УДК 004.075

К.А. Рухлис, А.Е. Дорошенко

СРЕДСТВА ПЛАТФОРМЫ APACHE HADOOP ДЛЯ ПАРАЛЛЕЛЬНЫХ И РАСПРЕДЕЛЁННЫХ ВЫЧИСЛЕНИЙ

В связи с широким распространением Grid-систем для научных вычислений важной становится проблема выбора наиболее подходящей платформы для решения конкретных задач. Проблема усложняется ещё и тем, что тенденция роста объёмов данных для обработки имеет экспоненциальный характер. В работе рассмотрена одна из новейших Grid-платформ, реализующая новейшие методики в управлении вычислениями, ресурсами, отказоустойчивости и балансировки. Приведено сравнение основных компонент данной платформы с существующими аналогами и конкурирующими платформами.

Введение

Ещё недавно, гетерогенные вычисления значительно уступали гомогенным как по количеству существующих программных платформ так и по их качеству. Однако, рост количества различных электронных устройств, повышение производительности и функциональности устройств, а также значительное снижение их стоимости привели к выравниванию такого положения. Кроме того, развитию гетерогенных вычислений способствует существование и развитие платформы Java, как основы гетерогенных сред, маскирующей всё разнообразие базовых операционных сред и аппаратных решений.

Развитие Java платформ для распределённых гетерогенных вычислений пошло двумя путями. Первый из них связан с реализацией парадигмы клиент-серверных вычислений, приведшей к сервисно-ориентированным системам. Самым распространённым представителем этого направления стала система Globus [1]. Второй путь развития – это одноранговые (peer-to-peer, P2P) решения, где нет чётко выраженного серверного узла, а взаимодействие между двумя узлами кластера происходит непосредственно. Из наиболее популярных можно выделить платформы JXTA [2] и JGroups [3]. К недостаткам существующих платформ можно отнести их слаборазвитую инфраструктуру для отказоустойчивых вычислений и отсутствие адекватных средств для обработки больших объёмов данных. Это привело к некоторой стагнации в развитии платформ для гетерогенных вычислений, учитывая постоянный экспоненциальный рост количества обрабатываемой информации. Встроенных средств распределённой инфраструктуры, адаптированных к обработке многомерных массивов данных, средств отказоустойчивости, средств репликации и эффективного кеширования у существующих платформ либо нет, либо недостаточно. Особенно актуальной эта проблема стала для поисковых и аналитических систем Google и Yahoo.

Именно тогда компания Google создала новую платформу (и, собственно, подход к вычислениям) MapReduce [4], предназначенную для высокоэффективных параллельных вычислений на очень больших объёмах данных (несколько петабайт), распределённую файловую систему Google File System [5] и СУБД BigTable [6].

Подход MapReduce состоит из двух частей (рис. 1).

Во время выполнения вычислений первой части, Map, управляющий узел (master) обрабатывает исходные данные/задачи и производит разбиение с рассылкой подзадач/блоков данных на остальные, подчиненные, узлы (worker) кластера.

На этапе выполнения второй части, Reduce, управляющий узел получает результаты вычислений с подчиненных узлов и на их основе формирует общий

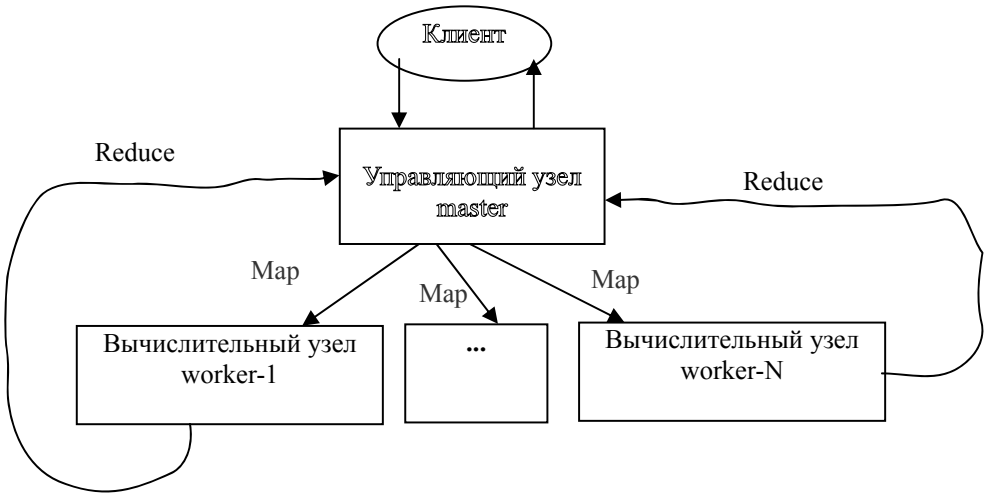

Рис. 1. MapReduce подход к вычислению задач

результат. Одна из особенностей данного подхода в том, что части Map/Reduce могут выполняться параллельно. Кроме того, в общем случае управляющих узлов может быть больше одного, что позволяет ускорить выполнение Map, и повысить отказоустойчивость.

При всей общей привлекательности подхода у этой систем есть два основных недостатка: проприетарность и возможность влиять на её развитие только у родительской компании Google, а также реализация на C++, что накладывает ограничения на переносимость платформы. Ответом Yahoo и Apache Foundation стала открытая реализация MapReduce парадигмы на Java, получившей название Apache Hadoop (http://hadoop.apache.org).

1. Структура Apache Hadoop

Платформа Apache Hadoop [7] – это развивающаяся и постоянно расширяющаяся платформа распределенных параллельных вычислений. Состоит из следующих компонент:

- Hadoop Common – ядро и утилитарные части для использования остальными подпроектами Hadoop;
- Avro – подсистема сериализации данных и интеграции с обработчиками сценарных (скриптовых) кодов;
- Chukwa – подсистема сбора данных для управления большими распределёнными системами;
- HBase – масштабируемая распределённая СУБД, конкурент Google BigTable;
- HDFS – распределённая высокоскоростная файловая система;
- Hive – инфраструктура для анализа данных;
- MapReduce – каркас (framework) для распределённой обработки данных на кластере;
- Pig – платформа для анализа больших объёмов данных с использованием парадигмы MapReduce;
- ZooKeeper – подсистема, координирующая жизненный цикл кластера.

В основу разработки Hadoop были положены следующие основные критерии:

- масштабируемость – реализовано надежное хранение и обработка огромных объемов данных (петабайты);
- гетерогенность и дешевизна – физическая инфраструктура для Hadoop представляет собой гетерогенный кластер; большая часть таких кластеров реализуется на публично доступном оборудовании, а каждый такой кластер может состоять из тысяч узлов;
- эффективность – эффективное распределение данных как основа для высокопроизводительных вычислений;
- надежность – использование любых доступных методологий для отказоустойчивого хранения данных;
- независимость от платформы – любая программно-аппаратная среда, для которой существует виртуальная машина

JVM, подходит для объединения в кластер Hadoop.

Система Hadoop включает в себя средства как для реализации обработки данных (Data Grid), так и высокопроизводительных вычислений (Computational Grid), что является одной из ключевых ее особенностей. Кроме того, в отличие от большинства существующих Grid-платформ, предназначенных либо для реализации Data Grid, либо для Computational Grid, либо для создания инфраструктуры, Hadoop объединяет в себе все эти средства. Это позволяет рассматривать Hadoop как универсальное законченное решения для вычислительных задач на больших объёмах данных, где важнейшее значение имеет не только скорость и отказоустойчивость обработки данных, но и вопросы их хранения и распространения.

Основные компоненты Hadoop – MapReduce, HDFS и HBase.

1.1. Hadoop MapReduce [8] – реализация подхода MapReduce в Hadoop. Предназначен для организации вычислений на больших объёмах данных. Единицами вычислений в Hadoop являются задания (Job). В кластере обязательно существует минимум один узел JobTracker (любой узел), который выполняет функции управления работой остальных вычислительных узлов. В его функции входит управление выполнением заданий, планирование выполнения, мониторинг выполнения, перераспределение задач в случае отказа вычислительных узлов (Task Tracker). Кроме того, в Hadoop MapReduce входят средства для управления кешированием, сжатием промежуточных данных, балансировкой нагрузки в кластере, отладки, выстраивание задач в очереди.

1.2. Hadoop HDFS [9, 10] – второй по важности компонент Hadoop, представляющий собой распределённую файловую систему. Её структура показана на рис. 2.

HDFS использует парадигму ведущий/ведомый (master/slave). Сам кластер состоит из:

1) одного ведущего серверного узла NameNode (master-server), управляющего файловой системой и доступом клиентов к файловой системе. К функциям NameNode относятся также распределение размещения файлов по файловой системе и работа с метаданными. Следует учесть, что в обмене данными конкретных файлов между хранилищем и клиентом NameNode непосредственно не участвует;

2) узлов данных DataNode, которые используются собственно для физического хранения конкретных файлов. Следует учитывать, что файл HDFS логически состоит из блоков, а DataNode хранят физи-

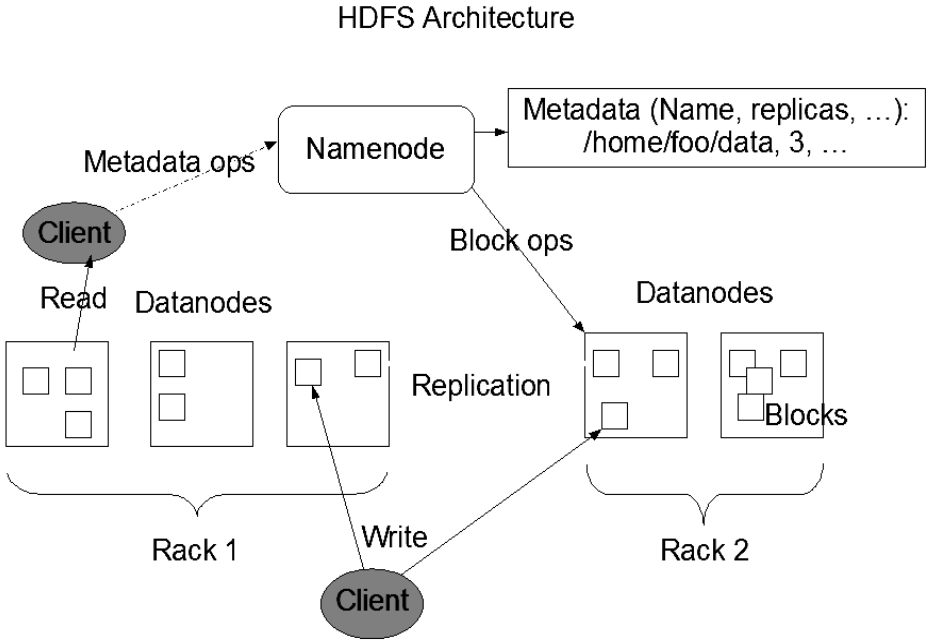

Рис. 2. Архитектура Hadoop HDFS

чески и обеспечивают операции чтения/записи/создания/удаления/репликации. Таким образом, файл HDFS физически может храниться блоками на нескольких узлах DataNode. Обычно, количество узлов DataNode равно количеству узлов кластера Hadoop;

3) клиентов – любых приложений, которым нужен доступ к определённому файлу/каталогу HDFS.

Пространство имён HDFS – иерархическое, где любой клиент может проводить операции создания/удаления/чтения/записи как каталогов, так и файлов. Отказоустойчивость обеспечивается репликацией блоков в нескольких экземплярах на разных узлах. Алгоритмы репликации и количество копий настраиваются в Hadoop. Кроме того, в HDFS реализован механизм отложенного удаления или удаления «в два этапа»: после запроса на удаление файл перемещается в каталог **/trash** и хранится там определенный период времени. Именно в этот период времени пользователь может восстановить его.

Кроме того, для обеспечения отказоустойчивости каждый DataNode раз в определенный период времени посылает сигнал NameNode, сигнализирующий NameNode, что этот узел «жив». В случае неполучения сигнала NameNode помечает конкретный DataNode как «мёртвый» и реплицирует все хранившиеся на нём блоки на другие узлы.

1.3. Hadoop HBase [11] представляет собой распределённое хранилище данных, реализующее парадигму BigTable. Данные организованы в таблицы. Но, в отличии от традиционной реляционной модели и существующих реляционных СУБД, одна и та же ячейка таблицы может содержать одновременно несколько значений. Для различения этих значений используется временная метка добавления этих данных. Основа парадигмы BigTable, в противовес реляционной модели, – это использование понятия Map (ассоциативного массива), как абстрактного типа данных, определяющего множество уникальных ключей, каждому из которых соответствует одно или несколько значений.

Физически таблица HBase может быть представлена следующим образом: **Map<byte[], Map<byte[], Map<byte[], Map<Long, byte[]>>>>.**|

Первый Map отображает ключи строк к семействам столбцов, второй – семейства столбцов на ключи столбцов, третий – ключи столбцов на временные метки, а последний – временные метки на конкретные значения ячеек. В качестве типов данных для ключей обычно используют строки. Временная метка хранится в целочисленном типе long, а значение колонки – как массив байтов. Ключ столбца всегда описывается в комбинации с его семейством: *family:key*. Для получения конкретного значения ячейки необходимо знать три координаты: ключ строки, ключ столбца, временная метка.

Вторым отличием HBase от обычных реляционных СУБД является её принадлежность к классу нереляционных распределенных (NoSQL) СУБД. Этот класс предназначен для решения основных проблем реляционных СУБД: масштабируемости, балансировки нагрузки на узлы, обеспечения эффективного и отказоустойчивого механизма хранения данных, линейного увеличения (в худшем случае) количества дополнительных обменов данными для распределённой транзакции и усложнения инфраструктуры для поддержки распределённых транзакций. Собственно, одной из нереляционных СУБД и является HBase.

К основным характеристикам HBase относятся [11, 12]:

1) использование парадигмы Map для представления данных;

2) распределённость – данные распределены по кластеру для ускорения, отказоустойчивости и балансировки нагрузки на узлы; постоянно используется репликация и оптимизация по ряду параметров и кэширование наиболее часто используемых данных;

3) отсортированность – характерная черта HBase, все пары ключ-значение хранятся в отсортированном порядке;

4) многомерность – одна и та же ячейка таблицы может содержать одновременно несколько значений; размерности ячеек

не обязаны совпадать;

5) разряжённость данных (sparse) – любая строка может иметь любое количество столбцов в семействе или не иметь их вовсе. Количество столбцов в двух разных строках независимо друг от друга;

6) сохраняемость – данные, которые были сохранены, остаются в базе до тех пор, пока не будут оттуда удалены соответствующим запросом. Ни перезагрузка/остановка серверов, ни сетевые проблемы не приведут к потере сохранённых данных.

2. Сравнение некоторых возможностей и производительности компонент Hadoop с конкурирующими платформами

Собственно Hadoop – уникальный продукт, объединяющий в себе средства для реализации распределенной обработки данных (Data Grid) , высокопроизводительных вычислений (Computational Grid) и обеспечения инфраструктуры. Поскольку конкурирующих продуктов с таким набором интегрированных подсистем фактически нет, то сравнивать можно только конкретные подсистемы.

Одной из основных подсистем Hadoop, фактически реализующей принцип MapReduce подход для построения высокопроизводительных вычислений, является Hadoop MapReduce. Его основным кросс-платформенным конкурентом, основанным на Java и также реализующий MapReduce парадигму явлется система GridGain (http://www.gridgain.com/). К преимуществам этой системы перед Hadoop MapReduce с функциональной точки зрения можно отнести:

1) автоматический поиск и обнаружение узлов в кластер;

2) автоматическую установку и развёртывание программ на узлы;

3) встроенную поддержку интеграции со всеми распространёнными прикладными программными интерфейсами и известными серверами приложений Java, таких как: JBoss, Spring, Spring AOP, JBoss AOP, AspectJ, JGroups, Coherence, GigaSpaces, JXInsight, Weblogic, Websphere;

4) развитый прикладной программный интерфейс для добавления собственных поставщиков услуг.

К преимуществам Hadoop MapReduce относятся более развитые средства отказоустойчивости, балансировки, мониторинга и отладки процесса выполнения, наличие возможности компрессии данных во время информационных обменов.

2.1. Сравнение производительности HBase/HDFS и производительности DBMS-X. DBMS-X – одна из самых распространённых параллельных реляционных СУБД с языком запросов SQL. Также как и HBase, эта СУБД использует репликацию данных для ускорения доступа и повышение отказоустойчивости. Для ускорения доступа к данным внутри таблиц, после репликации DBMS-X производит индексацию и сортировку данных по различным атрибутам и метаатрибутам. Хранение осуществляется по «строчному» принципу, в отличие от HBase. Основной сценарий использования и оптимизация этой СУБД заключается в применении её в режиме чтения/добавления данных, в чем назначение HBase и DBMS-X и пересекаются.

Одной из отличительных особенностей хранения данных DBMS-X является её способность сжимать таблицы данных, при этом в ряде случаев достигается 40 % увеличение производительности операций с БД в режиме чтения. Единственная проблема – то, что система недостаточно интеллектуальна, чтоб определить текущий способ использования того или иного экземпляра СУБД. В результате, если включена служба репликации DBMS-X либо же СУБД используется в режиме постоянного добавления-модификации-удаления данных, то компрессия данных приводит к потери производительности по сравнению с несжатым контентом.

Физически тестовая система представляет собой гетерогенный кластер из 250 узлов, с ЦПУ различной архитектуры и возможностей в большом диапазоне архитектур, от Pentium 4 3GHz до Intel Core 2

Quad Q9650 3.0GHz/12MB/1333MHz, объём ОЗУ от 3ГБ до 16ГБ на узел. Тесты состоят из следующих частей:

1) начальная инициализация БД с тремя наборами данных: 0.5GB, 1GB, 2GB на узел;

2) сохранение данных в БД отдельными транзакциями;

3) выполнение запросов на чтение к БД;

4) проверка отказоустойчивости.

Начальная инициализация БД в DBMS-X состоит из собственно изначальной загрузки дампа данных и, запускаемый вручную, этап последующего сжатия, индексирования загруженных данных. Кроме того, процесс репликации в DBMS-X является очень медленным, особенно, если сравнивать его производительность с производительностью подсистемы репликации HBase/HDFS.

При увеличении объёмов данных на узле, время, необходимое для инициализации БД DBMS-X растёт фактически линейно. При этом, HBase/HDFS фактически не тратит время на репликацию на «логическом уровне», как это делает DBMS-X, а использует средства репликации файловой системы HDFS.

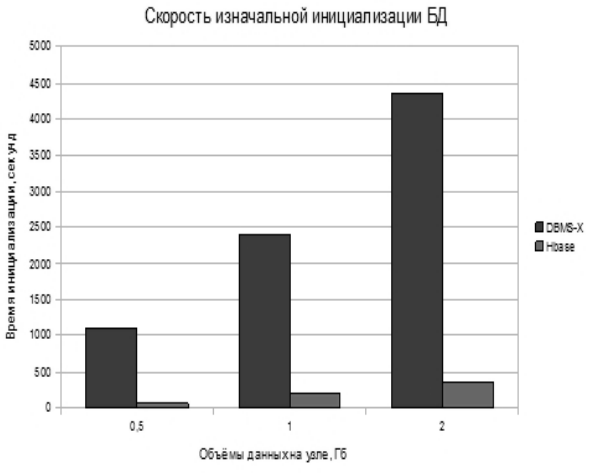

Рис. 3. Скорость изначальной инициализации БД

Сохранение данных в БД отдельными запросами – тест инициализации БД, только вместо загрузки данных одним дампом, каждая запись загружается в БД отдельным запросом. Ситуация повторяет начальную инициализацию БД, подсистемы репликации/индексирования DBMS-X не рассчитаны на такой сценарий работы и проигрывают методу «грубой силы» HBase/HDFS. Фактически, HBase/HDFS тратит время только на то, чтоб размещать новую запись в распределённом кеше, время от времени сохранять дамп блока на диск и осуществлять его репликацию по узлам.

Выполнение запросов к БД представляет собой два теста. В первом тесте осуществляется выборка данных из одной таблицы, обработка их бизнес-логикой с последующей выборкой данных из второй – каждым вычислительным узлом. Это один из вариантов повсеместного использования реляционных БД, Вторым тестом является поиск и сбор статистических данных по вхождению определённых текстовых шаблонов в текстовые поля таблиц. Это достаточно распространённая задача, к примеру, подсчёт статистических данных по ссылкам у поисковых систем. Результат показан на рис. 4.

Результаты тестов оказались достаточно интересными. Как видно из графиков, при выполнении теста 1, т. е., фактически, последовательности простых запросов SELECT, классическая реляционная СУБД DBMS-X оказалась в худшем случае в пять раз быстрее чем HBase (4 минуты против 20), в лучшем случае почти в 7 раз быстрее чем HBase (10.8 минут против 68.4 минут). Основная причина такого «провала» HBase – использование реляционной СУБД в идеальных для себя условиях, когда все запросы возможно выполнять используя заранее построенные индексы и сжатые таблицы данных. HBase фактически приходилось делать полный просмотр таблиц данных для отбора той или иной записи. Это особенно чётко видно, при увеличении объёмов обрабатываемых данных с 0.5ГБ на узел до 2ГБ на узел.

При этом время выполнения запросов DBMS-X увеличилось всего на 6.8 минут, а время выполнения запросов HBase – на 48.4 минут (!).

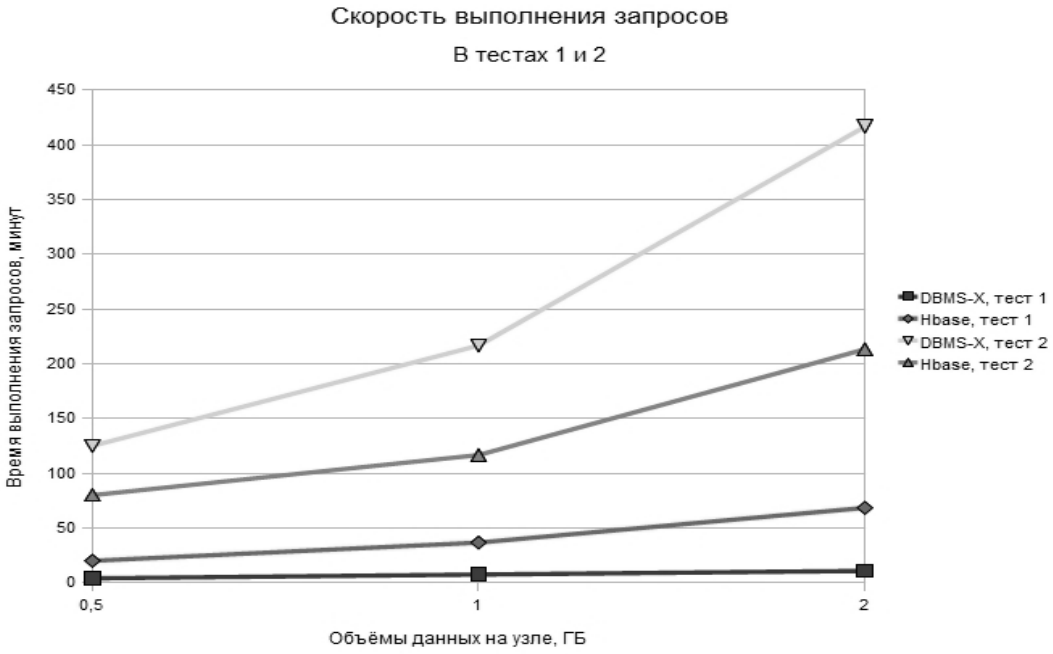

Рис. 4. Результаты теста 1 и теста 2 выполнения запросов

Также интересными являются и результаты второго теста, где DBMS-X уже не может использовать индексы, кроме того, ей приходится постоянно распаковывать данные, производить лишние обмены данными между узлами и т. д.

С ростом объёмов обрабатываемых данных эти недостатки становятся всё более явными. Архитектура HBase и находящаяся в основе HDFS намного лучше подходит для подобного рода задач.

Проверка отказоустойчивости осуществлялась путём принудительного физического отключения половины узлов кластера во время проведения второго теста выполнения запросов к БД. Измерялось общее затрачиваемое время на прохождение теста. После этого проверялось динамическое подключение узлов в кластер и последующий запуск этого теста. Измерялось ускорение от подключения дополнительных узлов в кластер. Результаты показали, приблизительно на 10 % большую эффективность подсистемы обеспечения отказоустойчивости и балансировки нагрузки HBase/HDFS по сравнению с DBMS-X.

Заключение

Рассмотрена одна из новейших Grid-платформ Apache Hadoop. Описаны её основные компоненты. Приведён практический пример использования этой платформы и сравнения эффективности её работы с конкурирующими платформами. В процессе сравнения выявлен ряд недостатков компонента СУБД HBase по сравнению с «традиционной» параллельной реляционной СУБД. Также описано, что вычислительной подсистеме Hadoop MapReduce не хватает средств P2P основанных платформ. Соответственно, имеется необходимость в дальнейшем улучшении основных компонент: MapReduce и HBase.

1. *Berman F., Fox G., Hey T. (eds)*. Grid Computing: Making the Global Infrastructure a Reality. Chichester: John Wiley & Sons, 2003. – 1060 p.
2. *Sun* JXTA, http://jxta.org
3. *JGroups*, www.jgroups.org
4. *MapReduce*: Simplified Data Processing on Large Clusters, http://labs.google.com/papers/mapreduce.html

5. *The* Google File System, http://labs.google.com/papers/gfs.html
6. *Bigtable*: A Distributed Storage System for Structured Data, http://labs.google.com/papers/bigtable.html
7. *T. White*, Hadoop: The Definitive Guide, Sebastopol: O'Reilly Media, Inc,2009.
8. *Hadoop* MapReduce Documentation, http://hadoop.apache.org/common/docs/current/
9. *Hadoop* Distributed File System Documentation, http://hadoop.apache.org/common/docs/current/hdfs_user_guide.html
10. *HDFS* Architecture, http://hadoop.apache.org/common/docs/current/hdfs_design.html
11. *HBase* Documentation, http://hadoop.apache.org/hbase/docs/r0.20.2/
12. *Understanding* HBase and BigTable, http://jimbojw.com/wiki/index.php?title=Understanding_HBase_and_BigTable .

Получено 03.03.2010

Об авторах:

Рухлис Константин Александрович,
младший научный сотрудник,

Дорошенко Анатолий Ефимович,
доктор физико-математических наук,
профессор, заведующий отделом.

Место работы авторов:

Институт программных систем
НАН Украины,
03187, Киев-187,
Проспект Академика Глушкова, 40.
Тел.: 0503122617,
e-mail: ukkr@yandex.ru