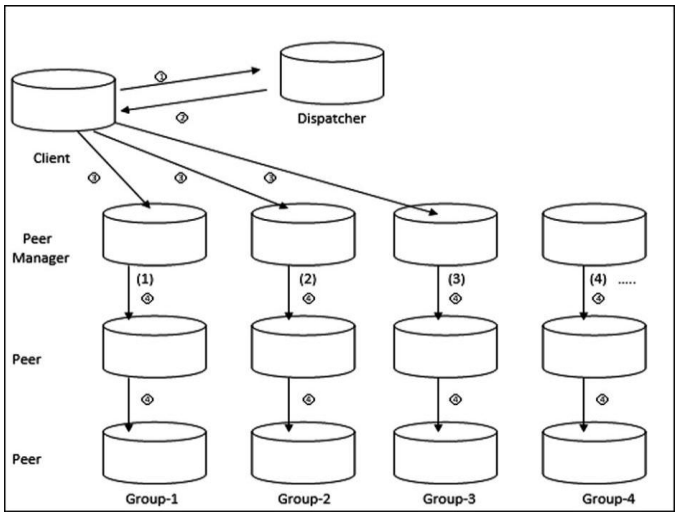


Fig. 4. Saving a file into the distributed storage system.

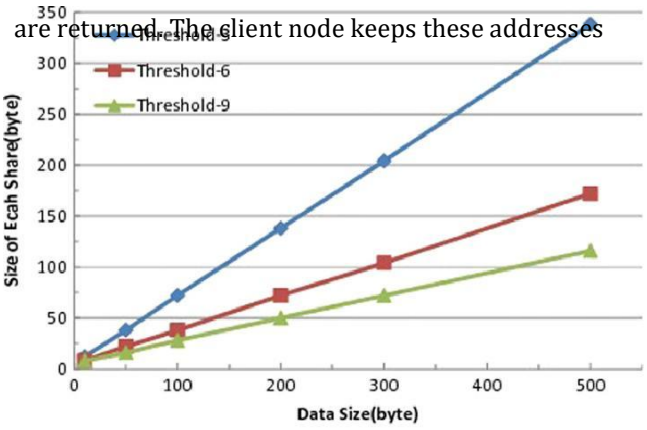
groups expand and shrink. For simplicity, device peers are parti-tioned into groups shown as columns in Fig. 4. Within each peer groups, star topology is adopted for internal network where the manager stays in the center and connects with each peer di-rectly.

4.1. File saving

To save a file into peers, the storage system goes through three steps: share generation, share transfer and share replication .

After a file is submitted, the client node converts the file into shares using the scaled secret sharing scheme. Then the client node contacts the dispatcher about where to save those shares. In this file management system, the dispatcher is the only centralized node. However, it only maintains the IP addresses of peer managers and will not involve in file operations. The dispatcher could be a bottleneck for server requests although it is not the bottleneck for data operations. Therefore, it can be implemented in a cluster man-ner to balance service requests among multiple server nodes. The dispatcher serves as a proxy and selects peer groups/managers ac-cording to the work load situation or in round robin manner. Based on the client's request, a number of peer groups are selected and their peer managers' IP addresses

are returned. The client node keeps these addresses

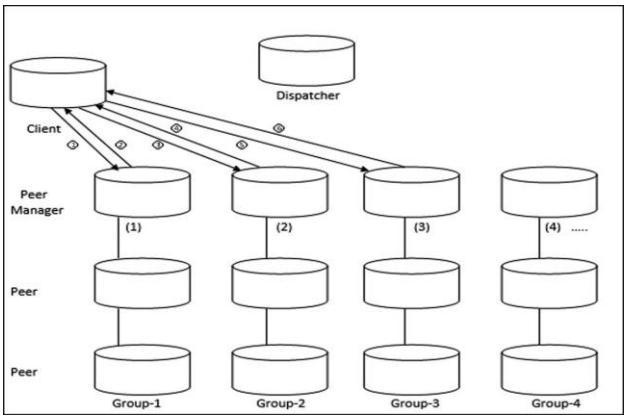


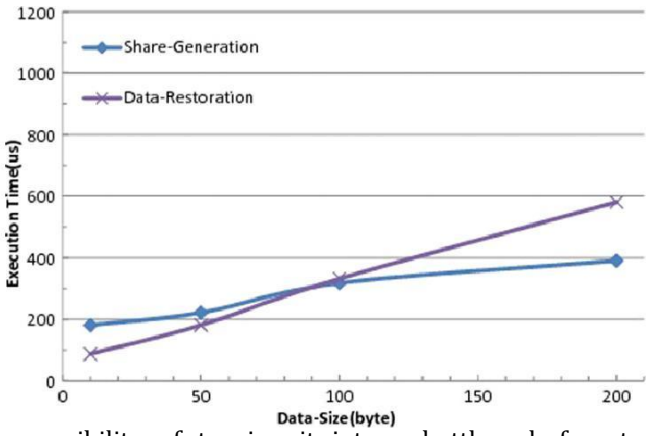
locally for the future use. Then shares are distributed to corresponding peer managers. Since the information in shares has been transformed in share format, data in transit pro-tection is provided.

Once peer managers receive their shares, replication phase starts for high availability. Transformed shares provide data at rest protection. Distributed RAID can be deployed for error detection and correction.

4.2. File retrieval

To retrieve a file, the client node does not need to contact the dispatcher with the locally cached addresses. This reduces the





possibility of turning it into a bottleneck for storage operations. File retrieval only contains two phases: sharing collection and file restoration as shown in Fig. 5.

With locally maintained manager lists, the client node can contact related peer managers directly. The latter ones can collect several replicas for error detection and correction. Then, the correct shares are transferred back to the client node. If a peer manager crashes and another manager is newly elected, only the dispatcher will be notified. Then, the peer group will be unavailable to the client node although the share replicas are still there. In such cases, the client node needs to contact the dispatcher again for the new group's (peer manager's) IP address. Only when all peers in a group crash, could the shares be permanently lost.

Once shares are collected back to the client node, the scaled secret sharing scheme will be used again to restore data blocks and assemble files. Even if some shares (peer groups) have been lost, enough other different shares are still able to recover the original data blocks, and then files.

5. Performance analyses and experimental results

Files can be transformed into shares and distributed to multiple peers in the IoT storage system. If the threshold or block size is fixed, same shares will be generated. Also, the original files can be recovered effectively.

Experimental files with small sizes are selected to demonstrate the performance trend of our distributed storage system. Such files can provide details as larger files follow the same trend. All tests are conducted on a machine with a 2.40 GHz Intel Pentium 4 CPU, 489.0 MB memory and 2.6.28-17-generic Linux kernel.

5.1. Storage cost

Assume there are M bytes in the original file. The threshold number is T, and share number is $S(S \geq T)$. Then each block has

mapped onto a polynomial with the highest degree $T - 1$. For each block, S one-byte polynomial values are treated as shares. Also, each share should include the unique value x. Therefore, the total byte number is $T^{M}_{-1} + 1$. As we know, each byte is represented by two hexadecimal digits. The output of each share will contain $(\lceil T^{M}_{-1} \rceil + 1) \times 2$ hexadecimal digits. For total S shares, there will be $(\lceil T^{M}_{-1} \rceil + 1) \times 2 \times S$ hexadecimal digits. Roughly speaking, to generate shares, a file is cut down by $T - 1$ for S times ($S \geq T$). The total extra bytes could be: $Overhead = \frac{S-1}{T-1}$ if $T - 1$ divides M. If only T shares are generated, the extra storage overhead will be T^{M}_{-1}. The overall data size is not increased much.

The relationship between single share's size and threshold is illustrated in Fig. 6. As threshold or block size increases, share size decreases.

5.2. Share generation and data restoration

In our experiments, files are transformed into shares. Also, the original data blocks can be restored from these shares. When thresholds are fixed, the overall overheads of share generation and data restoration are shown in Fig. 7. Obviously, the execution time of share generation increases slightly as the data size increases and is less than the one for data restoration. Overall, data restoration is slower than share generation for large files.

Share generation process consists of polynomial formation and $f(x)$ calculation. For larger thresholds, the degree of the polyno-mial (one less than the threshold) will be higher and there will be more coefficients in each polynomial. In the above test, each poly-nomial takes almost the same time to get $f(x)$ because the thresh-olds are the same. Because larger files need more data blocks, the corresponding polynomial calculation will be more costly. Then, the share generation overhead will increase.

On the other side, if the threshold increases, each block will con-tain more bytes of original data. For fixed size original data, there will be fewer data blocks. For each block, the larger the threshold is, the less time the single share generation will take since the poly-nomial can digest more data and fewer polynomial calculations are required. Even though it may take a little longer time to calculate effectiveness and correctness. The future work includes achieving high avail-ability of the dispatcher through redundant servers and adopting scalable network topologies (tree or P2P) for peer groups.

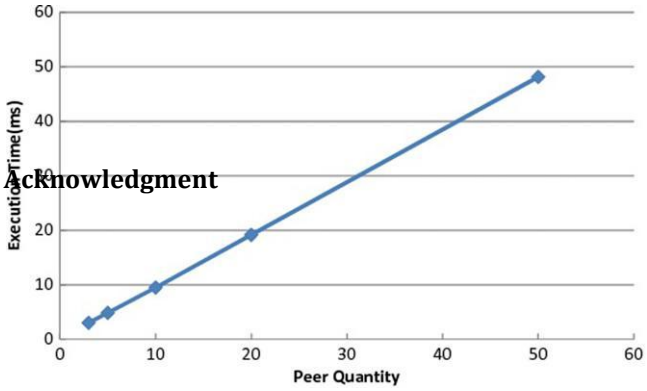
6. Related work

Shamir [5] and Blakley [7] brought up the concept of secret sharing independently in 1970s. The (t, n) threshold scheme con-cept was introduced. Since then, secret sharing draws more and more attention.

A number of designs applied in storage systems may generate computational overheads that induce potential bottlenecks in to-day's systems, which increasingly employ multi-Gbps links and/or are built composed of high throughput storage devices [9]. Hard-ware failure is one of major concerns. Many systems have consid-ered the fault tolerance issue and proposed some solutions, such as the ones in distributed RAID [10] and OceanStore [11]. They

7. Conclusions

This paper proposed a secure and scalable IoT storage sys-tem with consideration of security, scalability, flexibility and re-liability to meet the requirements of computing, communication and storage capabilities for data mining and analytics applications with large aggregate data in IoT. A refined Shamir's secret shar-ing scheme is developed for security and scalability at data level. Multi-coefficient strategy is deployed for improved data capac-ity. Files are divided into blocks which are then transformed into shares. A distributed storage infrastructure is proposed to spread data shares across multiple storage peer groups and replicate them among peer nodes within each group. This infrastructure can pro-vide scalability, flexibility and reliability at system level. Perfor-mance analyses and experiments demonstrate its



Acknowledgment

This research was supported in part by the US National Science Foundation under Grant No. 0959124.

References

[1] 50 Top Open Source Tools for Big Data, http://www.datamation.com/data- center/50-top-open-source-tools-for-big-data-1.html (Last Accessed on November 6, 2014).

[2] C. Aggarwal, N. Ashish, A. Sheth, The Internet of Things: a survey from the data-centric perspective, in: Managing and Mining Sensor Data, Springer, 2013.

[3] J. Sathish Kumar, Dhiren R. Patel, A survey on

Internet of Things: security and privacy issues, Int. J. Comput. Appl. 90 (11) (2014).

[4] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, Kaladhar Voruganti, POTSHARDS: secure long-term storage without encryption, in: Proceedings of USENIX Annual Technical Conference, 2007.

[5] Adi Shamir, How to share a secret, Commun. ACM (1979).

[6] J.L. Gonzalez, Ricardo Marcelín-Jiménez, Phoenix: fault-tolerant distributed Web storage based on URLs, J. Convergence (2011).

[7] G.R. Blakley, Safeguarding cryptographic keys, in: Proceedings of Inernational Workshop on Managing Requirements Knowledge, 1979, pp. 313–317.

[8] William Stallings, Crytography and Network Security: Principles and Practices, fourth ed., Pearson Education, Inc., 2006.

[9] M. Polte, J. Simsa, G. Gibson, Comparing performance of solid state devices and mechanical disks, in: Proceedings of Workshop on Petascale Data Storage, 2008.

[10] M. Stonebreaker, Gerhard A. Schloss, DistributedRAID—a new multiple copy algorithm, in: Proceedings of the Sixth International Conference on Data Engineering. 1990, pp. 430–437.

[11] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, John Kubiatowicz, Pond: the OceanStore prototype, in: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, 2003, pp. 1–14.

[12] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, The Google file system, in: Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003.

[13] K. Shvachko, Harong Kuang, S. Radia, R. Chansler, The Hadoop distributed file system, in: Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies, 2010.

[14] H.S. Gunawi, N. Agrawal, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, J. Schindler, Deconstructing commodity storage clusters, in: Proceedings of the 32nd Annual International Symposium on Computer Architecture, 2005.

[15] S. Quinlan, S. Dorward, Venti: a new approach to archival storage, in: Proceedings of the 1st USENIX Conference on File and Storage Technologies, 2002, pp. 89–101.

[16] M.O. Rabin, Fingerprinting by Random Polynomials, Center for Research in Computing Technology, Harvard University, 1981.

[17] Qiying Wei, Tingting Qin, Satoshi Fujita, A two-level caching protocol for hierarchical peer-to-peer file sharing systems, J. Convergence (2011).

[18] Su Chen, Ling Bai, Yi Chen, Hai Jiang, Kuan-Ching Li, Deploying scalable and secure secret sharing with GPU many-core architecture, in: Proceedings of the 13th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, Shanghai, China, 2012.

[19] Abdullah Gharaibeh, Samer Al-Kiswany, Sathish Gopalakrishnam, Matei Ripeanu, A GPU accelerated storage system, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, 2010.

[20] Matthew Curry, Weibin Sun, Robert Ricci, GPUstore: harnessing GPU computing for storage systems in the OS kernel, in: Proceedings of the 5th Annual International Systems and Storage, 2012.

[21] Jiwu Shu, Zhirong Shen, Wei Xue, Shield: a stackable secure storage system for file sharing in public storage, J. Parallel Distrib. Comput. 74 (9) (2014).

[22] Alok Kumbhare, Yogesh Simmhan, Viktor Prasanna, Designing a secure storage repository for sharing scientific datasets using public clouds, in: Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds, 2011.

BIOGRAPHIES

Sravanthi.A has received M.Tech degree in 2010 from JNTU university in the field of computer science with distinction. Awarded B.Tech in 2006.I have Published two journals at international levels.

P.Vijayalaxmi has received M.Tech degree in 2009 from JNTU university in the field of computer science .B.Tech in 2005 And GATE qualified in 2007.