Increasing the Functionality of the Modern NoSQL-systems with GPGPU-technology

A.A. Kozlov[1], A.A. Aleshina[2], I.S. Kamenskikh[3], M.M. Rovnyagin[4], D.M. Sinelnikov[5], D.A. Shulga[6]

National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)
Moscow, Russian Federation

[1]andrewkozlov@icloud.com, [2]aleshina1403@yandex.ru, [3]ilya.kamenskikh@gmail.com,
[4]m.rovnyagin.2015@ieee.org, [5]dsinelnikov96@gmail.com, [6]uinos@yandex.ru

Abstract — **The object of research is a distributed database management system Apache Cassandra and methods to improve its functionality by applying hybrid computing technologies. The encryption standard GOST 28147-89, which has a great potential for parallelization on the graphics core and can be used as the expansion module. Development was carried out in the Java language using the open source project Apache Cassandra and JCuda parallelization library for GPGPU systems. The article contains analysis of the Apache Cassandra architectural features and review of possibilities for connecting the expansion modules. The article presents a mathematical model of the proposed solution and experimental results (graphs of recording data rate to the base: in the standard mode, with using CPU encryption, with using GPU encryption).**

Keywords — **CPU/GPU hybrid architecture, NVIDIA CUDA, Distributed Databases, Encryption**

I. Introduction

Starting from the late 2000s, many NoSQL-systems having different capabilities have been developed. Some of them implement the programming interface that provides translation of some classic RDBMS instructions to internal representation, while others partially implement ACID (Atomicity, Consistency, Isolation, Durability) requirements. A common feature of such systems is their high index of data rate scaled (linearly, in most cases) depending on the number of storage servers used. According to [1], Apache Cassandra [2] is a NoSQL-system exhibiting the highest performance at the present time.

Today, the NoSQL approach is a de facto standard in implementing systems to handle Big Data. Many companies worldwide are currently applying high-performance distributed NoSQL solutions. Services of such corporations as IBM, Amazon, Facebook, Twitter, Google, Oracle, etc. directly depend on the efficiency of data retrieval and storage systems. The following indices are selected from the key efficiency criteria of NoSQL solutions: data rate, scalability, energy efficiency, maintenance costs. These indices evidently relate to requirements specified for state-of-the-art supercomputers [3].

At the present time, the leading position in supercomputer ratings are held by hybrid solutions [4], when a computing system (CS) incorporates both standard CPUs and special-purpose computers or discrete coprocessors (GPU, MIC, FPLD, etc.). These devices substantially speed up some operations and increase energy efficiency [5].

From the date of their creation, development technologies for hybrid systems (such as NVIDIA CUDA and AMD APP) have had a widespread feedback in the scientific environment. A significant number of works discuss the issues of higher performance of cryptoprotection algorithms using the CUDA technology.

The authors of the research [6] reviewed in their work several possible approaches to implementing a multithread AES version. The article presents comparison results for the performance of serial and parallel CPU and GPU versions. As a result, the GPU implementation turned out to have the highest performance.

In article [7], the authors overview all AES encryption modes and specify those fit for parallelization: ECB (Electronic codebook), CTR (Counter). For CBC (Cipher-block chaining) mode, parallelization may only be applied to the decryption operation. The authors present data rate graphs for these modes and give some advice on increasing the performance.

II. Problem Statement

Currently available NoSQL solutions including Apache Cassandra described in this paper are not intended for application in hybrid computing systems because all operations are carried out only by central processing units. In addition, they lack built-in security mechanisms for stored data, whereby stealing of a data medium, for example, a hard drive from a storage server, will inevitably result in data leakage. Therefore, the development of methods and tools for solving protected data storage problems with the application of hybrid supercomputer technologies is a crucial scientific and engineering task [8].

Selection of a specific encryption algorithm depends on the purpose of a high-performance retrieval and storage system. If the system is planned to be applied in

national security tasks, the GOST 28147-89 algorithm (hereinafter, GOST) is logically used as f. GOST is a block encryption algorithm based on the Feistel network. The Standard [9] specifies fours operation modes:

1) simple substitution;

2) stream;

3) stream with feedback;

4) authentication code generation.

The authentication mode is intended for checking the absence of accidental or deliberate distortions in the ciphertext, i.e. it is not an encryption mode as is generally understood.

Parallelization of GOST, as of any block cipher, may be accomplished only for some modes. In the simple substitution mode, there are no dependences between operations of encrypting plaintext blocks, and parallelization is thus possible. In the stream mode, gamma is independently applied to different plaintext blocks. The gamma generation process itself represents modulo 2^{32} and 2^{32-1} addition of the contents of respective half-blocks with constants. This recursion is easily transformed into parallel type. Thus, the stream mode is suitable for parallelization. Gamma in the stream mode with feedback is formed on the basis of a previous block of encrypted data. This mode is not parallelized.

In the stream mode, the initialization vector is one of the encryption function parameters besides the encryption key. To meet the requirement of one-to-one mapping, the initialization vector when accessing the system for identical data should be the same. In this case, however, if differing data are added to the storage, one and the same gamma will be generated. The difference between the ciphertext blocks will be caused only by the difference between the data, which is inacceptable. Therefore, the stream mode does not provide randomization of keys added to the tree. The simple substitution mode can thus be the only GOST mode applicable to implement the data storage system described above [10].

Let us consider a data storage subsystem. The storage nodes control data recording to random access or external memory, additionally processing the received data.

When handling data in RAM, Apache Cassandra utilizes commitlog [11] and table files on the hard drive. The tables consist of entries (lines), which in their turn are a set of cells. In the case of memory overflow (Java-heap [12]), a new table is generated in the memory (Memtable), while the previous one is recorded to the disk (SSTable). The log recording and table uploading procedures generate hard drive accesses, which expedites the system operation. Fig. 1 shows the operational scheme of the Apache Cassandra memory.

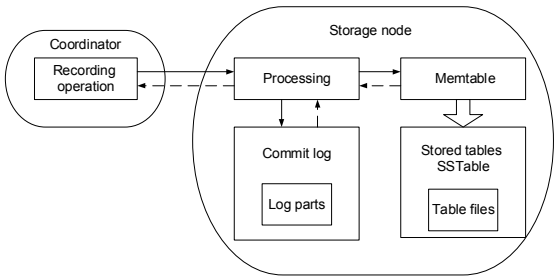

Fig 1. Apache Cassandra memory organization

This work analyzes subsystems responsible for Memtable and SSTable functioning.

Fig. 2 shows a layout view of a data storage node. The buffer data storage subsystem (Memtable) interacting with the user query processing subsystem is implemented in package org.apache.cassandra.db.Memtable. To record the table on the disk, the system must serialize its data. In Apache Cassandra, each table cell undergoes serialization individually, using for this purpose org.apache.cassandra.db.rows.Serializer class. This process was modified in the course of operation so as to add the user data encryption capabilities. The data further enters the data storage subsystem in the external memory (org.apache.cassandra.io.sstable packages), which records it on the disk and is also responsible for reading from the disk for subsequent use in the buffer.

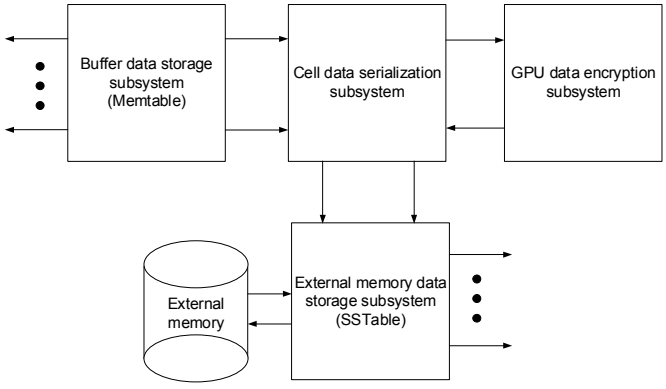

Fig 2. Diagram of data storage node

The data storage node and the distributed data storage system based on these components may be presented as a queuing network [13].

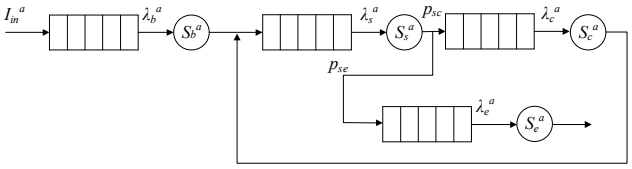

Fig. 3. Queuing network of storage node

Fig. 3 shows a queuing network that corresponds to the storage node diagram in Fig. 2. The following notation is introduced to formalize the task:

S^a – a set of subsystems of the data storage node;

S_b^a – buffer data storage subsystem;

S_s^a – cell data serialization subsystem;

S_c^a – data encryption subsystem;

S_e^a – external-memory data storage subsystem;

I_{in}^a – arrival rate of queries at storage node j;

λ_i^a – arrival rate at queuing system S_i^a;

Table 1 presents a probability matrix of storage node transmissions corresponding to the queuing network.

TABLE I. PROBABILITY MATRIX OF STORAGE NODE TRANSMISSIONS

	S_b^a	S_s^a	S_c^a	S_e^a	extern.
S_b^a	0	1	0	0	0
S_s^a	0	0	p_{cs}	p_{ce}	0
S_c^a	0	1	0	0	0
S_e^a	0	0	0	0	1
extern.	1	0	0	0	0

A model, which can be described by the below equation, was constructed to simulate the application operation on CPU/GPU:

$$A = \frac{T}{T_S+\tau} = \frac{T}{T_S+(\tau^{CPU}+\tau^{GPU})} = \frac{1}{(1-\alpha)+c\left(\frac{\alpha(1-\beta)}{n}+q\right)+\frac{\alpha\beta}{p^{GPU}}}, \quad (1)$$

where p is the number of processors (processor cores CPU/GPU/FPGA-CORES), n – number of flows, T – time of execution of the serial algorithm version, T_p – time of execution of the parallel program segment, T_s – time of execution of the serial program segment, t_c – context switch time (switching of the processor core from executing one thread to another), coefficient $q = t_c/T$, τ – time of execution of a program segment on a real processor (CPU/GPU), t – time of execution of a program segment on an ideal system (n = p), α – parallelism factor, $\alpha = T_p/T$, β – coprocessor utilization factor, A – acceleration.

The time of the query in the high-performance retrieval and storage system is thereby determined by the following equation:

$$\overline{T}_{\mathrm{pf}_a} = \frac{\lambda_b^a \frac{\overline{T}srv_b}{A_b-\lambda_b^a\overline{T}srv_b}+\lambda_s^a \frac{\overline{T}srv_s}{A_s-\lambda_s^a\overline{T}srv_s}+\lambda_c^a \frac{\overline{T}srv_c}{A_c-\lambda_c^a\overline{T}srv_c}+\lambda_e^a \frac{\overline{T}srv_e}{A_e-\lambda_e^a\overline{T}srv_e}}{I_{\mathrm{in}}^a}. \quad (2)$$

Acceleration factors A^i in the above equation are determined by the relation (2) and depend on the hardware configuration of the high-performance retrieval and storage system, applied algorithms, and their parallelizability (factors α, β).

III. IMPLEMENTATION OF GOST ENCRYPTION FOR HYBRID SYSTEMS

The storage nodes are connected to CUDA accelerators. The CUDA runtime is responsible for the execution of the parallel code of the CUDA core on the accelerator. The major difference between the interaction of CUDA runtime and CUDA interface and the interaction among other storage node modules is in a PCIe bus between CPU and the accelerator. Although up-to-date CUDA implementations make it possible to control the single address space, the issue actually concerns interaction between two devices, each having its memory and address space. At the same time, a CUDA coprocessor may be physically located in the same case with CPU or in a separate coprocessor unit. As regards the high-performance retrieval and storage system, a limited-performance interface between CPU and the coprocessor leads to additional delays in the coprocessing program run. Accordingly, the number of exchanges between a CUDA device and a host system (storage node or access server) should be minimized, and processing should preferably be performed in the batch mode. Similar reasoning may be applied to an external storage device utilized by the storage node. Data uploading/downloading from/to the external storage device occurs with a delay caused by parameters of the interface between the storage module and the external storage device.

Therefore, the Apache Cassandra modification is architecturally organized according to modularity. Storage nodes are physically represented by Java processors implemented on the nodes of Apache Cassandra equipped with CUDA coprocessors. Each Java process can control the CUDA accelerator using jCuda library for this purpose.

The jCuda library was used to provide interface between a CUDA program (referred to as CUDA core, or core) and the major application. The library and the involved IDE do not allow CUDA core debugging. Therefore, the cores were debugged in Nsight Visual

Studio Edition in an off-line mode. The code of the debugged cores was included into the master project in the JetBrains Idea environment.

Within the frames of this work, the GOST 28147-89 encryption algorithm was implemented (used in implementing the transparent data encryption technology illustrated in Fig. 4).

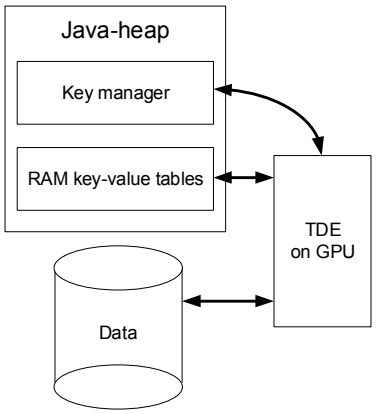

Fig. 4. Transparent data encryption

Two algorithm versions were written: CPU and GPU. In both GOST 28147-89 versions, two encryption modes were implemented: simple substitution and stream. A table used in cryptographic applications by the Central Bank of the Russian Federation was applied as the substitution unit (S-block) [14].

Function f and the encryption algorithm for one 64-bit block on CUDA are described similarly to CPU version. Differences result from mismatch of the data types in Java and CUDA C languages. For example, a 64-bit value is represented in CUDA C with the help of the unsigned long long type, while in Java the type is long. At the same time, due to the availability of unsigned types, the need for masking is less frequent in CUDA C as opposed to Java.

IV. Implementation Effectiveness Evaluation

Response time is the major criterion for the performance efficiency of a high-performance retrieval and storage system. This chapter presents the experimental study results for a modified Apache Cassandra system. The basic parameters measured are the response time and the data rate.

The modified Apache Cassandra is a distributed cluster data-storage system having hybrid architecture. That is why the experimental study of its properties may involve methods for load testing of cluster systems taking into account the hybrid architecture properties [15].

The testing was provided by Apache Jmeter utility configuration.

The experiment produced data about the average number of transactions per second for different functioning conditions of Apache Cassandra: an unmodified version from an official repository, a version with encryption on the central processor, a version with encryption with the application of Nvidia CUDA. The bar charts are shown in Fig. 5.

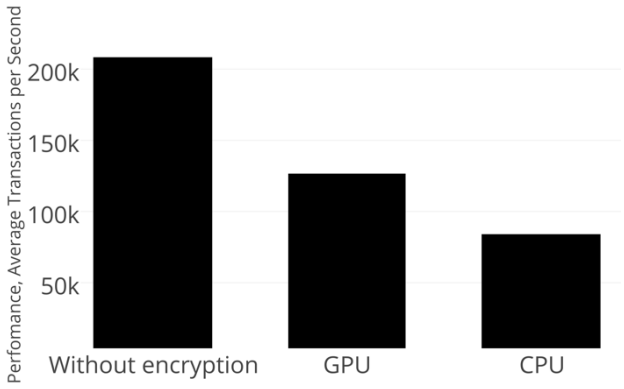

Fig. 5. Bar chart of system performance depending on data encryption technique

The efficiency of the technique also depends on the size of the data arriving in the system. It is determined under test conditions by the size of a string in the query for adding data to the base. The size threshold was determined, starting from which the implementation on the GPU becomes more efficient that that on the CPU. The plot is given in Fig. 6.

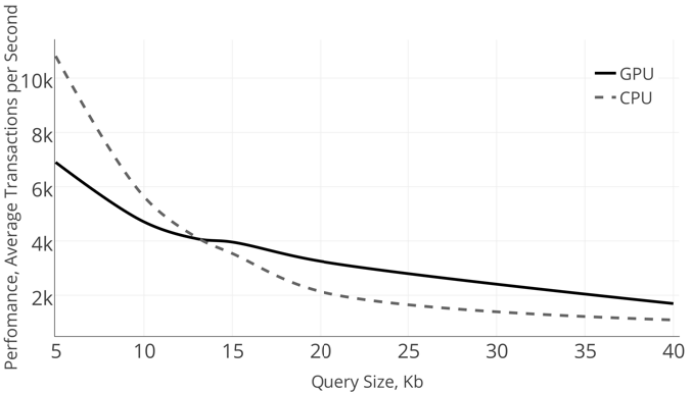

Fig. 6. Plot of system performance against query size

The experiment was conducted on a test bed specified in Table 2.

TABLE II. TESTING SYSTEM

Module	Model	Specification
Central Processing Unit	Intel Core i7	2,8 GHz
Random Access Memory	16 Gb	DDR3
Graphics Processing Unit	Nvidia GTX 760	Compute capability 3.0
Initial Java heap size	4096 Mb	
Maximum Java heap size	4096 Mb	

V. CONCLUSION

The work results in a data encryption module for Apache Cassandra. Versions with computation both on the CPU and on the CPU/GPU were implemented. Due to the application of hybrid architecture, the system performance losses were reduced to an acceptable level.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the Head of Computer Systems and Technologies Department of the National Research Nuclear University "MEPhI," Professor M. A. Ivanov for his assistance and support during the research, as well as to the Head of Inter-Facility Hybrid Computing Systems Laboratory, Assistant Professor N. P. Vasilyev for the provided equipment and consultations.

REFERENCES

[1] Benchmarking Top NoSQL Databases. [Online]. Available: http://blog.endpoint.com/2015/04/new-nosql-benchmark-cassandra-mongodb.html

[2] The Apache Cassandra Project. [Online]. Available: http://cassandra.apache.org

[3] Exascale Supercomputers: Achievements and Prospects. [Online]. Available: http://2013.nscf.ru/TesisAll/Plenar/0_02_Eisymont.pdf

[4] TOP500 Supercomputer Sites. [Online]. Available: http://www.top500.org

[5] Hybrid reconfigurable computing system layout and high-accuracy calculations based on it. [Online]. Available: http://2013.nscf.ru/TesisAll/Plenar/06_1140_LacisAO_P06.pdf

[6] Ortega, J. Trefftz, H. Trefftz, Parallelizing AES on multicores and GPUs, IEEE International Conference on Electro/Information Technology (EIT), 2011, vol., no., pp.1-5, 15-17 May 2011.

[7] Qinjian Li et al. Implementation and Analysis of AES Encryption on GPU, Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, p.843-848, June 25-27, 2012.

[8] S. Srinivas, A. Nair, Security maturity in NoSQL databases - are they secure enough to haul the modern IT applications?, International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2015, p.739- 744, 2015.

[9] GOST 28147-89. Encryption Algorithm [Online]. Available: http://protect.gost.ru/document.aspx?control=7&id=139177

[10] S. Ilyin, V. Korobitsin, GOST-28147 Encryption Implementation on Graphics Processing Units, Third International Conference on Availability, Reliability and Security, 2008, p. 967-974, March 4-7, 2008.

[11] Cassandra Durability. [Online]. Available: http://wiki.apache.org/cassandra/Durability

[12] H. Schildt. Java 8. The Complete reference. Moscow: Dialektika, 2015.

[13] B. Gnedenko. Introduction to Queuing Theory. Moscow: LKI, 2011.

[14] B. Schneier. Applied Cryptography. Protocols, Algorithms, and Source Code in C. Moscow: Triumf, 2002.

[15] Dyumin, A.A.; Kuznetsov, A.A.; Rovnyagin, M.M., "Evaluation of statistical properties of a modified Bloom filter for heterogeneous GPGPU-systems," in Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), 2015 IEEE NW Russia , vol., no., pp.71-74, 2-4 Feb. 2015