Employing GPU Accelerators for Efficient Enforcement of Data Integrity in Outsourced Data

Krishna Prasanth R[†*], R. Mukkamala[‡], P. K. Baruah[†]
[†]Sri Sathya Sai Institute of Higher Learning, Prasanthi Nilayam, India.
{kprashanth29@gmail.com, pkbaruah@sssihl.edu.in}
[‡]Old Dominion University, Norfolk, Virginia, USA.
{mukka@cs.odu.edu}

***Abstract*—Cloud computing provides on-demand web-based software, middleware, and computing resources. It is a service-oriented model and one of its service is Data as a Service (DaaS), also known as Outsourced Database (ODB) model. Although DaaS solves the problem of storing terabytes of data, the security of the data is a major concern for all the users of the service. Of the several security concerns, privacy and data integrity are the focus of this paper. The privacy and integrity enforcement module introduced in this paper resides on users' computing devices. It performs the desired integrity checks to ensure that the data has not been tampered with at the cloud provider. Here, we explore the benefit of implementing the module on a multicore platform with GPUs, thereby exploiting its inherent parallelism. The speed up is observed to be from 1 to 12 depending on the system architecture and the data size—larger is the data size, higher is the speedup.**

***Index Terms*—Cloud Computing, DaaS, data integrity, parallelism, privacy, outsourcing, speedup**

I. Introduction

Cloud computing provides various resources for the users on demand using internet. The different set of cloud architectures that currently exist are public clouds, private clouds, and hybrid clouds. A given cloud may provide one or more of the following services—software (SaaS), data storage (DaaS), testing platforms (PaaS), networks (NaaS), and infrastructure (IaaS). In this paper, our focus is on clouds offering DaaS that enable data owners to store their data remotely on a cloud server. This model is ideal for small organization to maintain their data without significant investment on stable storage systems. However, since the data is outsourced onto the cloud, these organizations are very much concerned about ensuring security of the data. While data security refers to several aspects such as data integrity, privacy, authenticity, availability, and reliability, in this paper our focus is on privacy and integrity. Data integrity itself deals with three aspects—completeness, correctness, and freshness. Completeness ensures whether or not the cloud provider has sent all data relevant to a particular request or query. Correctness ensures that the data returned is what the data owner had stored and has not been tampered with. Freshness guarantees that the returned results are based on the latest copy of the data reflecting all changes sent to the cloud server.

In this work, we concentrate on confidentiality and tamper-resistant aspects of outsourced data. While encryption is used for confidentiality, Secure Hash Algorithm (SHA1) are used for tamper-resistant property. Thus, if we consider a file or a database consisting of records or tuples, a hash value using SHA is constructed for each record by including all the fields of the record. The hash value is appended to the record and then encrypted using a symmetric key known to the data owner. However, the guarantees provided by the scheme come at the cost of additional processing time for each record while sending data to the cloud provider (for SHA1 and encryption) and while retrieving from the cloud provider (for decryption and verifying the SHA1).

Here, we use multicore and GPU to speedup this process. In particular, we experiment with parallelism at the record level so each record can be handled independently and in parallel. Depending on the size of the data and the number of processors and the architecture, we observe a speedup up as much as 12. We have also implemented the same using CUDA. In almost all cases, CUDA implementation on GPUs provided a higher speedup than a limited multi-thread implementation on multicore CPUs. Thus, it is clear that the extra processing due to outsourcing is not going to adversely affect the throughput of a system.

The paper is organized as follows. In section 2, we

[*]Student author

provide a brief summary of the related work. Section 3 describes the underlying system model. Section 4 describes the proposed approach. In section 5, we present the obtained experimental results. Finally, section 6 summarizes the observations and conclusions from this work.

II. Related Work

In this section, we look at some of the past research related to the present work. In particular, we discuss past work in the area of data integrity and encryption.

Challenge tokens have been a common tool used by several researchers for verifying the integrity of outsourced data. For example, Ateniese et al. compute challenge tokens in the setup phase while uploading a file and use same challenge tokens when verifying the integrity of the returned results [1]. Sion used fake queries as challenge tokens [2]. Here, when a batch of queries are submitted to the cloud server, some fake queries are inserted into the batch. The answers for the queries were precomputed at the owner. Aggregated Signatures using B+ trees has been another technique in this direction [3]. Here, during the set up phase, prior to offloading the data to the cloud provider, digital signatures on the blocks are used to form a B+ tree. The root of the tree is used in the verification phase. Merkle hash trees are also employed for enforcing integrity in static databases [4]. Here, the data owner computes the Merkle hash tree and the root of the tree is known as verification object. The verification object is distributed to all the users. When the cloud service facilitator sends the data, the users computes the root and verifies whether the one obtained is same as verification object during the verification phase.

Bloom filters are also used for providing fine grained integrity [5]. Here, the fields in a record are mapped using a set of hash functions and the hashes inserted into a filter, a string of bits. The owner could store the Bloom filters for the records and verify them when the results are retrieved from the server. Of course, it is a probabilistic data structure and hence might give rise to false positives. To provide deterministic way of proving the integrity, condensed-RSA is used [8]. It is computed by the homomorphic property of the RSA digital signature. Single aggregated signature is computed using multiplication of all the signatures of the tuples that are part of the results which is computationally intensive.

In the context of frequent itemset data mining, fake item insertion has been used to validate the integrity of the returned results [9]. Here, knowing the expected frequent itemsets for the fake items, the owner verifies whether or not the results returned by the cloud provider are correct.

In this paper, we employ encryption for symmetric key encryption and Secure Hash Algorithm (SHA1) for integrity.

III. System Model

In this section, we describe the model used in the paper. In general, data is stored in the form of files or databases. While data is logically divided into blocks in a file, it is divided into tuples in the case of database. In the case of database, for example, other levels of granularity are at table (or relation) level and attribute level.

In order for the model to be applicable at all levels, we use a generic term *data granule*, and represent it as D. So each data file (database) has a set of n data granules, say $\{D_1, D_2, D_3, ..., D_n\}$. Let $D = \{D_1, D_2, D_3, ..., D_n\}$, be the *Data space*.

To enforce confidentiality, as mentioned before, we use symmetric key encryption. Depending on the level of confidence needed, we could use a single key, multiple keys, or one key per granule. Let K be the $KeySpace$, where $K = \{k_1, k_2, k_3, ..., k_n\}$. Once the data granules are encrypted using an encryption function (say *Encrypt*), they result in cipher granules. We represent the *Cipher space* as C. Similarly, the function *Decrypt* decrypts the cipher granules back to plain data granules.

Similarly, for integrity, let H be the domain of the hash value that are output of the SHA1 hash function (say $Hash$) and let h represent a hash value for a granule. In summary,

$$Hash : D \longrightarrow H$$
$$Encrypt : K * D \longrightarrow C$$
$$Decrypt : K * C \longrightarrow D$$

IV. Proposed Approach

The paper is primarily geared towards improving the throughput of the data owner while uploading data to the cloud provider and while retrieving data from there. During the uploading, it needs to encrypt the data granules, and compute the hash value using SHA1 for each granule. This is referred to as *Storage phase*. The processing step during information retrieval is referred to as *Retrieval phase*. We now describe these two phases.

- *Storage phase*: Here, each data granule is read from the data file, its hash values, h computed. The hash value is appended to the data granule. The data granule is now encrypted. The encrypted Block C

is now written to file F' which is then outsourced to cloud provider. This process is summarized in Algorithm 1.

Algorithm 1 Storage Phase

Input: Generate $k \in KeySpace$
Input: File F which has to be outsourced.
Output: Encrypted File F'
  while NOT END OF FILE **do**
    Read next granule $D \leftarrow F$
    Compute h using D and $Hash$
    Append h in $D \rightarrow (D, h)$
    Compute $Encrypt(K, (D, h)) \rightarrow C$
    Append C to Encrypted file F'
  end while

- *Retrieval phase*: During this phase, the owner has downloaded the encrypted file F' from the cloud provider. Here, each data granule is first decrypted and the hash value is extracted. The hash value is checked with the data part for integrity. Once it is verified, the plain data granule is used to form file *F*. Algorithm 2 summarizes these steps.

In the both the phases, the two operations of computing hash value and encrypting the block is performed on set of Blocks, say $\{D_1, D_2, D_3, ..., D_n\}$. Since these two operations are performed on each block, they can be parallelized using the support of underlying hardware.

Algorithm 2 Retrieval Phase

Input: Generate $k \in KeySpace$
Input: Encrpyted File F' which has been outsourced.
Output: File F
  while NOT END OF FILE **do**
    Read next cipher granule $C \leftarrow F'$
    Compute $Decrypt(K, C) \rightarrow (D, h)$
    Extract $h \leftarrow (D, f)$
    Compute h' using D and $Hash$
    Append D to the File F
    if $h \neq h'$ **then**
      Integrity is lost.
      Exit
    else
      Integrity is maintained.
    end if
  end while

In the above two phases, both hash function and encryption (decryption) on each granule are performed sequentially. These are computationally intensive operations. Here, we propose to improve the response time of both the phases by employing parallel architectures to parallelize the sequential steps. Here, we experiment with multicore and GPU architectures.

A. Implementation of Application using multicore

The granules $D_1, D_2, D_3, ..., D_n$ are read from the file F to a buffer. Using omp_set_num_threads() function number of threads that have to be created is identified. Using the number of the threads that are created, the file is partitioned into those many chunks and each chunk is given to a single thread. Each chunk of data may have several granules. The operations on each chunk are performed in parallel, since each thread is assigned to a single core. The thread performs the hash computations and encryption (decryption) for the granules in the chunk assigned.

B. Implementation of Application using CUDA

The granules $D_1, D_2, D_3, ..., D_n$ are read from the file F to a buffer. The buffer is copied to the GPU device using cudaMemcpy. The number of threads created is equal to the number of granules that are present in the file. Each granule is assigned to a thread. The CUDA kernel performs the hash and encryption (decryption) functions. Each thread block has fixed number of threads as 1024. Number of thread blocks depends on the number of blocks D in the file.

V. Experimental Results

To measure the effectiveness of parallelism and parallel architectures in improving the performance of tasks involved in enforcing integrity and confidentiality for outsourced data, we have run several experiments. Here, we describe the experiments and the obtained results.

In order to measure the effect of the data size on performance, we have chosen data of three different sizes: 10 MB, 100 MB, and 1 GB. Speedup was chosen as the measure of effectiveness. We have used C-based code using OpenMP with multiple threads (1, 2, 4 and 16) on multicore systems. We have used CUDA code on GPUs which have numerous cores resulting in potentially millions of threads. Three different machines were used to run the storage phase and the retrieval phase—*Gordon* [10], *Forge* [11], and Local High Performance Computing (LHPC) node.

First system, the *Gordon*, is a data-intensive supercomputer. It has a compute node with 16 cores using Intel(R) Xeon(R) E5-2670 running at a clock rate of 2.60

GHz. Second system, the *Forge* (NCSA) supercomputer, has a compute node with 16 AMD cores and 64 GB of RAM. It is connected to six M2070 GPU units. Finally, the third one, the *LHPC*, has Intel(R) Xeon(R) CPU X5650 with a clock rate of 2.67 GHz. It has 6 cores per socket. Since the system is dual socket, there are 12 cores with 24 GB of RAM. It is connected to 4 Tesla S2050 GPU units. In summary, all three have multicore CPUs. In addition, Forge and LHPC have GPU units.

We measure the effectiveness of these three architectures on the three data sets during the storage phase and the retrieval phase of the data. We first look at the results from the runs on the Gordon system. We noted the following:

- In the storage phase (Fig. 1), good speedup is achieved with increase in the number of threads across all data sizes. While the speedup was almost the same in the case of 10 MB and 1 GB files, it was relatively lower for 100 MB.
- Almost similar speedup is observed even in the data retrieval phase (Fig. 2). Surprisingly, the speedup is almost identical for all the three data sets.

Let us now look at the results from the Forge system. We make the following observations:

- In the storage phase (Fig. 3), the speedup for Forge is about 25% less than that of the Gordon system for 1 to 8 threads. Surprisingly, even in the case of CUDA with almost unlimited number of threads, the speedup was lower than that of Gordon. This clearly indicates that the sequential part of the storage phase is a bottleneck in further improvements to speedup.
- In the retrieval phase (Fig. 4), while the speedup with threads 1 to 8 is almost identical to that on Gordon, the CUDA implementation was better for the 1 GB file. It was lower than that of on Gordon for 10 MB and 100 MB. Surprisingly, the speedup for the 100 MB was the worst of all three data sets. This clearly indicates that the way the data is stored and retrieved on Forge are playing a vital role in the performance of this phase.

Our observations for the LHPC system are somewhat different with respect to different data sizes (Fig. 5 and Fig. 6). The speedup was the lowest for 10 MB files and the maximum speedup that could be achieved at this size was only 4. The speedup increased with the data size. Maximum speedup with 1GB file was 10 for both storage and retrieval phases. In general, the speedup on LHPC was lower than that of the other two systems.

From these experiments, we conclude that: (i) A maximum speedup of about 12 can be achieved during storage and retrieval phases depending on the hardware architecture. This is more likely achievable with GPUs, especially for large data files. (ii) Typically, storage phase has lower achievable speedup than the retrieval phase. (iii) Besides the number of threads and CPU speeds, data storage and management policies seem to play a vital role in determining the achievable speedup. (iv) Unless, the two phases are further optimized, the current sequential portion of the two phases, which is typically at the start and end of the phases, no further improvements are possible in the speedups.

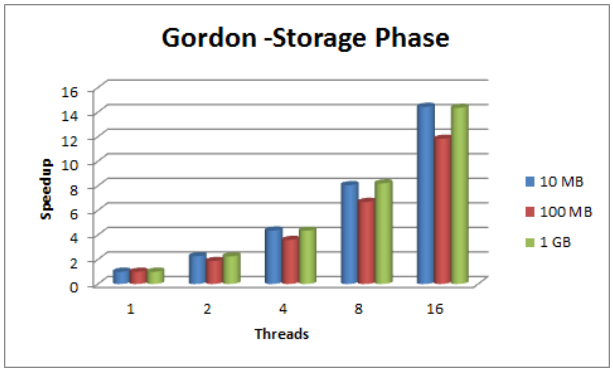

Fig. 1. Gordon Storage Phase

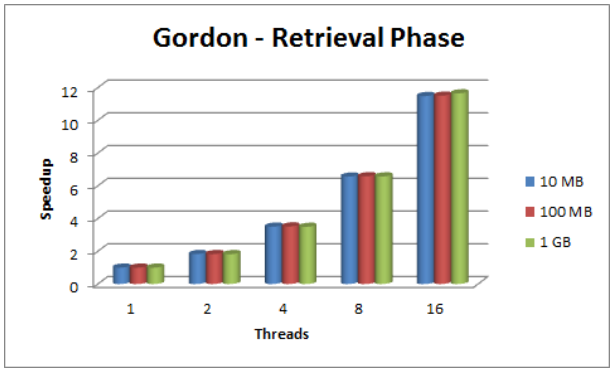

Fig. 2. Gordon Retrieval Phase

VI. Conclusion And Future Work

It is important to provide data integrity and privacy for the data that has to be outsourced. In this paper, we have considered symmetric key encryption for data confidentiality and SHA1 hash algorithm for the tamper-resistance. In order to remove the performance bottlenecks in executing the additional steps at the time of storage and retrieval of data, we have implemented them

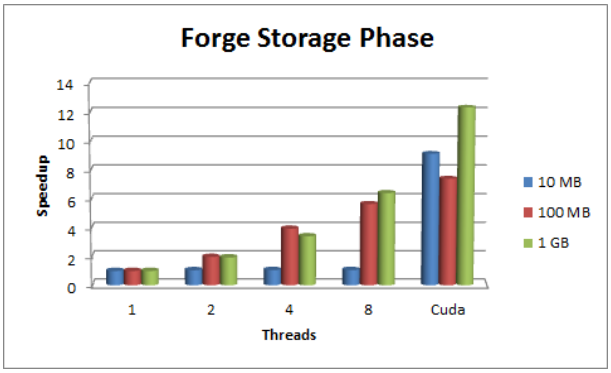

Fig. 3. Forge Storage Phase

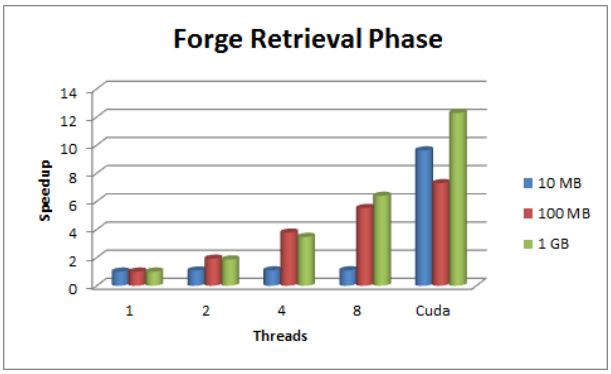

Fig. 4. Forge Retrieval Phase

on three different systems—Gordon, Forge, and Telsa. While they all resulted in a speedup, the maximum that could be achieved was only 12, even with GPUs. The future work should look at optimizing the initial and final steps of each of the storage and retrieval phases so better speedups can be obtained. For large data sets the speedup can be increased further using Multiple GPUs.

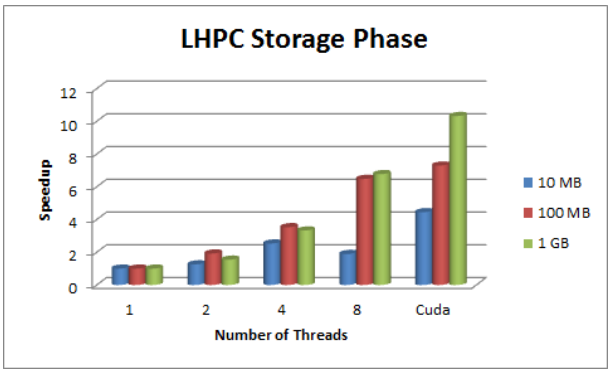

Fig. 5. LHPC Storage Phase

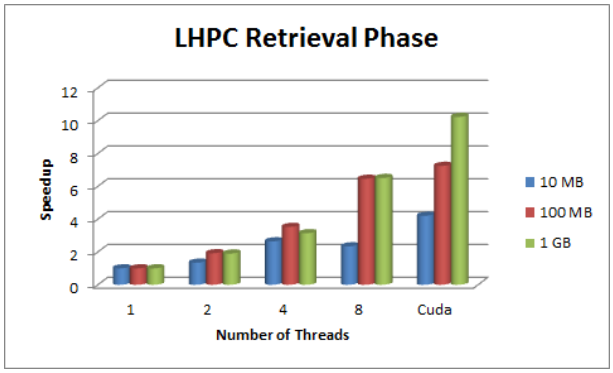

Fig. 6. LHPC Retrieval Phase

ACKNOWLEDGEMENT

We dedicate this work to the founder Chancellor of SSSIHL, Bhagawan Sri Sathya Sai Baba. This work was partially supported by NVIDIA, Pune grant under Professor partnership program and the Extreme Science and Engineering Discovery Environment (XSEDE).

REFERENCES

[1] G. Ateniese, R. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession", SecureComm 2008, September 22 - 25, 2008, Istanbul, Turkey.
[2] R. Sion, "Query Execution Assurance for Outsourced Databases", Proc. Very Large Databases Conf., VLDB 2005.
[3] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic Authenticated Index Structures for Outsourced Databases", SIGMOD 2006, June 2729, 2006, Chicago, Illinois, USA.
[4] M. Gertz, A. Kwong, C. U. Martel, G. Nuckolls, P. T. Devanbu, and S. S. Stubblebine, "Databases that tell the Truth: Authentic Data Publication", IEEE Data Engineering Bulletin, Volume 27, Number 1, March 2004.
[5] M. Zhang, K. Cai, and D. Feng, "Fine-Grained Cloud DB Damage Examination Based on Bloom Filters", Proceeding WAIM'10 Proceedings of the 11th international conference on Web-age information management, pp. 157-168, Springer-Verlag Berlin, Heidelberg, 2010.
[6] J. Heurix and T. Neubauer, "On the Security of Outsourced and Untrusted Databases", 9th IEEE/ACIS International Conference on Computer and Information Sciences, 2010.
[7] A. Opera, M. K. Reiter, and K. Yang, "Space-Efficient Block Storage Integrity", Proc. NDSS'05, 12th Annual Network and Distributed System Security Symposium, San Diego, California, 3-4 February 2005.
[8] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and Integrity in Outsourced Databases", ACM Transactions on Storage, Volume 2 Issue 2, May 2006, pp. 107 - 138, ACM New York, NY, USA.
[9] W.K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis, "An Audit Environment for Outsourcing of Frequent Itemset Mining", Proc. VLDB Endowment,Volume 2 Issue 1, August 2009, pp. 1162-1173
[10] https://www.xsede.org/sdsc-gordon.
[11] https://www.xsede.org/ncsa-forge.