Организация поиска данных в суперкомпьютерных системах на базе NoSQL-подхода и технологии NVIDIA CUDA

The organization of data search in supercomputer systems by using the NoSQL-based approach and NVIDIA CUDA technology

Васильев Николай Петрович
Ровнягин Михаил Михайлович

Аннотация: В статье рассматриваются некоторые вопросы, связанные с организацией хранения и поиска информации в многомашинных системах. Описываются основные подходы к решению проблемы хранения и обработки больших объемов данных. Предлагается архитектурно-независимый способ организации поиска информации на основе рандомизированного бинарного дерева и технологии Java. Предлагается способ повышения производительности системы за счет использования технологии NVIDIA CUDA.

Abstract: This paper discusses some of the issues related to the organization of the storage and retrieval of information in multicomputer systems. It describes the main approaches to solving the problem of storing and processing large amounts of data. It is proposed architecture independent way of organizing information retrieval based on a randomized binary tree and Java-technology. Provides a method of improving system performance using NVIDIA CUDA technology.

Ключевые слова: бинарное дерево, стохастическое преобразование данных, NVIDIA CUDA, JAVA

Keywords: binary tree, stochastic data transformation, NVIDIA CUDA, Java

Введение

В настоящее время для решения множества промышленных задач используются информационно-аналитические системы на основе классических SQL-решений. Однако, вследствие возрастания общего объема хранимой информации требуется все большее число вычислительных ресурсов для ее обработки Так, например, в одном из филиалов крупнейшего оператора мобильной связи в мире – китайской компании ChinaMobile - в 2006 году для предсказания поведения пользователей с помощью клиентской базы этого филиала использовалось коммерческое решение, включающее сервер с 8 процессорными ядрами, 32 Гб оперативной памяти и дисковым массивом хранения данных. Этот сервер был способен анализировать лишь 10% от получаемой ежедневно информации, и при этом имел стоимость порядка 1 млн. долларов США (с учетом стоимости ПО и расходов на сопровождение).

Для увеличения общей пропускной способности различных информационно-аналитических систем, а также ускорения процесса разработки распределенных приложений начиная с 2008 года крупнейшие мировые компании, такие как Yahoo, Last.fm, Facebook, TheNewYorkTimes используют технологию Hadoop [1]. Она базируется на основе концепции MapReduce в соответствие с которой данные хранятся на узлах вычислительного кластера распределенным образом, перераспределяясь во время вычислений, чтобы снизить число межузловых передач данных. В целом, можно говорить о том, что работает концепция перемещения обрабатывающего кода к данным, на не

наоборот, как это делалось ранее. Для хранения информации и обеспечения доступа к файлам применяется распределенная файловая система HDFS. Основным языком разработки параллельных приложений является Java. Применение технологии Hadoop позволило сократить расходы на наладку и поддержание работоспособности вычислительного кластера филиала ChinaMobile в 6 раз. Вместе с этим производительность Hadoop-решения оказалась почти на два порядка выше.

Для работы с большими объемами данных на современном этапе развития информационных технологий все чаще применяются NoSQL (not only SQL) решения, которые имеют следующие характеристики:

1. Нереляционная модель данных
2. Распределенность
3. Открытый исходный код
4. Хорошая горизонтальная масштабируемость.

Также NoSQL системы можно разделить по типам:

1. Поколоночные СУБД
2. Документо-ориентированные СУБД
3. Хранилища «ключ-значение», кортежные хранилища
4. Базы данных на основе графов

В отличии от декларативного описания необходимых данных в классических СУБД, в NoSQL-системах программист сам задает способ выборки данных и схему их хранения. Таким образом, отказываясь от традиционной реляционной модели, возрастает общая производительность системы, но, одновременно, появляется необходимость в описании методов работы с данными в формате «ключ-значение». Структуру хранения данных также должен задать программист. Обычно, применяются модификации классических структур данных: деревьев и хеш-таблиц. От того, насколько эффективно они реализованы, зависит общая производительность системы.

Современные проблемы организации поиска информации

Для хранения и поиска информации удобно использовать деревья поиска. Наиболее простым с алгоритмической точки зрения является бинарное дерево поиска. Основной проблемой с которой приходится сталкиваться в процессе работы с бинарными деревьями является проблема балансировки. Структура данного типа деревьев напрямую зависит от порядка добавления новых узлов. В худшем случае, при последовательном добавлении возрастающей/убывающей последовательности элементов двоичное дерево вырождается, превращаясь в связанный список (рис. 1c).

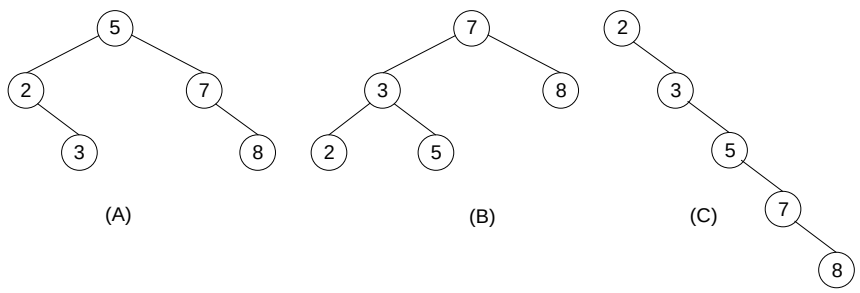


Рис. 1 Примеры двоичных деревьев поиска

Решается вышеизложенная проблема путем выполнения операций балансировки, при этом осуществляются «повороты дерева». Наиболее распространенными структурами данных, в которых определен механизм балансировки, являются АВЛ- и красно-черные деревья. Применяемые в этих случаях алгоритмы добавления/удаления/балансировки узлов позволяют приблизить сложность поиска к показателю $O(\log_2 n)$, где n-количество узлов дерева. Однако, возрастает также потребление ресурсов вычислителя по сравнению с аналогичными операциями в бинарном дереве.

Для хранения и поиска информации во внешней памяти применяются различные подвиды многоходовых B-деревьев. При этом, корневой узел храниться в системной памяти для оперативного поиска в нем нужной области дисковой памяти, в которой содержится следующий узел. Высота B-деревьев не велика, а основная проблема заключается, как правило, в задании эффективного механизма обхода дерева.

С появлением гибридных вычислительных комплексов большое внимание стало уделяться вопросам применения GPU для ускорения операций поиска в древовидных структурах. В статье [2] Jordan Fix и др. показали, что применение технологии CUDA для поиска в B+ дереве позволяет достичь более чем 10-кратного ускорения в операциях выборки. Но, если появляется необходимость в замене набора данных для поиска в памяти GPU на новый, такой подход оказывается неэффективным и приводит к многократному падению производительности по сравнению с CPU реализацией. Значительные накладные расходы появляются также вследствие необходимости синхронизации нитей CUDA.

Комплексное исследование производительности многопоточных CPU и GPU реализаций B-деревьев провели Changkyu Kim и др. в работе [3]. Для GPU-реализации удалось достичь 1.7-кратного роста производительности по сравнению с предыдущей GPU-версией алгоритма. Исследователи столкнулись с серьезными проблемами в тех случаях, когда деревья не помещались в памяти графического процессора. В основном, ускорения удалось достичь за счет привязки реализации к конкретной архитектуре. Размеры узлов дерева, их количество на уровне и способ передачи напрямую определялись архитектурными особенностями ускорителя GTX 280.

Экспериментальные результаты показывают, что из-за сильно ограниченного объема памяти GPU и большой латентности интерфейса передачи данных выигрыша в производительности удается достичь лишь в случае работы с редко меняющимися данными при условии, если программа хорошо адаптирована под конкретную разновидность GPU. С другой стороны, объем основной системной памяти, как правило, значительно превышает оный у GPU-ускорителя, а механизм многоуровневого кеширования информации в современных CPU позволяет получить существенный прирост производительности. Таким образом, создание архитектурно-независимой структуры для хранения, обработки и поиска данных является актуальной задачей.

Хранение и поиск информации в рандомизированном дереве

В [4] Т. Кормен и др. доказали, что математическое ожидание высоты случайного бинарного дерева поиска с n ключами равно $O(\log_2 n)$. Т.е. в результате вставки в дерево новых ключей в случайном порядке оно будет сбалансированным. Псевдослучайного характера добавляемых элементов можно добиться используя стохастическое преобразование поступающих данных. В таком случае, если применяется качественная функция преобразования, порядок следования ключей становится неважным. В рамках

данного исследования в качестве такой функции был выбран алгоритм ГОСТ 28147-89 (далее ГОСТ). Основной шаг преобразования изображен на рис. 2.

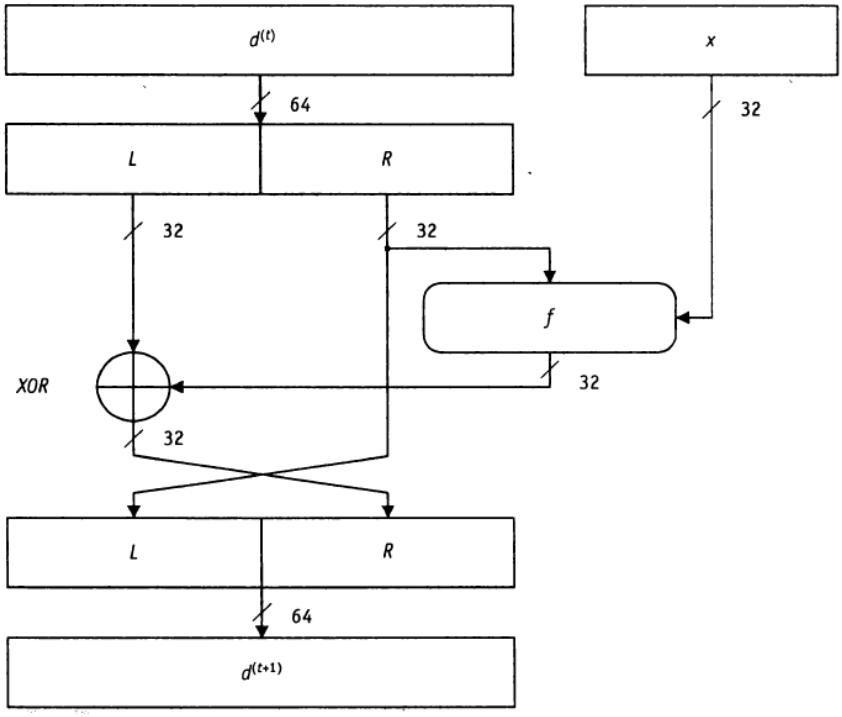


Рис. 2 Схема основного шага криптопреобразования ГОСТ 28147-89

ГОСТ является блочным алгоритмом шифрования на основе сети Фейстеля. В стандарте [5] определены четыре режима работы:

- простой замены
- гаммирование
- гаммирование с обратной связью
- режим выработки имитовставки

Режим выработки имитовставки предназначен для проверки отсутствия факта наличия в шифротексте случайных или преднамеренных искажений, т.е. не является режимом шифрования в общепринятом понимании. Особенности использования оставшихся трех режимов рассматриваются далее. Стоит особо отметить, что вместо ГОСТа может применяться любой алгоритм стохастического преобразования данных.

Для построения системы хранения данных, основанной на предлагаемом способе логично использовать ООП подход. Подобное решение придаст системе дополнительную «гибкость» и упростит ее настройку под требования конкретного заказчика (может меняться как алгоритм преобразования, так и структура хранения). Система должна быть также и архитектурно-независимой, для упрощения ее развертывания на различных аппаратных платформах. Одной из технологий, соответствующий этому требованию, является Java, основой которой является использования виртуальной машины Java (JVM) для запуска приложений. Таким образом, приложение может работать на любой платформе для которой существует версия JVM.

Принципиальная схема системы хранения данных, основанной на вышеописанном подходе изображена на рис. 3. На вход подаются данные, подлежащие добавлению, в формате «ключ-значение». Над ключами выполняется преобразование по функции f,

после чего рандомизированный ключ добавляется в дерево вместе с метаинформацией о сохраняемых данных (например, их размер и адрес во внешней памяти). При этом, дерево содержится в оперативной памяти, а сами данные могут храниться во внешней памяти.

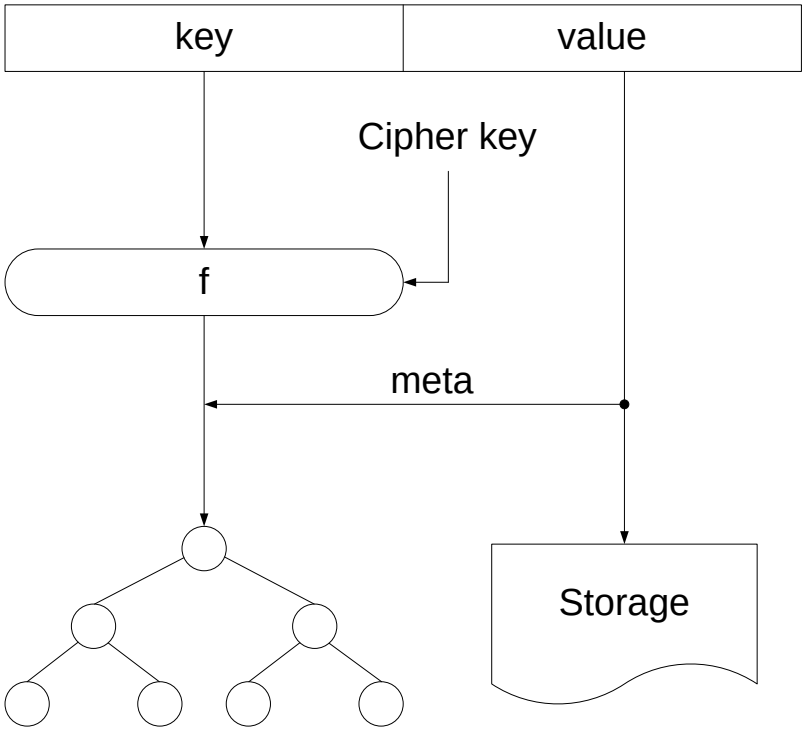


Рис. 3 Схема системы хранения данных на основе рандомизированного дерева

Применение технологии CUDA для повышения пропускной способности

Если в качестве f используется криптографическая функция, общая производительность системы будет низкой. Для повышения пропускной способности предлагается использовать гибридные вычислительные технологии: CUDA [6] или ATI Stream Technology.

В этом случае GPU может выполнять операции, связанные с вычислением функции f, разгрузив тем самым CPU для задач работы с памятью. Для языка Java разработана свободно-распространяемая библиотека, обеспечивающая удобный API вызова GPU-функций, под названием jcuda [7].

Распараллеливание ГОСТа, как и любого блочного шифра возможно осуществить лишь для некоторых режимов. В режиме простой замены отсутствуют зависимости между операциями шифрования блоков открытого текста, поэтому распараллеливание возможно. В режиме гаммирования гамма накладывается независимо на различные блоки открытого текста. Сам процесс выработки гаммы представляет собой суммирование по модулю 2^{32} и 2^{32}-1 содержимого соответствующих полублоков с константами. Данная рекурсия легко преобразуется к параллельному виду. Таким образом, режим гаммирования пригоден для реализации на параллельном вычислителе. В режиме гаммирования с обратной связью гамма формируется на основе предыдущего блока зашифрованных данных. Этот режим не распараллеливается.

Одним из основных требований к функции f является обеспечение взаимно однозначного отображения. Соответственно, при добавлении и поиске одинаковых ключей должна использоваться одинаковая ключевая информация для функции f. Таким образом, все

манипуляции с данными должны совершаться с применением одного и того же ключа шифрования.

В режиме гаммирования помимо ключа шифрования одним из параметров f является синхропосылка. Для выполнения требования взаимно однозначного отображения синхропосылка при доступе в систему за идентичными данными должна быть одинаковой. Однако, в таком случае при добавлении в хранилище различающихся данных будет генерироваться одна и та же гамма. Отличие между блоками шифротекста будет обусловлено лишь различием самих данных, что неприемлемо. Отсюда следует, что режим гаммирования не обеспечивает рандомизации ключей, при добавлении их в дерево. Таким образом единственным подходящим режимом ГОСТа для реализации вышеописанной системы хранения данных может быть режим простой замены.

Аналогичные рассуждения справедливы, если рассматривать в качестве функции f американский стандарт AES. В нем режим ECB соответствует режиму простой замены, Counter - гаммированию, а CFB - гаммированию с обратной связью.

Подходы к реализации параллельного алгоритма (например, AES) могут отличаться. Главным фактором, определяющим вид параллельного алгоритма, является гранулярность параллельной обработки. Важно сразу определить: сколько байтов открытого текста будет шифроваться каждой нитью. В классических параллельных реализациях это может быть 16, 8, 4, 1 байт.

Распараллеливание по 16 байтов на нить означает, что каждая нить обрабатывает свой фрагмент исходного текста, независимо от других. Подобная реализация избавляет разработчика от необходимости синхронизации и организации доступа к общей памяти из нитей, используя параллелизм, заложенный между блоками открытого текста.

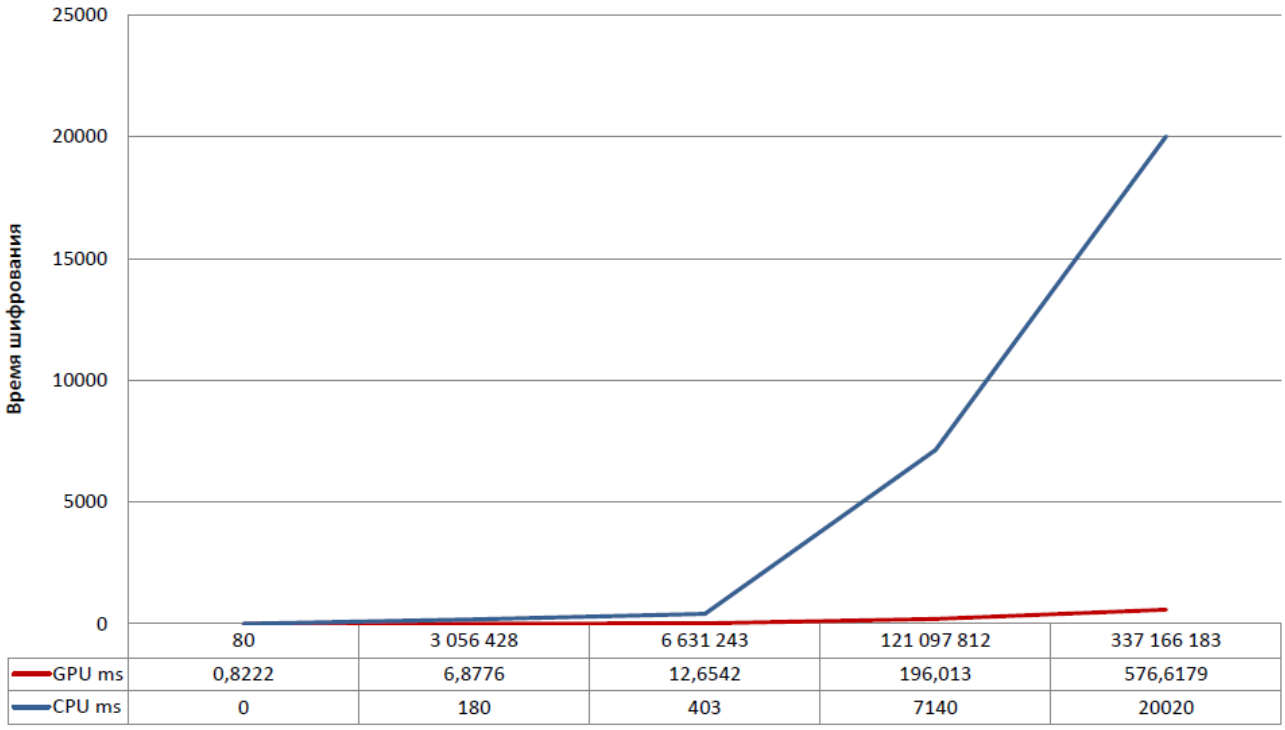
При гранулярности 8 байтов на нить один фрагмент исходного текста обрабатывается двумя нитями. Этот метод позволяет одновременно обрабатывать несколько блоков и выполнять операции по преобразованию блоков параллельно. Однако, такой подход требует использования разделяемой памяти и синхронизации для разделения промежуточных данных между процессами, что влечет дополнительные расходы. Гранулярность 4 байта на нить соответствует предыдущему случаю, но позволяет достичь большего параллелизма операций промежуточных преобразований.

Гранулярность один байт на нить предпочтительна для AES-шифрования на 32-битных обработчиках, но при использовании этой концепции для реализации параллельного CUDA-алгоритма возрастает число конфликтов по доступу к разделяемой памяти.

В статье [8] авторы анализируют возможность применения этих подходов для организации параллельных операций AES-шифрования. В результате проведения серии экспериментов выяснилось, что для реализации AES на ускорителе GTX 285 гранулярность в 16 байт (блок полностью) является наиболее предпочтительной.

При реализации режима простой замены ГОСТ на CUDA гранулярность составляла 8 байт, т.е. каждая нить обрабатывала свой блок. Для хранения ключевой информации использовалась разделяемая память. Некоторые циклы алгоритма были «развернуты» для повышения производительности. Исследования параллельной реализации ГОСТ на CUDA проводились в рамках работы [9]. Испытания выполнялись на вычислительном узле следующей конфигурации: Intel Core i7 – 2600, 6Gb DDR3, NVIDIA GeForce GTX660 (2Gb GDDR5).

График зависимости времени шифрования в режиме простой замены от размера входных данных представлен на рис. 4.



	80	3 056 428	6 631 243	121 097 812	337 166 183
GPU ms	0,8222	6,8776	12,6542	196,013	576,6179
CPU ms	0	180	403	7140	20020

Рис. 4 График зависимости времени шифрования в режиме простой замены от размера входных данных

Для шифрования в режиме гаммирования характер зависимости аналогичный. В таблице 1 приведены показатели прироста производительности для режима простой замены по сравнению с последовательной CPU-реализацией.

Размер файлов, байт	Простая замена		
	GPU ms	CPU ms	Прирост раз
80	0,822	0	*нет*
3056428	6,878	180	***26,172***
6631243	12,654	403	***31,847***
121097812	196,013	7140	***36,426***
337166183	576,618	20020	***34,720***

Таб. 1 Прирост производительности по сравнению с последовательной реализацией

Как можно видеть, максимальный прирост производительности можно наблюдать для больших объемов обрабатываемых данных. Поэтому преобразование ключей при добавлении в дерево должно происходить в пакетном режиме.

Применять GPU в качестве сопроцессора можно не только для рандомизации ключей, но и для зашифрования добавляемых данных в целях повышения безопасности системы.

Основные результаты и дальнейшие исследования

Для проверки работоспособности прототипа системы хранения данных была разработана тестовая программа, которая подавала на вход хранилища данные в формате «ключ-значение». Ключи представляли собой 64-битные значения. Добавление новых узлов в дерево происходило по наихудшему сценарию. На вход системы подавалась возрастающая последовательность ключей. В результате работы функции преобразования на основе ГОСТа происходила рандомизация. Дерево имело сбалансированный вид, а его высоту можно было аппроксимировать как $2{,}26*\mathrm{Log}_2 N$ [10].

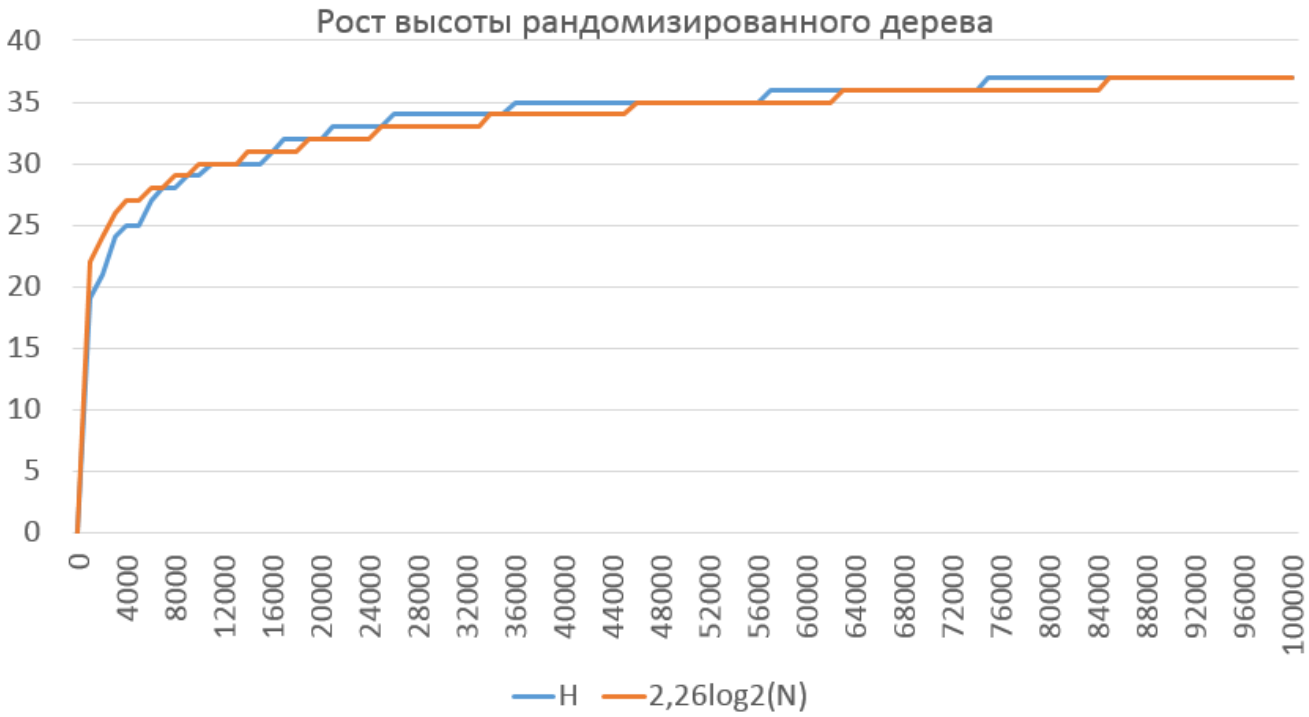


Рис 5. Зависимость высоты рандомизированного дерева от количества узлов

Дальнейшего изучения требует вопрос, связанный с распараллеливанием работы с памятью системы и шифрованием данных. Использование нитей java для работы с метаданными хранилища и одновременным сопровождением потоков CUDA выглядит наиболее перспективным вариантом.

Заключение

Предложенная система хранения данных является хорошо масштабируемой. В зависимости от конфигурации вычислительного узла может поддерживаться несколько деревьев, могут применяться более продвинутые алгоритмы шифрования, возможны доработки для более эффективной поддержки файлов данных во внешней памяти. Описанный подход является полностью архитектурно-независимым, если не планируется использование сопроцессоров для ускорения операций преобразования данных на входе. Использование NoSQL-системы на основе изложенного подхода видится эффективным в задачах интеллектуального анализа данных и высоконагруженных системах поиска различной информации. Многие алгоритмы обработки графической информации могут

быть интегрированы в систему наряду с функцией шифрования без видимых архитектурных проблем.

Литература

1. Ч. Лем Hadoop в действии: Пер. с англ., М.:ДМК Пресс, 2012

2. Jordan Fix, Andrew Wilkes, Kevin Skadron «Accelerating Braided B+ Tree Searches on a GPU with CUDA» A4MMC 2011 : 2nd Workshop on Applications for Multi and Many Core Processors, 2011

3. Changkyu Kim, Jatin Chhugani, Nadathur Satish et al, «FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs», ACM SIGMOD International Conference on Management of data, pp. 339-350, 2010

4. Т. Кормен и др., Алгоритмы: построение и анализ, 2-е изд.: Пер. с англ. - М.: Издательский дом «Вильямс», 2012

5. ГОСТ 28147-89 Алгоритм криптографического преобразования - 26 С. (официальный ресурс федерального агентства по техническому регулированию и метрологии) URL: http://protect.gost.ru/document.aspx?control=7&id=139177 (дата обращения: 14.09.2013).

6. А. В. Боресков и др. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учеб. пособие, М.:Издательство МГУ, 2012

7. jcuda.org - Java bindings for CUDA (официальный веб-сайт) URL: http://www.jcuda.org/ (дата обращения: 14.09.2013).

8. Keisuke Iwai, Naoki Nishikawa, Takakazu Kurokawa Acceleration of AES encryption on CUDA GPU International Journal of Networking and Computing 1(2), pp. 131-145, 2012

9. И. К. Тычинин Разработка программы для шифрования/расшифрования данных по алгоритму ГОСТ 28147-89 на основе технологии NVIDIA CUDA, дипломный проект, М.:МИФИ, 2013

10. Luc Devroye, Bruce Reed On The Variance Of The Height Of Random Binary Search Trees SIAM J. Comput Vol. 24, No. 6, pp. 1157-1162, 1995

Сведения об авторах
ФИО: Васильев Николай Петрович
Место работы и должность в настоящее время: НИЯУ МИФИ, доцент
Год окончания учебного заведения и его полное название: 1995 г. Московский инженерно-физический институт
Ученая степень и звание: к.т.н., доцент
Количество печатных работ и монографий: более 70
Область научных интересов: Облачные вычисления, гибридные суперкомпьютерные технологии, защита информации
Контакты: NPVasilyev@mephi.ru, +7(916)339-7937
Name: Vasilyev Nikolay Petrovich
Position: Associate Professor
Year of graduation: 1995
Academic degree: PhD
Number of publications and monographs: More than 70 publications

Research interests: Cloud computing, Hybrid supercomputer technologies, Information Security
Contact: NPVasilyev@mephi.ru, +7(916)339-7937

ФИО: Ровнягин Михаил Михайлович
Место работы и должность в настоящее время: аспирант, ассистент кафедры компьютерные системы и технологии НИЯУ МИФИ
Год окончания учебного заведения и его полное название: 2012 Национальный исследовательский ядерный университет «МИФИ»
Ученая степень и звание: нет
Количество печатных работ и монографий: более 10
Область научных интересов: Суперкомпьютерные технологии, организация поиска данных в высоконагруженных системах, разработка и верификация аппаратуры
Контакты: rovnyagin@gmail.com +7(915)2270086
Name: Michael M. Rovnyagin
Position: a postgraduate student, instructor in the Department of Computer Systems and Technologies NRNU MEPhI
Year of graduation: 2012 National Research Nuclear University «MEPhI»
Academic degree: no
Number of publications and monographs: More than 10 publications
Research interests: Supercomputer technology, data search in high load systems, development and verification of digital hardware
Contact: MMRovnyagin@mephi.ru +7(915)2270086