Evaluation of statistical properties of a modified Bloom filter for heterogeneous GPGPU-systems

A. A. Dyumin, A. A. Kuznetsov, M. M. Rovnyagin
National Research Nuclear University "MEPhI"
Moscow, Russian Federation
a.a.dyumin@ieee.org, dplex14@mail.ru, mmrovnyagin@mephi.ru

***Abstract* – The object of the research is a modified Bloom filter with counters representing the probabilistic data structure that contains information about the items in a data store. As a result of filter operation false-negative responses are excluded, however occurrence of false-positives is possible. The system is developed in Java using jCuda libraries to parallelize the algorithm on GPGPU systems. The article presents the mathematical evaluation of the probability of false-positives for the modified Bloom filter with counters. A statistical analysis has been made for runs with different parameters: number of hash functions, length of the counter, filter size, number of added elements. The article presents corresponding graphs of the probability distribution of false-positives.**

Keywords—Bloom filter, CPU/GPU hybrid architecture, NVIDIA CUDA, hashing

I. Introduction

At the present time NoSQL [1] is widely used to develop high-performance data storage and search systems. Usually NoSQL (not only SQL) systems do not possess complete functionality of relational database management systems (RDBMS), nevertheless they significantly increase throughput capacity of applications when processing large quantities of semi-structured data. Examples of such data are experimental data, web-service access statistics, weather data, dataflow from social networks, microblogs, etc. In each of the cases it is possible to implement parallel data processing on a distributed computing system. However, it would require storage systems capable of processing millions of simple queries.

NoSQL systems realize a program interface that translate some instructions of conventional RDBMSs into an internal representation; some of them partially implement the requirements of ACID. One common feature of such systems is high throughput capacity, scalability (in most cases linearly) in terms of number of used storage servers.

Presently many companies all over the world use the distributed high-performance NoSQL solutions. Servers of corporations like Amazon [2], Facebook [3], IBM [4] and Google [5] depend implicitly on the data storage and retrieval systems effectiveness.

Highly effective versions of data storage and search algorithms are used in modern NoSQL systems. The membership check algorithms considerably enhances effectiveness of the systems.

To resolve this problem, in particular, Bloom filter is used. This is a probabilistic data structure, proposed in 1970 by Burton Bloom [6]. Bloom filter provides information about the elements in an arbitrary data warehouse containing information in the format "key-value". Conventional realization of the filter is a store add-in in the form of a bit array of M bits (Bloom vector) initially containing zeros.

When a new data element is entered into the store, K hash-function of the key is computed, where values of the hash-function lay in the interval from zero to M-1. Algorithm writes 1-bits into the positions of the vector whose numbers are determined by the results of hash functions.

Traditional Bloom vector does not permit element deletion. This problem can be handled by utilizing a modification of Bloom filter with counters [7]. Within this approach Bloom vector is essentially an array of L-position counters storing values ranging from 0 to 2L–1. When new data is added to the store the hash-function computes key value and increments the required counter.

In the event when a counter reaches the maximum value its value remains unchanged. In the event of a deletion the counter is decremented by one, and is not changed when its value reaches zero or the maximum.

II. Problem Statement

Various scientific papers [8, 9, 10] demonstrate findings of research of common version of Bloom filter intended for heterogeneous supercomputer systems. These implementations use GPGPU coprocessors to accelerate execution of Bloom filter.

However, these implementations are hardly applicable for the use in NoSQL systems (most of which are cross-platform Java solutions). The aim of this article is the development of a cross-platform implementation of Bloom filter for heterogeneous GPGPU systems providing the possibility of its use in advanced NoSQL systems.

Within this project were investigated the performance and the statistical properties of the modified Bloom filter with counters.

978-1-4799-7306-4/15/$31.00 ©2015 IEEE

III. STATISTICAL PROPERTIES OF A MODIFIED BLOOM FILTER

Addition of new values to a modified Bloom filter causes uniform counter increment (provided an efficient hash-function be selected). It is known that for conventional Bloom filter probability of false-positives is:

$$p^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \quad (1)$$

The above estimation is true provided events of setting Bloom bit vector to one are considered independent. In the paper [11] researchers of Carleton University demonstrate the exact formula for estimation of false-positives:

$$p_{k,n,m} = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} i^k i! \left(\frac{m}{i}\right) \left\{\frac{kn}{i}\right\} \quad (2)$$

Nevertheless, the first formula (proposed by Bloom) may be used as the low bound of asymptotic estimated probability of false-positives. Consequently:

$$p^k < p_{k,n,m} \le p^k \times \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}\right)\right) \quad (3)$$

The above formulas are unacceptable for a modified Bloom filter since they do not account the capacity of the vector elements. If an effective hash-function is selected, than growth of counter values within the vector will be uniform.

Non-zero value of a modified Bloom vector can be viewed as a true value within a position of a conventional Bloom vector. Thus a vector consisting of g L-position counters has the following probability of false-positive (lower bound):

$$p^g = \left(1 - \left(1 - \frac{1}{g}\right)^{kn}\right)^k \quad (4)$$

The substitution:

$$g = \frac{m}{L} \quad (5)$$

presents the formula for estimation of probability of false-positives of a modified Bloom filter (since the switching events of the counter are independent - this is lower asymptotic estimate):

$$p_{\text{mod}} = \left(1 - \left(1 - \frac{L}{m}\right)^{kn}\right)^k \quad (6)$$

With a single-digit counter L=1 (conventional Bloom filter) the formula is transformed to (1).

As the capacity of a counter increases, probability of false-positive increases for a fixed number of added elements, n, vector length, m and number of hash-functions, k (L=2 and L=3 of Figure 1).

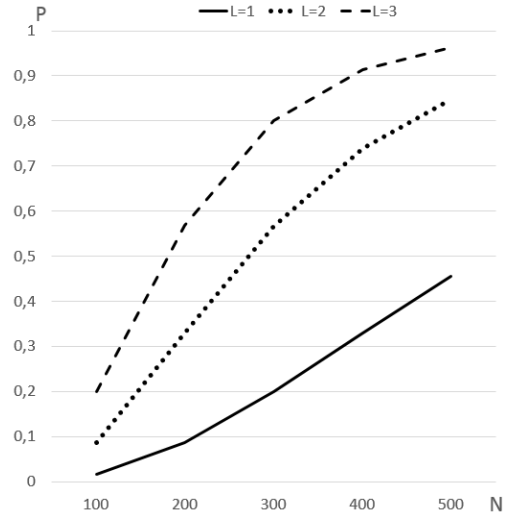

Fig. 1. Probability of false-positives against the number of added elements (for different counter sizes L)

For experimental verification of the resulting correlations a Java application was developed. Pseudo-random numbers were used as the values stored in the filter. These values were added to the filter and to the store.

After that an array of data was formed to verify their existence within the store and the filter. A search for the values was performed within the store and the Bloom filter. Search results were compared against each other. Observed probability of false-positive of the Bloom filter was estimated according to the following formula:

$$RESULT = \frac{FALSEPOSITIVE}{POSITIVE + NEGATIVE + FALSEPOSITIVE} \quad (7)$$

Legend:

- POSITIVE – the Bloom filter indicates the existence of a key, the key was found in the store.
- FALSEPOSITIVE – the Bloom filter indicates the existence of a key, the key was not found in the store.
- NEGATIVE – the Bloom filter indicates the absence of a key, the key was not found in the store.

Consequently, formulas (6) and (7) allow theoretical and observed estimation of the false-positive rate of the Bloom filter, which influences its effectiveness.

Observed diagram of the false-positive rate against counter capacity with fixed N is shown on Figure 2:

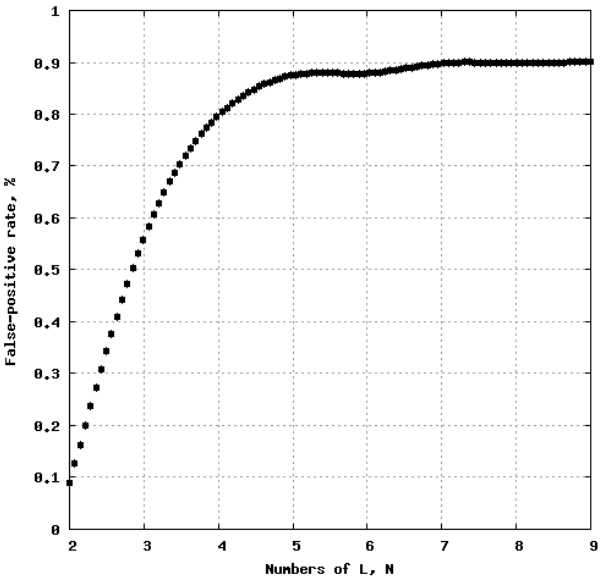

Fig. 2. Probability of false-positives against counter capacity

IV. IMPLEMENTATION OF THE MODIFIED BLOOM FILTER FOR HETEROGENEOUS SYSTEMS

The modified Bloom filter for hybrid systems was implemented in the form of a Java library in "jar" format [12]. The library includes methods of both sequential and parallel Bloom filter versions. This approach makes it possible to use the library in both conventional and heterogeneous systems, accordingly.

The parallel version of Bloom filter uses a CUDA co-processor to accelerate hashing operations [13]. CUDA-cores and Java-program communicate via jCuda library [14]. The parallel version supports three execution states:

- the Bloom filter is located in host memory of a computing system node, CUDA facility is used only for value hashing;
- the Boom vector is located in global memory of the CUDA device, temporary variables is located in "on-chip" shared and register memory (Figure 3). Verification of data existence in a vector is performed directly within the device (the state is only used in data search operations within a vector and is engaged upon request of a user);
- test mode (It allows to define a "threshold" value of the queries packet length. Starting from this value using of the CUDA-accelerator becomes reasonable).

The proposed implementation is well compatible with various NoSQL systems, the majority of which are realized in Java (or compatible with it). The library allows to perform initial testing of specific heterogeneous system. It is also useful for configuring the usage plan of CUDA-accelerator. This approach allows eliminating the penalties related to data transfer from primary memory to the memory of a CUDA facility and insufficient time period of system core operation under low system load.

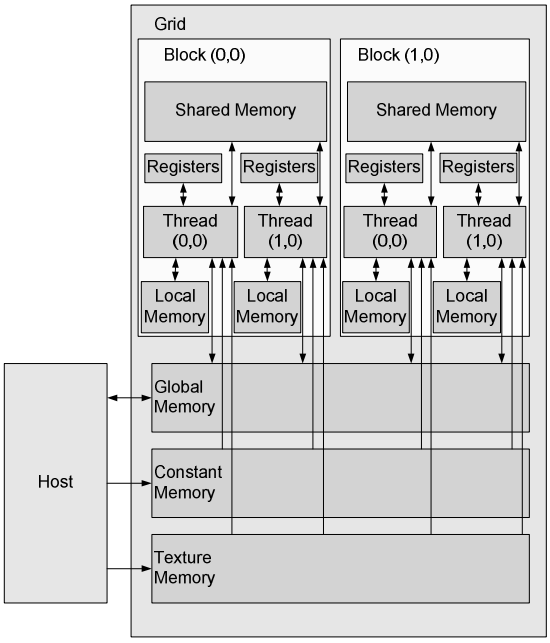

Fig. 3. CUDA memory hierarchy [13]

V. EFFICIENCY OF THE MODIFIED BLOOM FILTER

In order to reduce the false-positive rate, Bloom filter parameters were adjusted according to the known correlations [15]:

$$k = \frac{m}{n}\ln 2 \tag{7}$$

$$m = -\frac{n \ln p}{(\ln 2)^2} \tag{8}$$

Measurement of execution time of the sequential part of the program by CPU was performed with the use of System.nanoTime() function, which returns current timing value of an operating virtual Java machine with high accuracy, measured in nanoseconds. Time taken for execution is estimated by subtracting timing value before computation commencement from the timing value provided by the function after computation completion. Estimation of test time of program execution using a graphics processing unit is performed with the use of event API of CUDA. In CUDA, an event is a GPU time stamp, registered at a specific moment of time. Configuration of the testing system is shown in Table 1.

TABLE I. TESTING SYSTEM

Module	***Model***	***Specifications***
Central Processing Unit	Intel Core i7	2,6 GHz
Random Access Memory	8 Gb	DDR3
Graphics Processing Unit	GeForce GTX 260	Compute capability 1.3

The following (Figure 4) is a graph of the execution time depending on the hashing data size. T_CPU means the experiment results for the sequential version, T_CUDA – for the parallel. The diagram was obtained during the "library testing mode" execution.

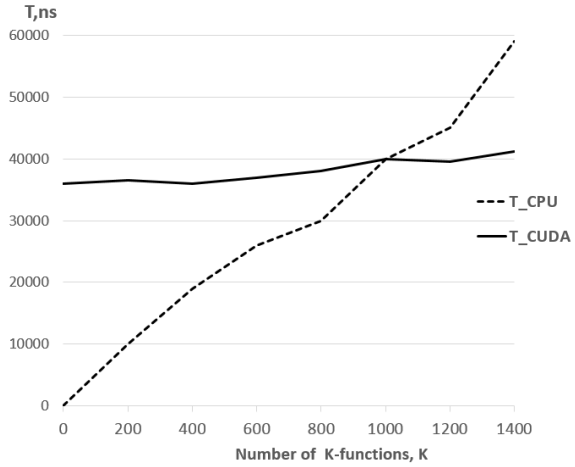

Fig. 4. Hashing time against data size

It should be noted that large data storages generally use long Bloom vectors, thus for one operation of adding an element into the filter there are many hashing operations. Observed diagram of dependence of probability of false-positives on the number of hashing operations is shown on Figure 5.

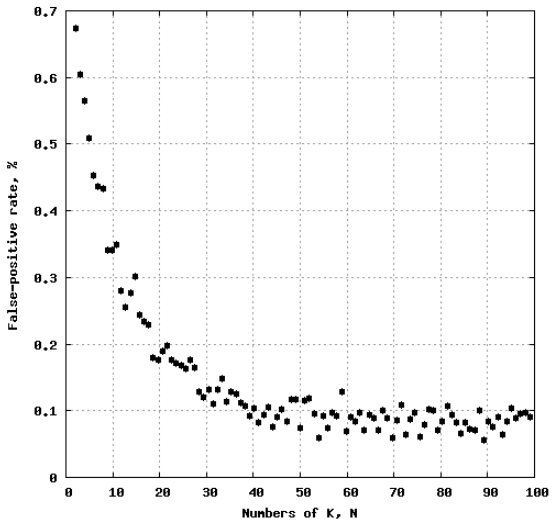

Fig. 5. False-positives probability against the number of hash functions

Thus, the modified Bloom filter not only makes it possible to solve the problem of data deletion, but also, it has the good heterogeneous system scalability properties.

This article also provides analytical and experimental estimations of the probability of false-positives of the modified Bloom filter, prompting the suggestion that increasing counter capacity not only leads to characteristic growth of memory penalties, but also increases the probability of false-positives manifold. Consequently while selecting parameters of a Bloom filter one needs to refer to both hardware restrictions imposed by GPGPU accelerator and to the lower-bound estimate of the probability of false-positive shown in the formula (6).

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] Pramod J. Sadalage, Martin Fowler NoSQL Distilled. — 1 edition — Addison-Wesley Professional, 2012. — P. 192

[2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels: "Dynamo: Amazon's Highly Available Key-Value Store". SOSP'07: Proc. Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, pages 205–220, 2007

[3] Dhruba Borthakur , Jonathan Gray , Joydeep Sen Sarma , Kannan Muthukkaruppan , Nicolas Spiegelberg , Hairong Kuang , Karthik Ranganathan , Dmytro Molkov , Aravind Menon , Samuel Rash , Rodrigo Schmidt , Amitanand Aiyer, Apache hadoop goes realtime at Facebook, Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, June 12-16, 2011, Athens, Greece

[4] Liu, Rong; Li, Qicheng; Li, Feng; Mei, Lijun; Lee, Juhnyoung, "Big Data architecture for IT incident management," Service Operations and Logistics, and Informatics (SOLI), 2014 IEEE International Conference on , vol., no., pp.424,429, 8-10 Oct. 2014

[5] Fay Chang, Jeffrey Dean, Sanjary Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber: "BigTable: a Distributed Storage System for Structured Data". *Proc. 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*. USENIX Association, 2006

[6] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol. 13, no. 7, pp. 422–426, 1970

[7] Vavrenyuk A.B., Vasilyev N.P., Makarov V.V., Matyukhin K.A., Rovnyagin M.M., Skitev A.A. Modified Bloom filter for high performance hybrid NoSQL systems. Life Sci J 2014;11(7s):457-461

[8] Lin Ma, Roger D. Chamberlain, Jeremy D. Buhler, and Mark A. Franklin, «Bloom Filter Performance on Graphics Engines», in Proc. of International Conference on Parallel Processing, September 2011, pp. 522-531

[9] Venkatesh Rao «Implementation of the Bloom Filter on GPU using CUDA», University of Minnesota, 2010

[10] Fan Zhang et al., «Fast lists intersection with Bloom filter using graphics processing units», in Proc. of 2011 ACM Symposium on Applied Computing, pp. 825-826

[11] Prosenjit Bose , Hua Guo , Evangelos Kranakis , Anil Maheshwari , Pat Morin , Jason Morrison , Michiel Smid , Yihui Tang, On the false-positive rate of Bloom filters, Information Processing Letters, v.108 n.4, p.210-213, October, 2008

[12] Herbert Schildt. Java The Complete Reference, 8th Edition, McGraw-Hill Osborne Media, pp.1152, 2011

[13] NVIDIA CUDA C Programming Guide 6.5 (official website). Date Views: 02.11.2014 https://docs.nvidia.com/cuda

[14] jcuda.org - Java bindings for CUDA (official website) URL: Date Views: 02.11.2014 http://www.jcuda.org/

[15] Paulo Sérgio Almeida , Carlos Baquero , Nuno Preguiça , David Hutchison, Scalable Bloom Filters, Information Processing Letters, v.101 n.6, p.255-261, March, 2007